Università degli Studi di Camerino

School of Advanced Studies

Dottorato di Ricerca in Scienze e Tecnologie
Computer Science - XXXI Ciclo

Formalisation of BPMN Models: a Focus on Correctness Properties

Relatore

Prof. Flavio Corradini

Co-Relatore

Prof. Francesco Tiezzi

Dottorando

Chiara Muzi

Commissione Esaminatrice

Prof. Thomas T. Hildebrandt

Prof. Alberto Lluch Lafuente

Anno Accademico 2018-2019

University of Camerino

School of Advanced Studies

Doctor of Philosophy in Sciences and Technology

Computer Science - XXXI Cycle

Formalisation of BPMN Models: a Focus on Correctness Properties

Supervisor

Prof. Flavio Corradini

PhD Candidate

Chiara Muzi

Co-Supervisor

Prof. Francesco Tiezzi

Doctoral Examination Committee

Prof. Thomas T. Hildebrandt

Prof. Alberto Lluch Lafuente

Academic Year 2018-2019

To David:
"Hard work beats talent, when talent doesn't work hard"

Abstract of the Dissertation

The BPMN standard has a huge uptake in modelling business processes within the same organisation or involving multiple ones. Its primary goal is to create a standardised bridge for the gap between process design and process implementation in order to entirely automate the business process management lifecycle. In this context, the BPMN lack of a precise semantics represents a big issue, since it can lead to different interpretations, and hence implementations, of its features. Thus, providing a formal semantics of the BPMN notation is crucial for shaping IT systems and guaranteeing that these systems behave as they are supposed to do. On the one hand a formal semantics allows to overcome misunderstandings due to the use of natural language in the standard; on the other hand it enables the study of model properties, paving the way for correctness verification. This is indeed a clearly perceived need, as business process models, while primarily intended for process documentation and communication, are nowadays often also used as input for developing process-aware information systems and service-oriented applications. High quality of business process models is important for all these goals and correctness is an important indicator of it. Its relevance is also highlighted by the observation that incorrect models can lead to wrong decisions regarding a process and to unsatisfactory implementations of information systems.

Tackling the above issues, this thesis contributes to the definition of a direct BPMN operational semantics, providing a uniform formal framework to study BPMN models and their main correctness properties. The formalisation allows to formally characterise important properties in the business process modelling domain, namely well-structuredness, safeness and soundness, thus classifying BPMN models according to the properties they satisfy. This leads to achieve advances in classifying BPMN models, by directly addressing BPMN models and their specificities. Concerning the formalisation, first a formal semantics for a core subset of BPMN elements is provided and then it is extended, by including other elements and business perspectives.

Specifically, are considered features widely recognised as tricky and challenging to be formalised such as the OR-Join gateway, the sub-process element and multiple interacting participants, who exchange messages and data, in order to check if the obtained results are still valid in an extended framework.

Acknowledgements

Pursuing a PhD project is both a difficult and enjoyable experience. It's just like playing a tennis match, game after game, accompanied with bitterness, hardships, frustration, encouragement and trust and with so many people's kind help. When I reached this beautiful win, I realised that it was, in fact, teamwork that got me there. Though it will not be enough to express my gratitude in words to all those people who helped me, I would still like to give my many, many thanks to all these people.

The PhD process successfully ends with the thesis defence, that could be not possible with the feedback provided by the committee members, professor Thomas T. Hildebrandt and professor Alberto Lluch Lafuente; thank you for your brilliant comments and suggestions.

Before coming to an end a process has to start; this was possible thanks to the PhD admissions committee, in particular to professor Emanuela Merelli, coordinator of the PhD program, and professor Andrea Polini.

The PhD process continued thanks to the whole PROS Lab group that welcomed me and contributed immensely to my personal and professional time at Unicam. The group has been a source of friendships as well as good advice and collaboration.

In parallel, I would like to thank my supervisor professor Flavio Corradini for having the opportunity to learn from his research expertise and for the excellent example he has provided as a successful professor and rector of my university. Many thanks to my co-supervisor professor Francesco Tiezzi for his valuable guidance, inputs and support I received throughout the research work. He offered me so much advice, supervising me and always guiding me in the right "formal" direction. Many many thanks to professor Barbara Re, who encouraged me in undertaking this PhD programme. She was the first person who believed I could make the big jump from Mathematics to Computer Science. This was possible also because of her unconditional support with an amicable and positive disposition. She has always made herself available to clarify my doubts despite her busy schedules and she has always been present

with moral and material support, advice and encouragement, contributing in avoiding an *abnormally termination of the PhD process*!!! I am also deeply thankful to the *activity* performed by my colleagues of adventures, Fabrizio, Andrea and Lorenzo. They offered me constant support and cooperation and, most of all, they had considered me as the most beautiful woman of the office!!!!

A real process involves many other external participants, who can provided support by sharing experiences, *messages*, and love. Among these I have to thank my family, that always believed in me and was there for practical and moral support in all those things of life beyond doing a PhD.

Last but not least, I thank a very special person, my boyfriend, David for his continued and unfailing support and understanding, that contributed in making the thesis completion possible. I know you are the person who expresses feelings with actions rather than words: you were always around at times I thought that it is impossible to continue, you pushed and motivated me and most of all you have always made me happy!

List of Publications

- Corradini, F., Fornari, F., Muzi, C., Polini, A., Re, B., and Tiezzi, F. On Avoiding Erroneous Synchronization in BPMN Processes. 20th International Conference on Business Information Systems, BIS 2017, Poznan, Poland, Volume 288 of LNBIP Springer, pp. 106-119, 28-30 June 2017.

- Corradini, F., Muzi, C., Re, B., Rossi, L. and Tiezzi, F. Global vs. Local Semantics of BPMN 2.0 OR-Join. 44th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2018, Krems an der Donau, Austria, Volume 10706 of LNCS Springer, pp. 321-336, 29 January - 2 February 2018.

- Corradini, F., Muzi, C., Re, B., and Tiezzi, F. A Classification of BPMN Collaborations based on Safeness and Soundness Notions. Combined 25th International Workshop on Expressiveness in Concurrency and 15th Workshop on Structural Operational Semantics, EXPRESS/-SOS 2018, Beijing, China, Volume 276 of Electronic Proceedings in Theoretical Computer Science, pp. 37-52, 3 September 2018.

- Corradini, F., Muzi, C., Re, B., Rossi, L. and Tiezzi, F. Animating Multiple Instances in BPMN Collaborations: from Formal Semantics to Tool Support. 16th International Conference on Business Process Management, BPM 2018, Sydney, Australia, Volume 11080 of LNCS Springer, pp. 83-101, 9-14 September 2018.

- Corradini, F., Muzi, C., Re, B., Rossi, L. and Tiezzi, F. MIDA: Multiple Instances and Data Animator. 16th International Conference on Business Process Management, BPM 2018 (Demo), Sydney, Australia, Volume 2196 of CEUR Springer, pp. 86-90, 9-14 September 2018.

- Muzi, C., Pufhal, L., Rossi, L., Weske, M., and Tiezzi, F. Formalising BPMN Service Interaction Patterns. 11th IFIP WG 8.1 working con-

ference on the Practice of Enterprise Modelling Vienna, Volume 335 of LNBIP Springer, pp. 3-20, 31 October - 2 November, 2018.

Contents

List of Figures

List of Tables

Part I

Introduction & Background

Chapter 1

Introduction

Modelling activities is recognised as an important phase in the software life cycle. In particular, modelling business processes in complex organisations allows a better understanding on organisations work and, at the same time, it supports the development and the continuous improvement of related IT systems [69, 34]. A significant challenge is to provide a precise semantics of the used modelling languages to guarantee that the modelled behaviours do what they are supposed to do. This permits, not only to overcome misunderstanding due to the use of natural language in the modelling languages specification, but also enables formal reasoning on model properties, so that diagnostic information can be reported on the diagram in a way that is understandable by all process stakeholders. This is especially useful in the case of organisations involving different participants that need to properly interact.

The main objective of this thesis is to provide a uniform formal framework to characterise business processes and their main correctness properties. This permits to classify models according to the properties they satisfy, thus providing systematic methodological advices for modelling business processes in a correct way. This chapter introduces the motivations and objectives that have driven this work and describes the thesis structure.

1.1 Motivations

The effective and efficient handling of business processes is a primary goal of organisations. Business Process Management (BPM) provides methods and techniques to support these endeavours [109]. Thereby, the main artefacts are business process models which help to document, analyse, improve, and automate organisation processes. For conducting a successful business, an organisation does not act alone, but it is usually involved in collaborations with other organisations. Therefore, it is interesting to consider collaborative

systems where participants can interact and share information and data. To ensure proper carrying out of such interactions, the participants should be provided with enough information about the messages they must or may send in a given context.

In this regard, Business Process Model and Notation (BPMN) [68] collaboration diagrams result to be an effective way to represent how multiple participants cooperate to reach a shared goal. Even if widely accepted, BPMN has a major drawback related to the complexity of the semi-formal definition of its meta-model and the possible misunderstanding of its execution semantics defined by means of natural text description, sometimes containing misleading information [90]. These issues worsen when taking into account BPMN elements that have a particularly tricky behaviour and when considering BPMN supporting tools, such as animators, simulators and enactment tools, whose implementations of the execution semantics may not be compliant with the standard and be different from each other, thus undermining models portability and tools effectiveness.

To overcome these issues, several formalisations have been proposed, usually by means of mapping the BPMN notation to formal languages (e.g. Petri Nets [29], process calculi [113]). However, on the one hand models resulting from a mapping inherit constraints given by the target formal language and so far none of them considers specific and distinctive features of BPMN such as different abstraction levels (i.e., collaboration, process, and sub-process), the asynchronous communication model, and the notion of completion due to different types of end event (i.e., simple, message throwing, and terminate). On the other hand, less attention has been paid to provide a formal semantics capturing the interplay between control features, message exchanges, and data. This is a particularly important challenge in the BPM domain, as a precise semantics does not leave any room for ambiguity and is a prerequisite to ensure the appropriate carrying out, in practice, of processes and interactions among them. Moreover, a precise characterisation enables to formal reason on model properties at a level as close as possible to BPMN diagrams, so that the resulting findings are easily understandable by process stakeholders, and provide advices for modelling business process models in a correct way, avoiding errors in execution.

1.2 Research Objectives

The **main objective** of this thesis is to provide a formal characterisation of the BPMN semantics specifically given for collaboration models, with the aim of formally defining a classification of these models according to relevant properties of the business process domain. The intention is to introduce a unique formal framework to allow BPMN designers to achieve a better

understanding of their models, and relative properties. In order to deal with this objective, a set of **sub-objectives** has been established.

1. Provide a formal semantics for the core of BPMN elements by means of an Operational Semantics, specifically and directly given for BPMN, to enable the application of formal methods.

2. Consolidate the state of the art on business process correctness properties through their formal definition in a uniform formal framework.

3. Achieve advances with respect to already available classifications of BPMN models, by directly addressing BPMN collaboration models and their specificities.

4. Extend the formal framework by considering the OR-Join gateway, sub-processes, data objects and multiple interacting participants, who exchange messages and data, in order to show that the obtained results are still valid in an extended framework.

5. Validate the formal framework, both in terms of the considered BPMN elements and of the expected semantic behaviour.

1.3 Thesis Structure

The thesis is organized into four parts.

Part I - Introduction and Background is divided into two chapters. Chapter 1, *Introduction*, motivates the research work of this thesis, presenting its target objectives. Chapter 2, *Background*, provides background materials for the understanding of the thesis with concepts from the Business Process Management area, with a focus on relevant properties of the domain and some notions on formal methods techniques.

Part II - BPMN Formalisation and Correctness Properties is divided into two chapters that exploit a formal characterisation of the collaborations' semantics, specifically and directly given for core BPMN elements, to provide their classification. Chapter 3, *Core BPMN Operational Semantics*, presents the syntax and operational semantics defined for a relevant subset of BPMN elements and compares the given formalisation with existing approaches present in the literature. This permits to achieve sub-objective 1. Chapter 4, *BPMN Correctness Properties*, provides a rigorous characterisation, with respect to the introduced BPMN formalisation, of the key properties studied in this thesis: well-structuredness, safeness and

(message-relaxed) soundness. This permits to achieve sub-objectives 2 and 3.

Part III - Extended Framework is divided into four chapters that extend the BPMN formalisation and classification in order to reason on a more complete set of BPMN elements and validate the proposed semantics. Chapter 5, *OR-Join Gateway*, provides a global and local semantics of the OR-Join gateway, proves the correspondence between them and checks the correctness behaviour of models including the OR-Join gateway. Chapter 6, *Sub-Processes*, adds another abstraction level, the sub-process, and investigates how it can impact on the satisfaction of the considered properties. Chapter 7, *Multiple Instances and Data*, discusses and formalises the interplay between control features, message exchanges, and data in multi-instance collaborative scenarios. This allows to classify a wider set of models by taking into account also the data perspective. The contribution of these chapters permits to achieve the sub-objective 4. Finally Chapter 8, *Formalising BPMN Patterns*, visualises and formalises the workflow patterns expressed in BPMN, thus validating the provided semantics. This permits to achieve sub-objective 5.

Part IV - Conclusion and Future Work is formed by Chapter 9, *Conclusion and Future Work*, that summarises the work done and presents areas and topics that could be investigated in the future.

Chapter 2

Background

Background notions are set for the fully understanding of the dissertation content. In particular, BPM and BPMN 2.0 are introduced together with an overview of business process correctness properties. Finally, relevant notions about formal methods techniques are given, focussing on the process of model formalisation via an operational semantics.

2.1 Business Process Management

BPM is the set of all activities to support and improve organisation performance by managing chains of events, tasks, and decisions that ultimately add value to the organisation. It includes concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of business processes [109].

A Business Process is described as "*a collection of related and structured activities undertaken by one or more organisations in order to pursue some particular goal. Within an organisation a business process results in the provisioning of services or in the production of goods for internal or external stakeholders. Moreover 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*" [51].

In order to analyse, improve and enact business processes, one has first to identify business process goals and stakeholders, and design a business process model, that is a set of activities and execution constraints between them. This step represents the *Design Phase* of the business process life cycle [109, 110, 13]. During this phase the business process model is designed by a business process designer. Since process models are meant to facilitate communication between stakeholders they should be easy to understand. To this aim, modelling languages are used to design business process models, so

that different stakeholders can efficiently communicate, refine and improve the models.

In deriving a business process model many different information and perspectives of an organisation can be captured [85]. Among the others, the thesis focuses on: information related to the activities to be performed (function perspective), who should perform them (organisation perspective), when they should be performed and how they are organised in a flow (behaviour perspective) and finally which data are needed and produced (information perspective). These perspectives are shown in Figure 2.1 (that is a revised version of the figure in [85, Sec. 2.4]).

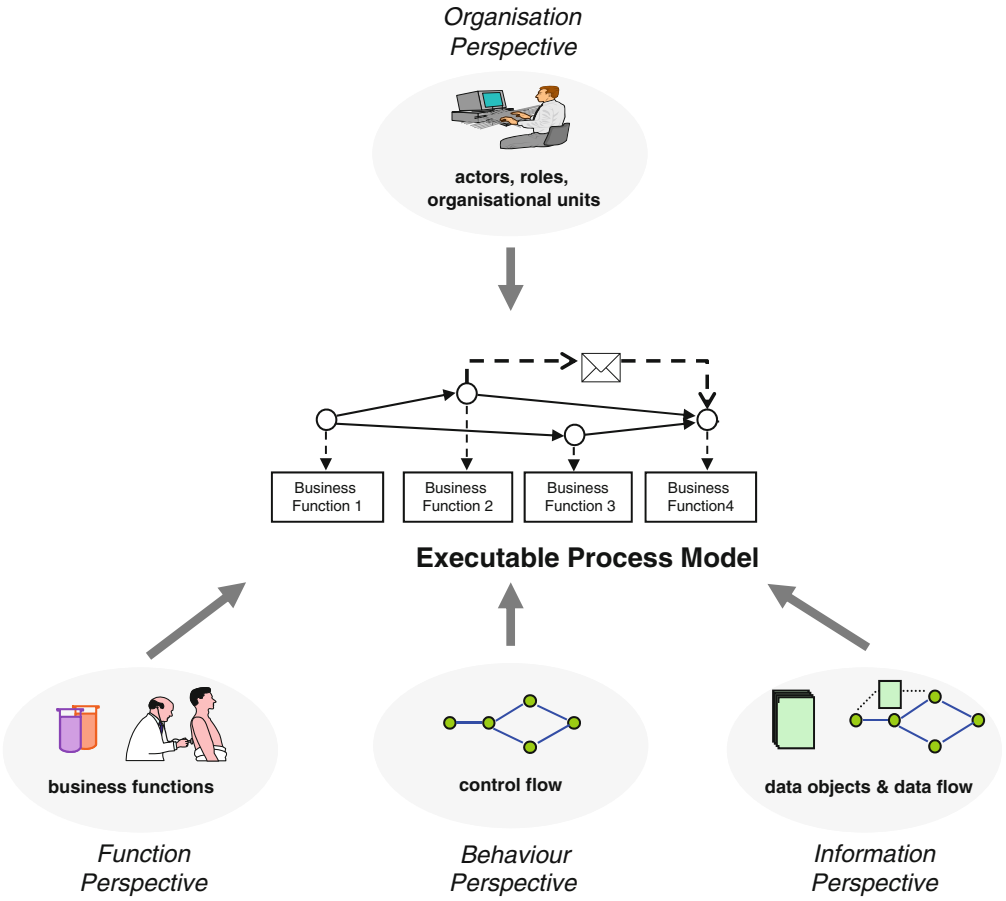


Figure 2.1: Business Process Perspectives [85, Sec. 2.4].

Considering the *behaviour perspective*, the dynamic behaviour of a process model comes into play. In this regard, it is necessary to verify that the business process model designed in the *design phase* works seamlessly. Analysis can be done to detect syntactic, structural and behavioural problems. This step constitutes the *analysis phase* in the business process life cycle. One

of its aim is to ensure correctness of the designed business process model, because an erroneous behaviour can cause high costs for the involved organisations and damage their reputation. In order to check for model correctness, the qualities a model has to exhibit have to be clearly defined. This step is highly recommended, and will be addressed in the thesis, to proceed with business process analysis and the other phases of the business process life cycle. This latter includes the *enactment and execution phase* (the execution of the business process by stakeholders) and the *monitoring and improvement phase* (the monitoring of the running business process instances in order to maintain the business process models updated respect to the reality), that are out of scope of this thesis.

2.2 Business Process Model and Notation 2.0

Many different languages and graphical notations have been proposed to represent business process models, differing both in the possibility to express aspects related to the organisation perspectives and in the level of formality used to define the elements composing the notation. BPMN 2.0 [68] is currently acquiring a clear predominance, also thanks to its capability to close the gap between IT and business teams. It has been standardised by the Object Management Group (OMG) and it is now widely accepted both in industry and academia. Its first goal is to provide a notation that is readily understandable by all business users. This includes the business analysts that create the initial drafts of the processes to the technical developers responsible for implementing the technology that will perform those processes.

Business process models are expressed in business process diagrams. Each diagram consists of a set of modelling elements. These elements are partitioned into a core element set and a complete element set. In particular, the BPMN notation allows to design different kinds of diagrams: process, collaboration, choreography and conversation diagrams. For a complete and detailed description of each BPMN diagram, please refer to the official BPMN Specification.[1] The thesis focuses on process diagrams, which are used to represent processes within a single organisation, and on collaboration diagrams, that model processes of different organisations exchanging messages and cooperating to reach a shared business goal.

Next sections describe the BPMN elements that will be considered in the thesis and then introduce a running example used to show the reader how to employ the BPMN notation and through the work as a running example to show the thesis results.

[1]BPMN Specification – https://www.omg.org/spec/BPMN/

2.2.1 BPMN Notation

This section illustrates the BPMN elements considered in this thesis which include: pools, activities, events, gateway, connecting edges and data objects.

- **Pools** (Figure 2.2) are used to represent participants or organisations involved in the collaboration, and include details on internal process specifications and related elements. Pools are drawn as rectangles, and they usually have a name associated with, referring to the name of the organisation. BPMN allows to assign a multi-instance marker (three vertical lines) to a pool, representing multiple instances playing the same role.

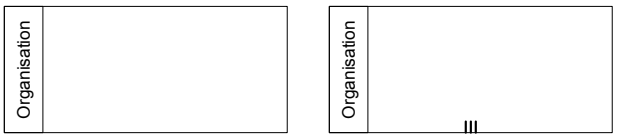


Figure 2.2: BPMN Pools.

- **Activities** (Figure 2.3) are used to represent a specific work to be performed within a process. They can be atomic or non-atomic (compound) and are drawn as rectangles with rounded corners. A *Task* is an atomic activity which represents work that can not be interrupted. It can be also used to send (*Send Task*) and receive (*Receive Task*) messages. It can have a multi-instance marker (three vertical or horizontal lines) when several activity instances are needed. This means that an activity is performed many times with different data sets. A *Sub-Process* a self-contained, composite activity that can be broken down into smaller units of work. A sub-process is collapsed when the details of the sub-process are not visible in the Diagram. A "plus" sign in the lower-center of the shape indicates that the activity is a sub-process and has a lower-level of detail. The sub-process is expanded when the details (a Process) are visible within its boundary.

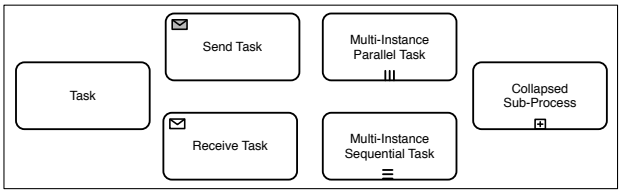


Figure 2.3: Considered BPMN Activities.

- **Connecting Edges** (Figure 2.4) are used to connect process elements inside or across different pools. *Sequence Edges* are solid connectors

used to specify the internal flow of the process, thus ordering elements in the same pool, while *Message Edges* are dashed connectors used to visualise communication flows between organizations [2].

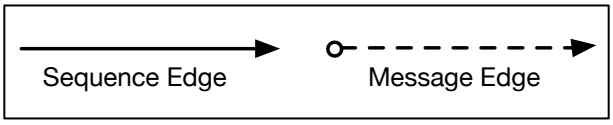


Figure 2.4: BPMN Connecting Edges.

- **Events** (Figure 2.5) are used to represent something that can happen. An event can be a *Start Event* representing the point from which a process starts, an *Intermediate Event* representing something that happens during process execution, or an *End Event* representing the process termination. Events are drawn as circles. When an event is source or target of a message edge, it is called *Message Event*. According to the different kinds of message edge connections, it is possible to distinguish between the following type of events.

  - *Start Message Event* is a start event with an incoming message edge; the event element catches a message and starts a process.
  - *Throw Intermediate Event* is an intermediate event with an outgoing message edge; the event element sends a message.
  - *Catch Intermediate Event* is an intermediate event with an incoming message edge; the event element receives a message.
  - *End Message Event* is an end event with an outgoing message edge; the event element sends a message and ends a process.

  There is also a particular type of end event, the *Terminate End Event*, displayed by a thick circle with a darkened circle inside; it stops and aborts the running process - all the ongoing activities are aborted and the process is abnormally terminated.

[2] As a matter of terminology, Sequence Edge and Message Edge are referred in the BPMN specification as Sequence Flow and Message Flow, respectively.

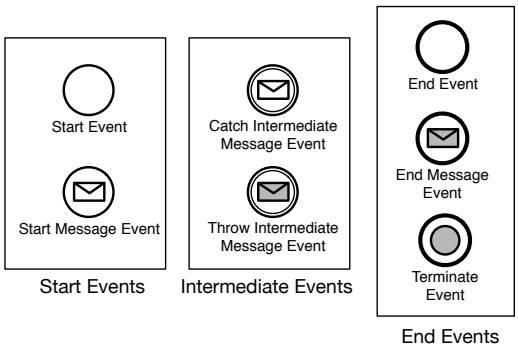


Figure 2.5: Considered BPMN Events.

- **Gateways** (Figure 2.6) are used to manage the flow of a process both for parallel activities and choices. Gateways are drawn as diamonds and act as either join nodes (merging incoming sequence edges) or split nodes (forking into outgoing sequence edges). It is possible also to express gateways with multiple incoming and multiple outgoing edges in BPMN. These gateways are called mixed gateways. Since two behaviours (split and join) are expressed by a single concept, best practice is not to use mixed gateways but to use a sequence of two gateways with the respective split and join behaviour instead. Different types of gateways are available.

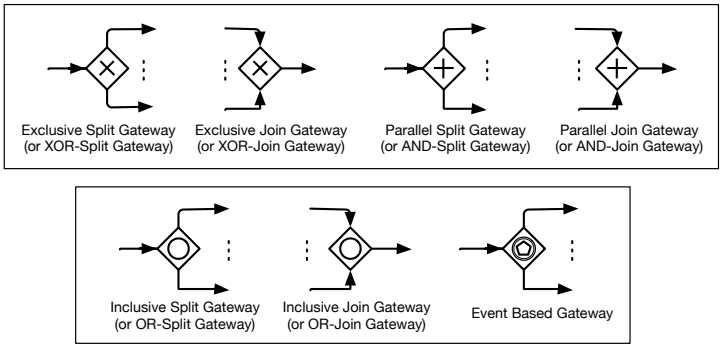


Figure 2.6: Considered BPMN Gateways.

  - An *Exclusive Gateway* (or *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. A XOR gateway is drawn with a diamond marked with the symbol "×".

  - A *Parallel Gateway* (or *AND gateway*) enables parallel execution flows. An AND-Split gateway is used to model the parallel execution of two or more branches, as all outgoing sequence edges are activated simultaneously. An AND-Join gateway synchronizes the execution of two or more parallel branches, as it waits for all incoming sequence edges to complete before triggering the outgoing flow. An AND gateway is drawn with a diamond marked with the symbol "+" .
  - An *Inclusive Gateway* (or *OR gateway*) gives the possibility to select an arbitrary number of (parallel) flows. In fact, an OR-Split gateway is similar to the XOR-Split one, but its outgoing branches do not need to be mutually exclusive. An OR-Join gateway synchronizes the execution of two or more parallel branches, as it waits for all *active* incoming branches to complete before triggering the outgoing flow. An OR gateway is drawn with a diamond marked with the symbol "○".
  - An *Event-Based gateway* is used after a decision to fork the flow into branches according to external choice. Its outgoing branches activation depends on taking place of catching events. Basically, such events are in a race condition, where the first event that is triggered wins and disables the other ones. An event-based gateway is drawn with a diamond marked with the symbol "⬠" double rounded.

  Notably, XOR and OR splitting gateways may have guard conditions in their outgoing sequence edges. The thesis considers both the case in which the decision on XOR-Split gateways is taken non-deterministically and the case where conditions are used to decide which edge to activate according to data values.

- **Artefacts** (Figure 2.7) are used to show additional information about a business process that "is not directly relevant for sequence flow or message flow of the process", as the standard mentions. Each artefact can be associated with flow elements. There are different types of artefacts. Among these, the thesis takes into account data objects and text annotations.

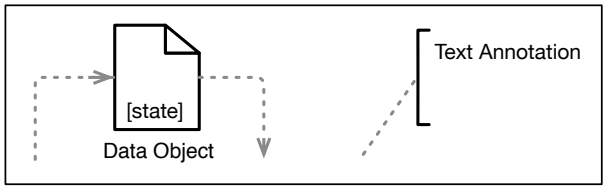


Figure 2.7: Considered BPMN Artefacts.

- *Data objects* (Figure 2.8) represent information and material flowing in and out of activities. They are depicted as a document with the upper-right corner folded over, and linked to activities with a dotted arrow with an open arrowhead (called *data association* in BPMN). The direction of the data association is used to establish whether a data object is an input or output for a given activity. Paper documents and electronic documents, as well as information on any type of medium can be represented by data objects. Sometimes data objects can refer to a state. Indicating data objects' states is optional: it can be done by appending the name of the state between square brackets to a data object's label. Different types of Data Objects are available, they are reported in the following.
  - (i) The *Data Object Collection* element refers to multi-instance Data Objects; using this a designer can express the involvement of more than one Data Object.
  - (ii) The *Data Input* element is used to express (external) input data, and it can be read by an activity.
  - (iii) The *Data Output* element is used to express output data, and it can be generated by an activity.
  - (iv) The *Data Store* element is used to express data that persist after the process instance finishes.
- *Text Annotations* are depicted as an open-ended rectangle encapsulating the text of the annotation, and are linked to a process modelling element via a dotted line (called association). Text annotations do not bear any semantics, thus they do not affect the flow of tokens through the process model.

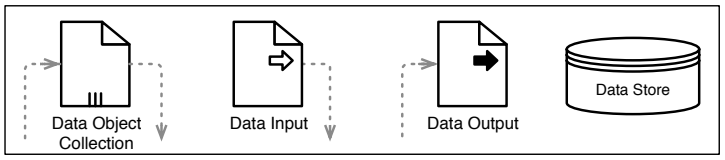


Figure 2.8: BPMN Types of Data Objects.

Finally, a key concept related to the BPMN process execution refers to the notion of *token*. The BPMN standard states that "a token is a theoretical concept that is used as an aid to define the behaviour of a process that is being performed" [68, 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

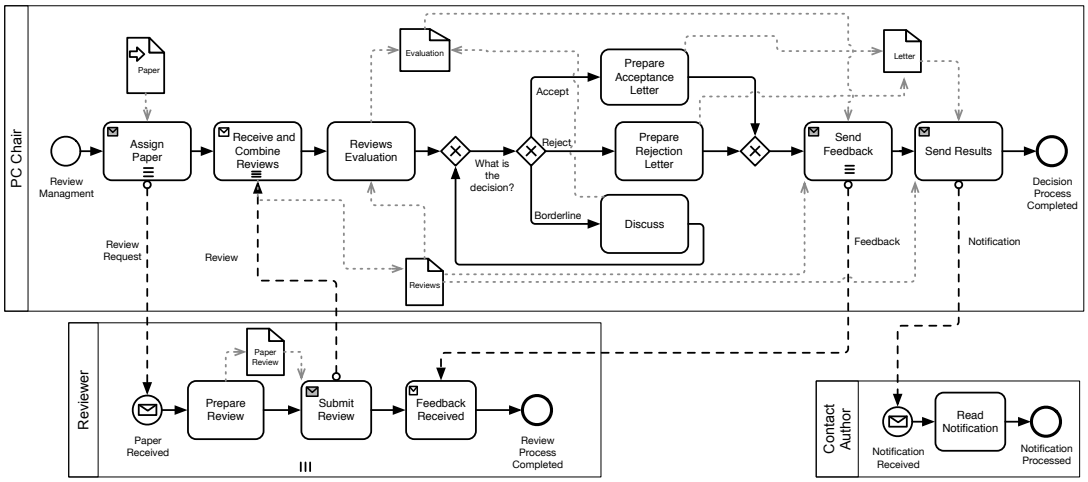


Figure 2.9: Paper Reviewing Collaboration Model.

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 flows able to generate messages. They will be referred as message flow tokens.

2.2.2 Running Example

The business process diagram in Figure 2.9 shows the reader how to use the BPMN notation and will be used through the thesis as a running example. Note, it will be appropriately adapted according to the chapter's focus.

It concerns the management of a single paper, which is revised by three reviewers; of course, the management of all papers submitted to the conference requires to enact the collaboration for each paper. It is modelled in BPMN as a collaboration whose participants are: **Program Committee (PC) Chair**, the organiser of the reviewing activities, whose behaviour is represented by the process within the PC Chair pool (for the sake of simplicity, it is assumed that the considered conference has only one chair); **Reviewer**, a person with knowledge in some of the conference topics who performs the reviewing activity and, since more than one reviewer takes part in this, he/she is modelled as a process instance of a multi-instance pool; **Contact Author**, who submitted a paper to the conference and acts on behalf of the other authors (contact author).

The reviewing process is started by the PC Chair. This is reflected by the start event of the process at the PC Chair pool. The activity enacted first is the assignment of the paper to each reviewer (via a multi-instance sequential activity with loop cardinality set to 3 according to the number of involved reviewers for each paper). The paper is passed to the PC Chair process by means of a data input. After all reviews are received, and combined in the

Reviews data object, the chair starts their evaluation. According to the value of the *Evaluation* data object, the chair prepares the acceptance/rejection letter (stored in the *Letter* data object) or, if the paper requires further discussion, the decision is postponed. The decision behaviour is rendered via a data-based XOR-Split gateway. Discussion interactions are here abstracted and always result in an accept or reject decision. The chair then sends back a feedback to each reviewer, attaches the reviews to the notification letter, and sends the result to the contact author, via send tasks.

2.3 Relevant Business Process Properties

Ensuring correctness is crucial in the business process modelling field, because an erroneous behaviour can cause high costs for the involved organisations. This is especially important when considering collaboration diagrams, where many parties need to properly and quickly interact at the base of the models.

To avoid undesired behaviours of BPMN models, modelling guidelines suggest to use structured building blocks [55]. This insight has triggered a stream of research on transforming unstructured models into structured ones. Informally, a process is well-structured if, for every element with multiple outgoing edges (a split), there is a corresponding node with multiple incoming edges (a join), such that the fragment of the model between the split and the join forms a single-entry-single-exit process component [44].

However, in some cases structuredness can only be achieved at the expense of increased model size [32], or it cannot be applicable at all [74, 73]. When possible, it would anyway limit BPMN expressiveness and designer freedom [72]. Thus, the thesis considers models with *arbitrary topology*; this is necessary to enable a classification of both structured and unstructured models.

Regarding the considered properties, the thesis relies on a well-known class of properties in the domain of business process management, namely *safeness* [100] and *soundness* [104, 99].

A BPMN process model is *safe*[3], if during its execution no more than one token occurs along the same sequence edge. The notion of safeness is inspired by the Petri Net formalisation where it means that the Petri Net does not contain more than one token at each place in all reachable markings [100].

A BPMN process model is *sound*, if once its execution starts, it is always possible to reach a marking where either (i) each marked end event is marked by at most one token and there are no other tokens around, or (ii) all edges are unmarked. Thus, soundness requires the successful completion of the process execution. Also soundness is inspired by the BPM literature that

[3]Notably, the notion of safeness is different from that of safety; safeness is a standard term in the BPM literature.

since the mid nineties presents several versions of soundness [100, 104, 99, 36]. The first soundness definition was provided by van der Aalst [96] in the context of workflow nets. However, this property can be applied to a wide range of process description languages either by transformation to workflow nets, if net-based formalisms are available [57], or by considering the corresponding execution semantics and checking the behaviour according to the model completion requirements.

There are other alternative notions of soundness described in the literature, that motivated some results of the thesis. These notions strengthen or weaken some of the requirements mentioned in the already presented soundness definition, called classical soundness [104].

- **k-soundness** [104] focuses on the "option to complete" property, which requires that any running process instance must eventually complete. It is parametrized with a variable k which indicates the initial number of tokens in the source place.

- **Weak soundness** [104] for business processes was developed in the context of process choreographies. It disallows deadlocks (i.e. a marking in which no transition is enabled), but it allows certain parts of the process not to participate in any process instance, that is it permits "dead activities".

- **Lazy soundness** [80] relaxes weak soundness, because it allows activities to be executed after the final state has been reached; however, deadlocks are not permitted before the final state has been reached. Furthermore, the final node will be executed exactly once, while other nodes representing activities can still be or become executed. However, they must not trigger the final node again.

- **Relaxed soundness** [104] states that all activities of a business process participate in it (dead activity freedom) and each transition is contained in at least one sound firing sequence of the system.

- **Easy soundness** [104] requires that a final state is reachable from some initial states.

There are a lot of different properties definitions 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 [54], and for Workflow Nets van der Aalst [104] provides two equivalent soundness definitions). To escape from the jungle of definitions and aiming to capture the BPMN expressibility the thesis provides a uniform formal framework for BPMN collaboration diagrams that enables to give a precise formal characterisation of relevant properties of the domain.

2.4 Operational Semantics

Providing a formal semantics of informal languages is an essential step to clearly define the requirements a system has to satisfy. In this regard, process algebras are mathematically rigorous languages with well defined semantics that permit describing and verifying properties of concurrent communicating systems [66].

The basic component of a process algebra is its syntax, that defines what well-formed models are. Specifically, it is the combination of operators and more elementary terms. Many different approaches (operational, denotational, algebraic) can be used for describing the meaning of processes, that is to provide a semantics of the considered operators. However, the operational approach has become the reference one. By relying on the so called Structural Operational Semantics (SOS), an operational semantics models a process as a labelled transition system (LTS), that consists of a set of states, a set of transition labels and a transition relation. The states of the transition system are just process algebra terms while the labels of the transitions between states represent the actions or the interactions that are possible from a given state and the state that is reached after the action is performed by means of visible and invisible actions. Formally, an LTS is defined as follows.

Definition 1 (Labelled Transition System)**.** *A labelled state transition system is a tuple* (S, Σ, δ) *such that*

- S *is a finite set of states,*
- Σ *is a finite alphabet (i.e. a finite, non-empty set of symbols which refer to actions that a system can perform),*
- $\delta \subset S \times \Sigma \times S$ *is a state transition relation.*

Inference systems are used to associate LTSs to process terms and are defined as follows.

- *Inference Systems* are a set of inference rule of the form

$$\frac{p_1, \ldots, p_n}{q}$$

  where $p_1, \ldots, p_n$ are the premises and q is the conclusion. Each rule is interpreted as an implication: if all premises are satisfied then also the conclusion is inferred.

- *Axiom* is a rule without premises and it is written as

$$\frac{}{q} \quad \text{or} \quad q$$

- *Transition Rules* represent transitions between states. In the case of operational semantics the premises and the conclusions will be triples of the form $P \xrightarrow{\ell} Q$ and thus the rules for each operator *op* of the process algebras will be alike the one below, where $\{i_1, \ldots, i_m\} \subseteq \{1, \ldots, n\}$ and $E'_i = E_i$ when $i \notin \{i_1, \ldots, i_m\}$.

$$\frac{E_{i1} \xrightarrow{\ell_1} E'_{i1} \ldots E_{im} \xrightarrow{\ell_m} E'_{im}}{op(E_1, \ldots, E_n) \xrightarrow{\ell} C[E'_1, \ldots, E'_n]}$$

  In the rule the target term $C[\,]$ indicates the new context in which the new sub terms will be operating after the reduction. Sometimes, these rules are enriched with *side conditions* that determine their applicability. Therefore, transition rules, given a term representing a state of the system, permit to determine the enabled actions and the corresponding reachable states, thus defining an LTS.

- *Basic Actions* represent the atomic (non-interruptible) step of a computation that is performed by a system to move from one state to the next. Actions represent various activities of concurrent systems, like sending or receiving a message, synchronizing with other processes etc. In process algebras two main types of atomic actions are considered, namely *visible* (or external) actions and *invisible* (or internal actions), the latter referred by the Greek letter τ.

Part II

BPMN Formalisation and Correctness Properties

Chapter 3

Core BPMN Operational Semantics

Providing a solid foundation to enable BPMN designers to understand their models in a consistent way is becoming more and more important. It is with the objective of reaching this goal that this thesis defines and exploits a formal characterisation of the collaborations' semantics, specifically and directly given for BPMN models, to successively provide their classification. More specifically, it is provided an operational semantics to BPMN in the SOS style [71] by relying on the notion of LTS.

Notably, as shown in [64], even if the BPMN specification is quite wide, only less than 20% of its vocabulary is used regularly in designing business process models. Indeed, in this part of the thesis, only a core fragment of BPMN elements will be considered. However, it will be also shown how the proposed framework can be extended including other constructs (Part III).

In detail, this chapter reports the work regarding the definition of an Operational Semantics for the core BPMN notation. A running example is first presented (Section 3.1) to illustrate both the considered elements and the formal framework (Section 3.2).

Highlights. Distinctive aspects of the proposed semantics are:

1. it is a direct semantics, given in terms of features and constructs of BPMN, instead of an encoding into other formalisms (e.g. Petri Nets) as in most proposals in the literature;

2. it is specially given for collaboration diagrams, thus it focuses on elements used for describing communications between different processes (e.g. message events, tasks and pools) that are usually not considered by other formalisations;

3. it is suitable to model both process and collaboration diagrams;

4. it refers to models involving processes with arbitrary topology, thus overcoming the well-structuredness limitations.

3.1 Running Example

To introduce the core elements set, the business process model shown in Figure 3.1 is discussed. This collaboration diagram represents a modified version of the reviewing process for a scientific conference, which has been already introduced in Section 2.2.2. Since the goal of this example is to introduce the core elements, simplifications are in place. Specifically, the reviewing process usually involves many reviewers and many authors. For convenience, just one author and one reviewer are considered.

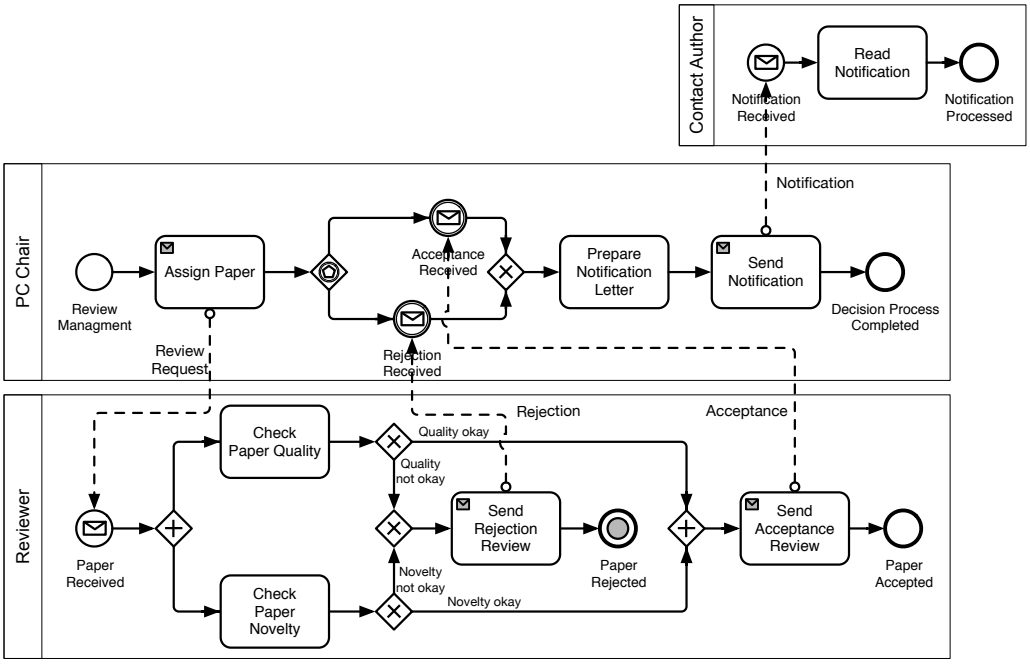


Figure 3.1: Paper Reviewing Collaboration Model.

The program committee chairperson is the central role in the collaboration; this role is represented by the **PC Chair** pool. There are two additional roles involved, namely the **Contact Author** who submitted the research paper (the submission is omitted here) and waits for a notification and the **Reviewer** who writes a rejection or acceptance review for the submitted paper, according to some requirements. Considering the Reviewer pool, from left to right we have that in order to decide the score of the paper, novelty and quality of the paper are checked in parallel, via an AND-Split gateway.

If the results of both checks are positive, the Reviewer sends an acceptance review to the PC Chair. If one of the check has a negative result (e.g. the paper lacks of novelty), the result of the other check is irrelevant, because the paper will be rejected anyway. Thus, as soon as one check provides a negative results, the Reviewer sends a rejection review to the PC Chair, completing its activities by means of a terminate event which stops and aborts the running process, including the other check. Considering the PC Chair pool, after assigning the paper to the Reviewer, the PC Chair waits either for an acceptance review or for a rejection review. This behaviour is rendered by means of an event based gateway. Then, the PC Chair prepares the notification letter and sends it to the Contact Author, who can read and process it.

3.2 Formal Framework

This section presents the BPMN formalisation. The proposed direct semantics is inspired by [21], but its technical definition is significantly different and its presentation is more compact and, hence, suitable for a formal study. In particular, configuration states are here defined according to a global perspective. Specifically, the section first presents the syntax and operational semantics which have been defined for a relevant subset of BPMN elements.

3.2.1 Syntax

To enable the formal treatment of collaborations' semantics, a BNF syntax of their model structure is defined (Figure 3.2). 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 and gateways.

It is worth noticing that the syntax is too permissive with respect to the BPMN notation, as it allows to write collaborations that cannot be expressed

$$
\begin{array}{lcl}
C & ::= & \mathsf{pool}(\mathsf{p}, P) \quad | \quad C \,\|\, C \\
P & ::= & \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \mid \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}) \mid \mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp}) \\
 & | & \mathsf{terminate}(\mathsf{e}_i) \mid \mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_h, \mathsf{e}_{oh})) \\
 & | & \mathsf{andSplit}(\mathsf{e}_i, E_o) \mid \mathsf{xorSplit}(\mathsf{e}_i, E_o) \mid \mathsf{andJoin}(E_i, \mathsf{e}_o) \mid \mathsf{xorJoin}(E_i, \mathsf{e}_o) \\
 & | & \mathsf{task}(\mathsf{e}_i, \mathsf{e}_o) \mid \mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{empty}(\mathsf{e}_i, \mathsf{e}_o) \\
 & | & \mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid P_1 \,\|\, P_2
\end{array}
$$

Figure 3.2: Syntax of BPMN Collaboration Structures.

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 this work, as it is not proposing an alternative modelling notation, but it is only using a textual representation of BPMN models, which is more manageable for writing operational rules than the graphical notation. Therefore, the thesis analysis only considers terms of the syntax that are derived from BPMN models.

Intuitively, a BPMN collaboration model is rendered in this 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.

Notably, when considering the BPMN core set of elements it is not possible to distinguish the difference between communicative tasks and intermediate events (see Figure 3.3). This difference would be clear when extending the formal framework with data.

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. Besides, $\mathsf{e} \in \mathbb{E}$ denotes a *sequence edge*, while $E \in 2^{\mathbb{E}}$ a set of edges; it is required $|E| > 1$ when it is used in joining and splitting gateways. Similarly, an event-based gateway is required to contain at least two message events, i.e. $h > 1$ in each $\mathsf{eventBased}$ term. For the convenience of the reader, e_i refers to the edge incoming in an element and e_o to the edge outgoing from an element. In the edge set $\mathbb{E}$, also spurious edges are included, denoting the enabled status of start events and the completed status of end events, named *enabling* and *completing* edges, respectively. In particular, edge e_{enb}, incoming to a start event, is used 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. It is assumed that a unique sequence (resp. message) edge name is associated to each sequence (resp. message) flow in the BPMN model.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- $C_1 \| C_2$ represents a composition of elements in order to render a collaboration structure in terms of a collection of elements.

To achieve a compositional definition, each sequence (resp. message) edge of the BPMN model is split in two parts: the part outgoing from the source element and the part incoming into the target element. The two parts are correlated since edge names in the BPMN model are unique.

Here in the following some auxiliary functions are 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.

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

To refer to the enabling edges of the start events of the considered process function $start(P)$ is used.

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

Notably, it is assumed that each 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.

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

Similarly functions $end(P)$ and $end(C)$ are defined on the structure of processes and collaborations in order to refer to end events in the considered process.

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

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

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

Function $edges(P)$ refers the edges in the scope of P and $edgesEl(P)$ indicates the edges in the scope of P without considering the spurious edges (the formal definitions can be found in Appendix B).

Running Example 1. *The BPMN model in Figure 3.1 is expressed in the syntax as the following collaboration structure (at an unspecified step of execution):*

$$\mathsf{pool}(\mathsf{PC\ Chair}, P_{pc}) \parallel \mathsf{pool}(\mathsf{Reviewer}, P_r) \parallel \mathsf{pool}(\mathsf{Contact\ Author}, P_{ca})$$

with P_{pc}, P_r and P_{ca} are expressed as follows, where for simplicity the edges are identified in progressive order e_i (with $i = 0 \ldots 26$):

$$
\begin{aligned}
P_{pc} \;=\;& \mathsf{start}(\mathsf{e}_0, \mathsf{e}_1) \parallel \mathsf{taskSnd}(\mathsf{e}_1, \text{Review Request}, \mathsf{e}_2) \parallel \\
& \mathsf{eventBased}(\mathsf{e}_2, (\text{Acceptance Received}, \mathsf{e}_3), (\text{Rejection Received}, \mathsf{e}_4)) \parallel \\
& \mathsf{xorJoin}(\{\mathsf{e}_3, \mathsf{e}_4\}, \mathsf{e}_5) \parallel \mathsf{task}(\mathsf{e}_5, \mathsf{e}_6) \parallel \mathsf{taskSnd}(\mathsf{e}_6, \text{Notification}, \mathsf{e}_7) \parallel \\
& \mathsf{end}(\mathsf{e}_7, \mathsf{e}_8) \\
P_r \;=\;& \mathsf{startRcv}(\mathsf{e}_8, \text{ReviewRequest}, \mathsf{e}_9) \parallel \mathsf{andSplit}(\mathsf{e}_9, \{\mathsf{e}_{10}, \mathsf{e}_{11}\}) \parallel \\
& \mathsf{task}(\mathsf{e}_{10}, \mathsf{e}_{12}) \parallel \mathsf{task}(\mathsf{e}_{11}, \mathsf{e}_{13}) \parallel \mathsf{task}(\mathsf{e}_5, \mathsf{e}_6) \parallel \mathsf{xorSplit}(\mathsf{e}_{12}, \{\mathsf{e}_{15}, \mathsf{e}_{16}\}) \parallel \\
& \mathsf{xorSplit}(\mathsf{e}_{13}, \{\mathsf{e}_{14}, \mathsf{e}_{17}\}) \parallel \mathsf{xorJoin}(\{\mathsf{e}_{14}, \mathsf{e}_{15}\}, \mathsf{e}_{18}) \parallel \\
& \mathsf{taskSnd}(\mathsf{e}_{18}, \text{Rejection}, \mathsf{e}_{19}) \parallel \mathsf{terminate}(\mathsf{e}_{19}) \parallel \mathsf{andJoin}(\{\mathsf{e}_{16}, \mathsf{e}_{17}\}, \mathsf{e}_{20}) \parallel \\
& \mathsf{taskSnd}(\mathsf{e}_{20}, \text{Acceptance}, \mathsf{e}_{21}) \parallel \mathsf{end}(\mathsf{e}_{21}, \mathsf{e}_{22}) \\
P_{ca} \;=\;& \mathsf{startRcv}(\mathsf{e}_{23}, \text{Notification}, \mathsf{e}_{24}) \parallel \mathsf{andSplit}(\mathsf{e}_9, \{\mathsf{e}_{10}, \mathsf{e}_{11}\}) \parallel \\
& \mathsf{task}(\mathsf{e}_{24}, \mathsf{e}_{25}) \parallel \mathsf{end}(\mathsf{e}_{25}, \mathsf{e}_{26})
\end{aligned}
$$

Moreover, considering functions defined on the structure it follows: $participant(\mathsf{pool}(\text{PC Chair}, P_{pc}) \parallel \mathsf{pool}(\text{Reviewer}, P_r) \parallel \mathsf{pool}(\text{Contact Author}, P_{ca})) = \{P_{pc}, P_r, P_{ca}\}$, $start(P_{pc}) = \{\mathsf{e}_0\}$, and $end(P_{pc}) = \{\mathsf{e}_8\}$. Finally, $edges(P_{pc}) = \{\mathsf{e}_0, ..., \mathsf{e}_8\}$, $edgesEl(P_{pc}) = \{\mathsf{e}_1, ..., \mathsf{e}_7\}$. The others are defined in a similar way. □

Notice, the one-to-one correspondence between the syntax used here to represent a BPMN model and the graphical notation of BPMN, that is exemplified by means of the running example in Figure 3.1, is also reported in detail in Appendix A.1.

3.2.2 Semantics

The syntax presented so far permits to describe the mere structure of a collaboration and a process. To describe their semantics it is necessary to enrich it with a notion of execution state, defining the current marking of sequence and message edges. *Collaboration configuration* and *process configuration* 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 $\sigma : \mathbb{E} \to \mathbb{N}$ is a function mapping edges to a number of tokens. The state obtained by

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

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

Definition 2 (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 3 (Initial state of collaboration)**.** *Let $\langle C, \sigma, \delta \rangle$ be a collaboration configuration, then, the 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 2. *The initial configuration of the collaboration in Figure 3.1 is as follows. Given $participant(C) = \{P_{pc}, P_r, P_{ca}\}$, it results that $\langle P_{pc}, \sigma \rangle$, $\sigma(\mathsf{e}_0) = 1$ and $\sigma(\mathsf{e}_i) = 0\ \ \forall \mathsf{e}_i$ with $i = 1 \ldots 8$, $\langle P_r, \sigma \rangle$, $\sigma(\mathsf{e}_j) = 0\ \ \forall \mathsf{e}_j$ with $j = 9 \ldots 23$ and $\langle P_{ca}, \sigma \rangle$, $\sigma(\mathsf{e}_k) = 0\ \ \forall \mathsf{e}_k$ with $j = 24 \ldots 28$. Moreover, $\delta(\mathsf{Review\ Request}, \mathsf{Rejection}, \mathsf{Acceptance}, \mathsf{Notification}) = 0$.* □

The operational semantics is defined by means of an LTS on collaboration configurations 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 processes.

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 ℓ, 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*. It will be written $\langle P, \sigma \rangle \xrightarrow{\ell} \langle P, \sigma' \rangle$ to indicate that $(\langle P, \sigma \rangle, \ell, \langle P, \sigma' \rangle) \in \rightarrow$ and say that process configuration $\langle P, \sigma \rangle$ performs a transition labelled by ℓ 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 the structure is omitted from the target configuration of the transition. Thus, a transition $\langle P, \sigma \rangle \xrightarrow{\ell} \langle P, \sigma' \rangle$ is written as $\langle P, \sigma \rangle \xrightarrow{\ell} \sigma'$. The labels used by this transition relation are generated by the following production rules.

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

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 Figure 3.3. Before commenting on the rules, the auxiliary functions they exploit need to be introduced. Specifically, function $inc : \mathbb{S} \times \mathbb{E} \to \mathbb{S}$ (resp. $dec : \mathbb{S} \times \mathbb{E} \to \mathbb{S}$), 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: $inc(\sigma, \varnothing) = \sigma$ 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. Function $reset : \mathbb{S} \times \mathbb{E} \to \mathbb{S}$ allows updating a state by setting to zero the number of tokens marking an edge in the state. Formally, it is defined as follows: $reset(\sigma, \mathsf{e}) = \sigma \cdot [\mathsf{e} \mapsto 0]$. Also in this case the function extends in a natural ways to sets of edges as follows: $reset(\sigma, \varnothing) = \sigma$ and $reset(\sigma, \{\mathsf{e}\} \cup E)) = reset(reset(\sigma, \mathsf{e}), E)$.

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

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

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

Now the operational rules in Figure 3.3 are described. Rule *P-Start* starts the execution of a 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-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-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-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-Terminate* starts when there is at least one token in the incoming edge of the terminate event, which is then removed. Rule *P-EventG* is activated when there is a token in the incoming edge and there is a message m_j to be consumed, so that the application of the rule moves the token from the incoming edge to the outgoing edge corresponding

to the received message. A label corresponding to the consumption of a message is observed.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{\langle P_1,\sigma\rangle \xrightarrow{kill} \sigma'}{\langle P_1 \parallel P_2,\sigma\rangle \xrightarrow{kill} reset(\sigma', edges(P_1 \parallel P_2))}\ (P\text{-}Kill_1)$$

$$\frac{\langle P_2,\sigma\rangle \xrightarrow{kill} \sigma'}{\langle P_1 \parallel P_2,\sigma\rangle \xrightarrow{kill} reset(\sigma', edges(P_1 \parallel P_2))}\ (P\text{-}Kill_2)$$

$$\frac{\langle P_1,\sigma\rangle \xrightarrow{\ell} \sigma' \quad \ell\neq kill}{\langle P_1 \parallel P_2,\sigma\rangle \xrightarrow{\ell} \sigma'}\ (P\text{-}Int_1) \qquad \frac{\langle P_2,\sigma\rangle \xrightarrow{\ell} \sigma' \quad \ell\neq kill}{\langle P_1 \parallel P_2,\sigma\rangle \xrightarrow{\ell} \sigma'}\ (P\text{-}Int_2)$$

Figure 3.3: BPMN Semantics - Process Level.

Rule *P-AndSplit* is applied when there is at least one token in the in-

coming 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 of the AND-Join gateway 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 of the XOR-Join gateway, 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, sends 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. Rule $P\text{-}Kill_1$ and $P\text{-}Kill_2$ deal with the propagation of killing action in the scope of P and rule $P\text{-}Int_1$, $P\text{-}Int_2$ deal with interleaving in a standard way for process elements.

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 ℓ, 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*. It will be written $\langle C, \sigma, \delta \rangle \xrightarrow{l} \langle C, \sigma', \delta' \rangle$ to indicate that $(\langle C, \sigma, \delta \rangle, l, \langle C, \sigma', \delta' \rangle) \in \rightarrow$ and say that collaboration configuration $\langle C, \sigma, \delta \rangle$ performs transition labelled by l 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 the structure is omitted from the target configuration of the transition. Thus, a transition $\langle C, \sigma, \delta \rangle \xrightarrow{l} \langle C, \sigma', \delta' \rangle$ is written as $\langle C, \sigma, \delta \rangle \xrightarrow{l} \langle \sigma', \delta' \rangle$.

Recall, ℓ are the following: label τ denotes an action internal to the process, while $!\mathsf{m}$ and $?\mathsf{m}$ denote sending and receiving actions, respectively. The rules related to the collaboration are defined in Figure 3.4.

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

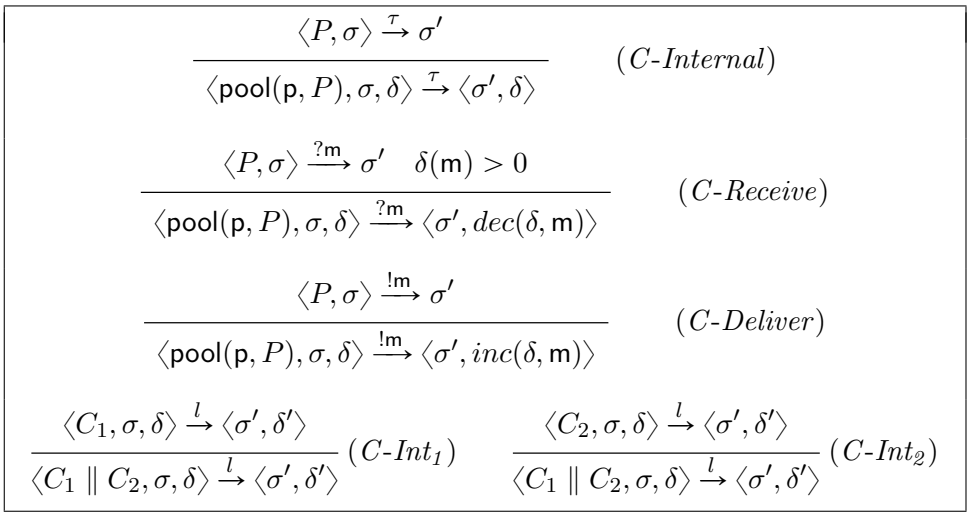


Figure 3.4: BPMN Semantics - Collaboration Level.

at collaboration layer. Notably, rule *C-Receive* can be applied only if there is at least one message available (premise $\delta(\mathsf{m}) > 0$); of course, one token is consumed by this transition. Recall indeed that at process label ?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 !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*$_1$, *C-Int*$_2$ permit 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.

3.3 Comparison with other Approaches

Much effort has been devoted to the formalisation of the BPMN standard. In the following the most relevant attempts are referred. First the other direct formalisation approaches available in the literature are considered, then some mappings from BPMN to well-known formalisms are discussed.

3.3.1 BPMN Direct Formalisations

With regard to direct formalisations, relevant work are those by Van Gorp and Dijkman [105], Christiansen et al. [18], El-Saber and Boronat [35], and Borger and Thalheim [14]. Among them, the contribution of this thesis is mainly inspired by the one presented in [105]. They propose a BPMN 2.0 formalisation based on in-place graph transformation rules; these rules are

defined to be passed as input to the GrGen.NET tool, and are documented visually using BPMN syntax. With respect to this thesis, the used formalisation techniques are different, since here an operational semantics in terms of LTS is provided. This allows to apply verification techniques based on transition labels, as e.g. model checking of properties expressed as formulae of action-based temporal logic.

Another interesting work is described in [18], where Christiansen et al. propose a direct formalisation of the BPMN 2.0 Beta 1 Specification using algorithms based on incrementally updated data structures. The semantics is given for BPMNinc, that is a minimal subset of BPMN 2.0 containing just inclusive and exclusive gateways, start and end events, and sequence flows. This work differs from the one presented here with respect to the formalisation method, as it proposes a token-based semantics à la Petri Nets, while here an operational semantics with a compositional approach à la process calculi is defined. Moreover, the work in [18] also lacks to take into account BPMN organisational aspects and the flow of messages, whose treatment is a main contribution of this thesis.

El-Saber and Boronat proposed in [35] a formal characterisation of well-formed BPMN processes in terms of rewriting logic, using Maude as supporting tool. This formalisation refers to a subset of the BPMN specification considering elements that are used regularly, such as flow nodes, data elements, connecting flow elements, artefacts, and swimlanes. Interesting it is also the mechanism given to represent and evaluate guard conditions in decision gateways. Differently from the other direct formalisations, this approach can be only applied to well-structured processes. Besides this limitation, there is another drawback concerning BPMN organisational aspects and messages flow. In particular, even if it is stated that messages are included in the formalisation, their formal treatment is not explained in the paper. Borger and Thalheim [14] define an extensible semantical framework for business process modelling notations using Abstract State Machines as method. However, the proposed formalisation is based on the version 1.0 of BPMN, which does not include notation meta-model and gives more freedom to the authors in the interpretation of the language.

3.3.2 BPMN Formalisations via Mapping

The most common formalisations of BPMN 2.0 are given via mappings to various formalisms, such as Petri Nets [42, 46, 83, 8, 29], YAWL [115, 24] and process calculi [113, 112, 7, 75, 80, 78]. However, models resulting from a mapping inherit constraints given by the target formal language and so far none of them considers BPMN features such as different abstraction levels (i.e., collaboration, process), the asynchronous communication model, and the notion of completion due to different types of end event (i.e., simple,

message throwing, and terminate).

Regarding the mapping from BPMN to Petri Nets, the most relevant attempt is the one provided by Dijkman et al. [29]. Differently from the approach proposed in this thesis, even if the mapping deals with messages, they rendered a message (token) as a (standard) token in a place. Moreover, it does not properly consider multiple organisation scenarios, and does not provide information to the analysis phase regarding who are the participants involved in the exchange of messages. Finally, while for the basic BPMN modelling elements the encoding in Petri Nets is rather straightforward, for other elements, such as the terminating events, such a mapping is quite difficult. This is due to the inherent complexity of managing non-local propagation of tokens in Petri Nets, which instead is supported by the proposed semantics.

Other relevant mappings are those from BPMN to YAWL, a language with a strictly defined execution semantics inspired by Petri Nets. In this regard, worth mentioning are the ones by Ye and Song [115] and Dumas et. al. [24]. The former is defined under the well-formedness assumption, which instead this thesis does not rely on. Moreover, although messages are taken into account in the mapping, pools and lanes are not considered; thus it is not possible to identify who is the sender and who is the receiver in the communication. This results in the lack of capability to introduce verification at message level considering the involved organisations. The latter mapping, instead, formalises a very small portion of BPMN elements. In particular, limitations about pools and messages are similar to the previous approach: pools are treated as separate business processes, while messages flow is not covered by the mapping.

Process calculi has been also considered as means for formalising BPMN. Among the others, Wong and Gibbons presented in [112] a translation from a subset of BPMN process diagrams, under the assumption of well-formedness, to a CSP-like language based on Z notation. Even if messages have been omitted in the formalisation presented in [112], their treatment is discussed in [113]. Messages are also considered by Arbab et al. in [7], where the main BPMN modelling primitives are represented by means of the coordination language Reo. Differently from the other mappings, this one considers a significantly larger set of BPMN elements. Prandi et al., instead, defined in [75] a semantics in term of a mapping from BPMN to the process calculus COWS, which has been specifically devised for modelling service-oriented systems. Last but not least, also π-calculus was taken as target language of mapping by Hutchison et al. [80] and Puhlmann [78]. Even if the presented proposal differs from the above ones, as it is a direct semantics rather than a mapping, it has drawn inspiration from those based on process calculi for the use of a compositional approach in the SOS style.

Chapter 4

BPMN Correctness Properties

The formalisation of a relevant subset of BPMN elements presented in the previous chapter allows to define a classification of BPMN collaboration models according to important properties of the business process domain. It is worth noticing that this work aims at providing a classification specific for BPMN models. This is made possible by a formal semantics directly defined on BPMN elements. Regarding the considered properties, the classification relies on a well-known class of properties in the domain of business process management, namely *safeness* [100] and *soundness* [104, 99]. So far, despite the large body of work on this topic, no formal definition of such properties directly given on BPMN is provided. The thesis reconciles in a single framework properties taken into account by different languages, like Petri Nets [65], Workflow Nets [100], and Elementary Nets [87]. Studying different properties in the same framework does not leave any room for ambiguity, and increases the potential for formal reasoning on their relationship.

Thus, this chapter first provides the definition of the studied properties (Section 4.1) and shows their practical relevance (Section 4.2). Then, it reports the classification of BPMN models (Section 4.3) and how it has been obtained (Section 4.4). Finally it presents the study on the compositionality of safeness and soundness (Section 4.6).

Highlights. The main benefits of the proposed approach are:

1. rather than converting the BPMN model to a Petri or Workflow Net and studying relevant properties, and their relationships, on the model resulting from the mapping such properties are directly reported on BPMN considering its complexity;

2. being close to the BPMN standard, the resulting classification allows to catch the language peculiarities, such as the asynchronous communication models, message passing and the completeness notion distin-

guishing the effect of end event and the terminate event, showing their impact on soundness;

3. the classification provides a novel contribution by extending the reasoning from BPMN processes to BPMN collaborations.

4.1 Properties of BPMN Collaborations

This section provides 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. These properties are characterised both at the process and collaboration levels.

4.1.1 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 *well-structuredness*. In this thesis the notion of well-structurednes is inspired by the definition of well-structuredness given by Kiepuszewski et al. [44]. Such a definition was given on workflow models and it is not expressive enough for BPMN, so it is here extended to well-structured collaborations including all the elements defined in the proposed semantics (e.g., event based gateway).

Before providing a formal characterisation of well-structured BPMN processes and collaborations, some auxiliary definitions need to be introduced. In particular, functions $in(P)$ and $out(P)$ are inductively defined. They determine the incoming and outgoing sequence edges of a process element P (the formal definition is relegated to Appendix B).

Moreover, to simplify the definition of well-structuredness for processes, the definition of well-structured core is also provided, 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:*

$$\mathsf{start}(\mathsf{e},\mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'',\mathsf{e}''') \tag{4.1}$$

$$\mathsf{start}(\mathsf{e},\mathsf{e}') \parallel P' \parallel \mathsf{terminate}(\mathsf{e}'') \tag{4.2}$$

$$\mathsf{start}(\mathsf{e},\mathsf{e}') \parallel P' \parallel \mathsf{endSnd}(\mathsf{e}'',\mathsf{m},\mathsf{e}''') \tag{4.3}$$

$$\mathsf{startRcv}(\mathsf{e},\mathsf{m},\mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'',\mathsf{e}''') \tag{4.4}$$

$$\mathsf{startRcv}(\mathsf{e},\mathsf{m},\mathsf{e}') \parallel P' \parallel \mathsf{terminate}(\mathsf{e}'') \tag{4.5}$$

$$\mathsf{startRcv}(\mathsf{e},\mathsf{m},\mathsf{e}') \parallel P' \parallel \mathsf{endSnd}(\mathsf{e}'',\mathsf{m},\mathsf{e}''') \tag{4.6}$$

where $in(P') = \{\mathsf{e}'\}$, $out(P') = \{\mathsf{e}''\}$, and $isWSCore(P')$.

Predicate $isWSCore(\cdot)$ *is inductively defined on the structure of its 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. $isWSCore(\mathsf{interSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}'))$

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

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

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

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

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

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. The following events are allowed: a start event or a start message event and one simple end event or terminate event or end message 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 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 Figure 4.1. A process

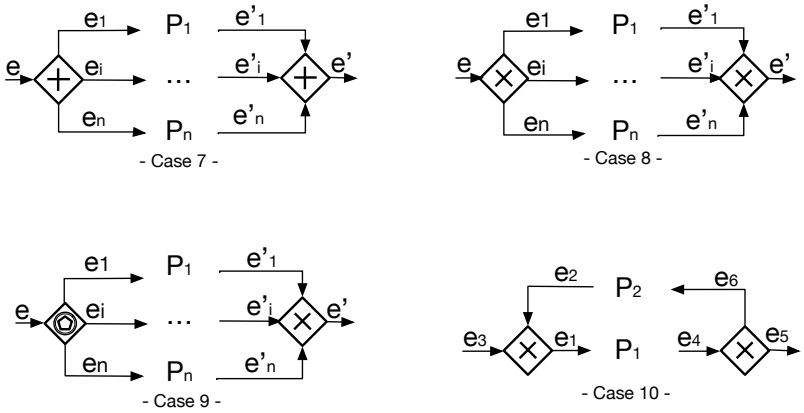


Figure 4.1: Well-structured Core Elements (cases 7-10).

element collection is well-structured core if its processes are well-structured and sequentially composed.

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 $isWS(P)$;*
- *$isWS(C_1 \parallel C_2)$ if $isWS(C_1)$ and $isWS(C_2)$.*

Running Example 3. *Considering the running example proposed in the previous chapter, and according to the above definitions, processes P_{pc} and P_{ca} are well-structured, while process P_r is not well-structured, due to the presence of the unstructured process fragment formed by two XOR-Split and only one XOR-Join. Thus, the overall collaboration is not well-structured.* □

4.1.2 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, the following definition determining the safeness of a process in a given state needs to be introduced.

Definition 6 (Current state safe process)**.** *A process configuration* $\langle P, \sigma \rangle$ *is current state safe (*cs-safe*) if and only if* $\forall \mathsf{e} \in edgesEl(P) \,.\, \sigma(\mathsf{e}) \leqslant 1$.

Now the definition of safe processes and collaborations can be given. It 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. In the following $\rightarrow^*$ denotes 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 4. *Considering again the running example depicted in Figure 3.1, processes* P_{pc} *and* P_{ca} *are safe since there are not process fragments capable of producing more than one token. Process* P_r *instead is not safe. In fact, if the paper lacks both quality and novelty two tokens will arrive at the XOR-Join gateway. This will produce more than one token in the sequence flow connected to the Send Rejection Review task. Thus, also the resulting collaboration is not safe.* □

4.1.3 Sound BPMN Collaborations

Another relevant property from the business process modelling domain is soundness. It is defined here 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 must be processed during the execution (i.e. no pending message tokens are observed).

Before providing a formal characterisation of sound BPMN processes and collaborations, the following definition determining the soundness of a process in a given state needs to be introduced.

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 holds:*

(i) $\forall\, \mathsf{e} \in marked(\sigma,end(P))\ .\ \sigma(\mathsf{e}) = 1$ *and* $\forall \mathsf{e} \in edges(P)\backslash end(P)\ .\ \sigma(\mathsf{e}) = 0$.

(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 capability of the proposed formalisation to distinguish sequence tokens from message tokens the soundness property can be relaxed by defining message-relaxed soundness. It extends the usual soundness notion by considering sound also those collaborations in which asynchronously sent messages are not handled by the receiver.

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

Running Example 5. *Considering again the running example of the previous chapter, it is easy to see that processes P_{pc} and P_{ca} are sound, since, in each of them, it is always possible to reach the end event and when reached there is no token marking the sequence flows. Also process P_r 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, if both the results of the checks are negative, task Send Rejection Review would be executed twice but only one Rejection message can be handled by the PC Chair. Thus, even if the processes complete, the message lists are not empty. However, the collaboration satisfies the message-relaxed soundness property.* □

4.2 Relevance into Practice

To get a clearer idea of the impact of well-structuredness, safeness, and soundness on the real-world modelling practice, this section reports an anal-

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

Table 4.1: Classification of the Models in the BPM Academic Initiative Repository.

ysis of the BPMN 2.0 collaboration models available in a well-known, public, well-populated repository provided by the BPM Academic Initiative (http://bpmai.org). The models have been created by students from various universities as part of their process modelling education. As such, they are representative of real modelling practice and show a high heterogeneity along various dimensions; indeed empirical insights that are grounded on them can be assumed to have a high external validity [49]. Thus, the repository is not representative of good modelling and is not a collection of validated models, but it is particularly suited to investigate real modelling practice, modelling styles and the relevance of modelling constructs.

From the raw dataset, to avoid uncompleted models and low quality ones, only those with 100% of connectedness (i.e., all model elements are connected) have been selected. This results on 2.740 models suitable for the investigation. To better understand the trend, in Table 4.1 the models are grouped in terms of number of contained elements. From the technical point of view, well-structuredness has been checked using the PromniCAT platform[1], while safeness and soundness have been checked using the S^3 tool[2]. These tools have been chosen because they faithfully implement the provided definitions: PromniCAT is able to verify the well-structuredness definition, which is a standard one, while S^3 has been developed implementing the overall formal framework, thus also the safeness and soundness notions.

It results that 86% of models in the repository are well-structured. Anyway, more interesting is the trend of the number of well-structured models with respect to their size. It shows that in practice BPMN models starts to become unstructured when their size grows. This means that, even if structuredness is good property, it should be regarded as a general guideline;

[1] https://github.com/tobiashoppe/promnicat
[2] http://pros.unicam.it/s3/

one can deviate from it if necessary, especially in modelling complex scenarios. The balancing between the two classes motivates, on the one hand, the design choice of considering in the formalisation BPMN models with an arbitrary topology and, on the other hand, the necessity of studying well-structuredness and the related properties.

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

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

4.3 Classification Results

The formal definition of well-structuredness, safeness and (message-relaxed) soundness directly on BPMN models allows to provide a precise classification of those models according to the property they satisfy.

This section shows how the classification advances the state of the art with respect to other available classifications in the literature. It also discusses how the formal framework enables a more precise classification of the BPMN models considering language peculiarities.

4.3.1 Advances with respect to already available classifications

Differently from other classifications reasoning at the process level by means of Workflow Nets [27, 96] and π-calculus [81], the proposed study directly addresses BPMN collaboration models. By relying on a uniform formal framework, novel results are achieved. They are synthesized in the Euler diagram in Figure 4.2. In particular the diagram 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.

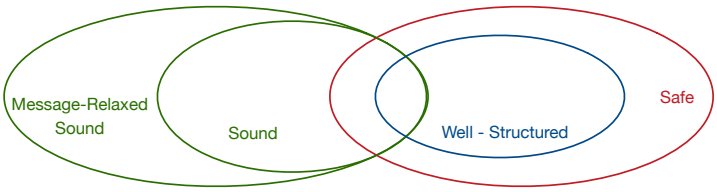


Figure 4.2: Classification of BPMN Collaborations.

Result *(i)* demonstrates that well-structured collaborations represent a subclass of safe collaborations. It will be also shown that such relation is valid at the process level, where the classification relaxes the existing results on Workflow Nets, stating that a process model to be safe has to be not only well-structured, but also sound [96].

Result *(ii)* shows that there are well-structured collaborations that are not sound. Well-structuredness instead implies soundness at the process level, confirming results provided on Workflow Nets, [106], but relaxing the one obtained in Petri Nets [27], where relaxed soundness and well-structuredness together imply soundness.

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

Result *(iii)* shows that there are sound and message-relaxed sound collaborations that are not safe. This can also be observed at process level resulting in a novel contribution strictly related to the expressiveness of BPMN and its differences with respect to other workflow languages. In fact, Van der Aalst shows that soundness of a Workflow Net is equivalent to liveness and boundedness of the corresponding short-circuited Petri Net [97]. 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 [38, 76]. Thus, a sound workflow is always safe. In BPMN instead there are unsafe processes that are sound.

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

4.3.2 Advance in Classifying BPMN Models

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

Considering collaboration models, by means of the proposed formalisation it can be observed pools that exchange message flow tokens, while in each pool the execution is rendered by the movements of the sequence flow tokens in the process. In this setting, there is a clear difference between the notion of safeness directly defined on BPMN collaborations with respect to that defined on Petri Nets and applied to the Petri Nets resulting from the translation of BPMN collaborations. Safeness of a BPMN collaboration only refers to tokens on the sequence edges of the involved processes, while in its Petri Nets translation it refers to token both on message and sequence edges. Indeed, such distinction is not considered in the available mappings [29, 50], because a message is rendered as a (standard) token in a place. Hence, a safe BPMN collaboration, where the same message is sent more than once (e.g., via a loop), is erroneously considered unsafe by relying on the Petri Nets notion (i.e., 1-boundedness), because enqueued messages are rendered as a place with more than one token. Therefore, the notion of safeness defined for Petri Nets cannot be safely applied as it is to collaboration models. Similarly, regarding the soundness property, it is possible to consider different notions of soundness according to the requirements imposed on message queues (e.g., ignoring or not pending messages). Again, due to lack of distinction between message and sequence edges, these fine-grained reasoning are precluded using the current translations from BPMN to Petri Nets.

The study of BPMN models via the Petri Nets based frameworks has another limitation concerning the management of the terminate event. Most of the available mappings, such as the ones in [50] and [43], do not consider the terminate event, while in the one provided in [29], terminate events are treated as a special type of error events, which however occur mainly on subprocesses, whose translation assumes safeness. This does not allow reasoning on most models including the terminate event. Moreover, given the local nature of Petri Nets transitions, such cancellation patterns are difficult to handle. This is confirmed in [92], stating that modelling a vacuum cleaner, (i.e., a construct able to remove all the tokens from a given fragment of a net) is possible but results in a spaghetti-like model.

The ability of the given formal framework to properly distinguish sequence flow tokens and message flow tokens, together with the treatment of the terminate event without any of the restrictions mentioned above, leads to provide a more precise classification of the BPMN models as synthesised

by the Euler diagram in Figure 4.3.

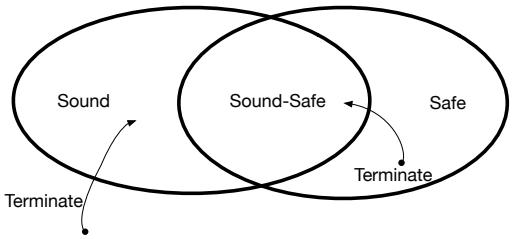


Figure 4.3: Impact of the Terminate Event on the Classification.

The diagram underlines reasoning that can be done at process level on soundness. Here it emerges how the terminate event can affect model soundness, as using it in place of an end event may render sound a model that was unsound.

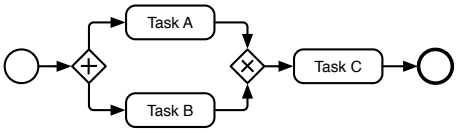


Figure 4.4: Unsound Process.

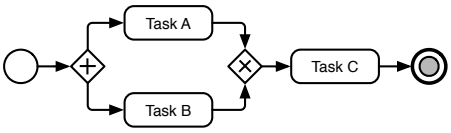


Figure 4.5: Sound Process.

An example is the process in Figure 4.4; it is a simple process that first runs in parallel *Task A* and *Task B*, then executes two times *Task C*. According to the proposed classification the model is unsound. In fact, there is a marking where the end event has two tokens.

Now, by considering another model (Figure 4.5), obtained from the one in Figure 4.4 by replacing the end event with a terminate event, it results that it is sound. This is due to the behaviour of the terminate event that, when reached, removes all tokens in the process. Notably, although the two models are quite similar, in terms of the classification they result to be significantly different.

4.4 Relationships among Properties

This section studies the relationships among the considered properties both at the process and collaboration level. In particular it investigates the relationship between (i) well-structuredness and safeness, (ii) well-structuredness and soundness, and (iii) safeness and soundness. The section includes the sketches of the proofs, while the complete proofs of the results are reported in the Appendix C.1.

4.4.1 Well-structuredness vs. Safeness in BPMN

Considering well-structuredness and safeness it is here demonstrated that all well-structured models are safe (Theorem 1), and that the reverse does not hold. To this aim, first it is shown that a process in the initial state is cs-safe (Lemma 1). Then, it is shown 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 Figure 3.2 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 (see Appendix B).

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

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* $isInitEl(P, \sigma')$ *and* $\langle P, \sigma' \rangle \rightarrow^* \sigma$.

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

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

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

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

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

Proof (sketch). 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.*

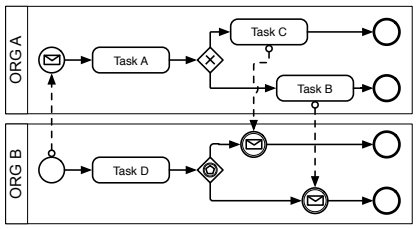


Figure 4.6: Safe BPMN Collaboration not Well-structured

Proof (sketch). Showing that if $\langle P, \sigma \rangle \rightarrow^* \sigma'$ then $\langle P, \sigma' \rangle$ is cs-safe, it follows 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 Figure. 4.6 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.

Now the previous results can be extended to collaborations.

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

Proof (sketch). By contradiction. □

4.4.2 Well-structuredness vs. Soundness in BPMN

Considering the relationship between well-structuredness and soundness it is proved that a well-structured process is always sound (Theorem 3), but there are sound processes that are not well-structured. To this aim, first it is shown 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 (the formal definition is relegated to Appendix B).

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). By induction on the structure of well-structured core process. □

Theorem 3. *Let $isWS(P)$, then P is sound.*

Proof (sketch). 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 Figure 4.7. This process is surely unstructured, and it is also trivially sound, since it is always possible to reach an end event without leaving tokens marking the sequence flows.

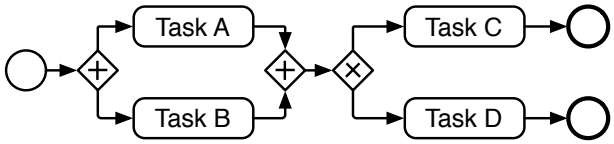


Figure 4.7: Example of Sound Process not Well-Structured.

However, Theorem 3 does not extend to the collaboration level. In fact, when putting 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, is$WS(C)$ does not imply C is sound.*

Proof (sketch). By contradiction. □

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

Proof (sketch). By contradiction. □

4.4.3 Safeness vs. Soundness in BPMN

Considering the relationship between safeness and soundness this section demonstrates that there are unsafe models that are sound. This is a peculiarity of BPMN, faithfully implemented in the provided semantics, thank to its capability to support the terminate event.

At process level it results as following.

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

Proof (sketch). By contradiction. □

Concerning the collaboration level, it results that there are unsafe collaborations that could be 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). By contradiction. □

Running Example 6. *Considering the collaboration in the running example, PC Chair and Contact Author processes are both safe and sound, while the process of the Reviewer is unsafe but sound, since the terminate event permits to reach a marking where all edges are umarked. The collaboration is unsafe, and it is also unsound but message-relaxed sound, since there could be messages in the message lists.* □

4.5 Class of (Sound) Unsafe Models

Soundness of business process models has been proposed as the main correctness criterion verifying different business process formalisations. However, reasoning directly on BPMN, soundness does not guarantee a model to be free of errors. In fact, as proved in Section 4.4.1, there are sound BPMN models that are not safe. Indeed, the lack of safeness may cause issues concerning process execution, related e.g. to erroneous synchronizations among concurrent control flows. Thus, this section refers to the class of unsafe models, including models with an arbitrary topology and concurrent behaviour. Specifically, first a motivating scenario is introduced to clarify the issue to address (Section 4.5.1), then the approach to resolve such a issue is presented (Section 4.5.2) and finally, it is shown how the proposed solution works into practice (Section 4.5.3).

4.5.1 Motivating Scenario

This section refers to process models with an arbitrary topology including concurrent behaviour, which may lead to the occurrence of erroneous synchronizations. These processes are typically discarded by the modelling approaches proposed in the literature, as they are over suspected of carrying bugs. Unfortunately, this attitude significantly limits the use of concurrency in business process modelling, which is an important feature in modern systems and organizations. Instead, in these cases the designer could keep the 'offending' model and solve the issue by better clarifying the intended behaviour. In fact, the problem typically is not in the model itself but it is due to the underspecification of the BPMN standard in dealing with concurrency issues within a single process instance.

The relevance of such an issue is pointed out also by studies stating that an increase in the level of concurrency for BPMN models implies an increase in modeling error probability [56, 63]. Thus, this section focusses on the management of concurrent behaviour in a single process instance, with the aim of achieving synchronization correctness in unstructured processes.

To better clarify the issues, a simplified version of the running example presented in Section 2.2.2 is introduced.

It is modelled as the collaboration in Figure 4.8. The participants as

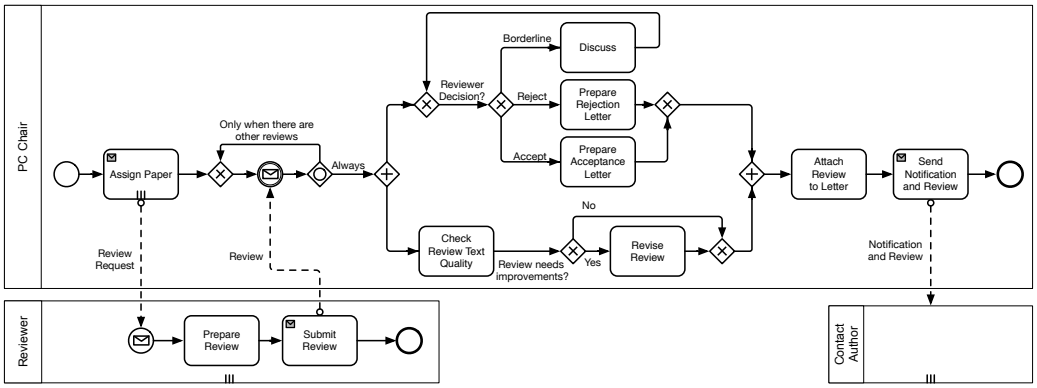


Figure 4.8: Paper Reviewing Collaboration Model (revisited).

usual are: the **PC Chair**, the **Reviewer**, and the **Contact Author**, whose role is modelled here as a multi-instance blackbox pool since details on the author behaviour are not relevant to the current purposes.

The reviewing process is started by the chair, who assigns (via a parallel multi-instance activity) each submitted paper to a reviewer. Then, the chair receives the reviews and evaluates them. In particular, as soon as a review is received, the chair starts its evaluation and is immediately ready to receive and process another review. This behaviour is rendered in BPMN by means of a loop, realized via an OR-Split gateway and a XOR-Join gateway, whose single iteration consists of receiving a paper review and starting its evaluation. The evaluation of each review is modelled by the process fragment enclosed by the AND-Split gateway and the AND-Join one. Indeed, the evaluation proceeds along two concurrent control flows: *(bottom branch)* the chair checks the quality of the received review and, if necessary, he/she revises it to improve and *(top branch)*, according to the reviewer decision, the chair prepares the acceptance/rejection letter or, if the paper requires further discussion, the decision is postponed. In the last part of the process (after the AND-Join gateway), the chair attaches the review to the notification letter, and sends it to the contact author.

Despite its simplicity, the model described so far hides some subtleties that may affect its correct enactment. For instance, it may happen that an author of a paper will receive a notification with attached the review of another paper.

Considering the reception of the review for a paper, say $paper1$, this event produces a token that activates the OR-Split; assuming that other reviews are waited, the OR gateway produces in its own turn two tokens: one is used to reactivate the receiving message intermediate event, while the other to

activate the evaluation of *paper*1's review. This latter token is split into two tokens for activating the two evaluation branches described above. Then, a review for another paper, say *paper*2, is received and dealt with in a similar way. The evaluations of the two reviews proceed, hence, along two concurrent control flows. After some steps, it could arise the current situation: *(i)* *paper*1's review has been revised by the chair and a corresponding token reached the AND-Join gateway from the bottom incoming flow, while the other *paper*1 token is still marking the *Discuss* task, as the paper received a borderline score; *(ii)* *paper*2 received a reject score, thus, while the chair is still checking the review quality, a *paper*2 token reached the AND-Join gateway from the top incoming flow.

In this situation, the two incoming flows of the AND-Join carry a token. Thus, according to the standard semantics of BPMN, the AND gateway triggers the flow through its outgoing sequence flow. In fact, the semantics does not distinguish tokens related to the evaluation of the *paper*1's review from those related to the *paper*2's one. This **erroneous synchronization** of tokens allows the process execution to continue with the notification task, using the revised review of *paper*1 and the rejection letter of *paper*2. Notably, in order to have the situation described above, during the execution of the considered process more than one token must concurrently transit along the same sequence flow. In the reviewing scenario this happens each time a review is assigned as result of the OR-Split behaviour specification. Moreover, the other condition leading to situations of erroneous synchronization is the presence of *concurrent control flows*, where the generated multiple tokens are split and then have to be synchronized. In the presented scenario, the concurrent control flows correspond to the two evaluation activities performed by the chair.

In the following it is presented a possible approach to avoid erroneous synchronizations to take place when the above conditions are met.

4.5.2 Methodology

Different solutions can be proposed in order to overcome the addressed concurrency issues. With reference to the introduced reviewing scenario, a first solution proposes to model the PC Chair by means of two processes: one that assigns papers, collects reviews and instantiates the other (multi-instance) process, whose instances separately deal with the evaluation of paper reviews. As no interaction can take place among these instances, erroneous synchronizations cannot emerge. A second solution is to put in sequence the various evaluation activities performed by the chair. This avoids concurrent flows and, hence, the possibility of erroneous synchronizations. However, the first alternative does not fit well with the reality, as the behaviour of a single human person is split into two separate processes, one of which is

multi-instance. The second one, instead, imposes to put in sequence a set of activities that originally were parallel. Most of all, the two alternative solutions require an alteration of the original structure, as well as of the semantics, of the designed process. This requires the designer to be expert enough to identify the concurrency issue in his model and, then, to solve it by properly restructuring the model.

The approach proposed here provides instead a general solution to the problem, without altering the structure of the process. It is based on an advanced use of BPMN text annotations to enrich the model with information suitable to deal with concurrent execution of control flows. Moreover, this study contributes by refining the process execution semantics by taking into account token identities to avoid erroneous synchronizations.

The use of tokens with identity enables the AND-Join gateway in the motivating scenario to distinguish the two incoming flows, hence avoiding the erroneous synchronization. In fact, only tokens with the same identity, i.e. referring to the same paper review, synchronize. When synchronization cannot take place, the incoming tokens just wait for the arrival of 'brother' tokens.

Since token identity can evolve during the process execution, as it can have different meanings in different parts of a process, the token can be identified by means of different (unique) identifiers, whose scope can be limited to the part of interest in the process. This is achieved by enriching BPMN models with additional information, via specific text annotations on sequence flows, called *check-in* and *check-out*, that enclose the part of process defining the scope of a token identifier. Such scope is application specific, hence it must be the designer in charge of explicitly specifying this information on the process model.

In detail, a **Check-in** represents a point of the process from where the identity of the traversing tokens is enriched with a fresh identifier; a **Check-out** represents a point of the process where the identifiers created by the corresponding check-in are no longer needed and, hence, are removed from the identity of the traversing tokens.

Check-ins and check-outs are identified by their names, ranged over by n. Each check-out must be correlated with one check-in, i.e. there is a check-in in the process model with the same name; on the other hand, each check-in is correlated with zero or more check-outs. Graphically (Figure 4.9) a check-in (resp. check-out) is a standard BPMN text annotation with the peculiarity of being attached to a sequence flow and of enclosing a text of the form Check-in (n) (resp. Check-out (n)).

As already mentioned, a token can have different meanings in the process. This can be achieved by means of more check-ins. Notably, a check-in can occur inside a check-in/check-out block. Therefore, the identity of a token

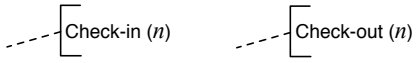


Figure 4.9: Graphical notation of check-in and check-out annotations.

is defined as a set T of pairs of the form (n, id), where n is the name of a check-in traversed by the token and id is an identifier freshly[3] generated by the check-in n. When a token is generated by the activation of a start event, it is initialized with a default identity represented by the set $\{(init, 0)\}$, where $init$ is a reserved check-in name and 0 is an identifier. The identity of a token changes only when it traverses check-in or check-out points, while its flow during the process execution is regulated by the standard BPMN execution semantics unless when it meets a synchronization point (i.e., an AND or an OR join gateway). In detail, when a token traverses a check-in point n, its identity is not altered if it already contains an identifier generated by n, otherwise the token identity is enriched with a new identifier pair. Formally, the token identity evolution determined by a check-in is defined by function *TraverseCheckIn* that, given as input a check-in name and the identity set of an incoming token, returns as output the identity of the outgoing token

$$\mathit{TraverseCheckIn}(n, T) = \begin{cases} T & \text{if } (n, id) \in T \\ T \cup \{(n, \mathit{fresh}(n))\} & \text{otherwise} \end{cases}$$

where $\mathit{fresh}(n)$ is a function that returns a fresh identifier for the check-in n (notably, this function can be straightforwardly implemented by relying on a counter local to each check-in).

Dually, when a token traverses a check-out point n, its identity is not altered if it does not contain an identifier generated by n, otherwise the corresponding identitifier pair is removed from the identity set of the token. Formally, the token identity evolution determined by a check-out is defined by the following function.

$$\mathit{TraverseCheckOut}(n, T) = \begin{cases} T \setminus (n, id) & \text{if } (n, id) \in T \\ T & \text{otherwise} \end{cases}$$

Notably, the synchronization requires a complete match of identities among tokens, which means that the identity sets must have the same pairs. It is also worth noticing that, in case of synchronization of tokens whose identity is given by the default value $\{(init, 0)\}$, the refined semantics coincides with the one prescribed by the BPMN standard. In other words, the proposed semantics is conservative with respect to the standard one, i.e., if no check-in and check-out annotations are introduced in the model then the two semantics coincide.

[3]An identifier, generated by a check-in n, is called *fresh* if it is different from all other identifiers previously generated by the check-in n.

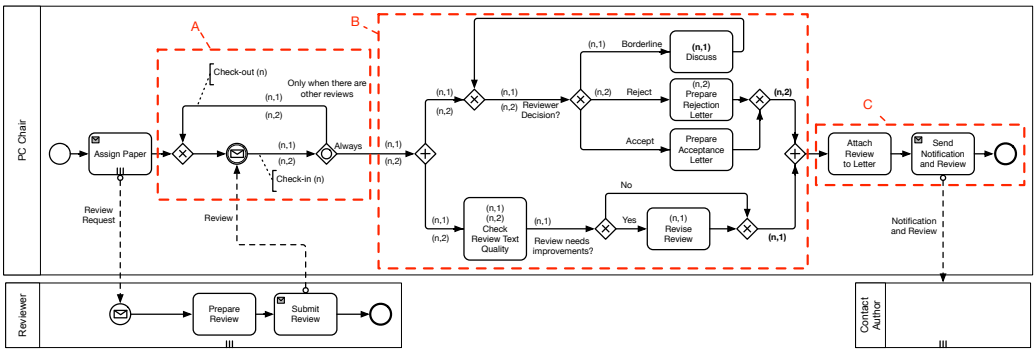


Figure 4.10: Paper Reviewing Collaboration Execution with Tokens id.

4.5.3 Approach at Work

This section illustrates how the approach can be applied in practice. Figure 4.10 shows how check-ins and check-outs are used to specify in which part of the process, within the PC Chair pool, tokens represent the control flows of the paper evaluation. For the sake of presentation, three relevant parts of the process, named A, B and C, are identified. Moreover, to show the flow of each token, the corresponding path in the process is marked with token identities (curly brackets and the default identifier are omitted).

At the beginning of the execution, the token placed on the start event has the default identity, represented by the set $\{(init, 0)\}$. Then, the token enters into, and hence activates, Part A of the process, which is a token generator. Thus, for each received review, a new token identity has to be generated. To this aim, the designer introduced a check-in named n so that, as soon as a token traverses the check-in, its identity is enriched with the new identifier $(n, 1)$, denoting that the token is related to the review of $paper1$. The OR-Split gateway then splits the token into two tokens with the same identity $\{(init, 0), (n, 1)\}$. One of the generated tokens will go back into the loop, traversing the check-out point n and hence loosing its $paper1$ identity. This shows the usefulness of the check-out annotation: sometimes it is necessary that a token looses its identity as, e.g., it enters in a path where it is merely used as a control flow signal. In the shown case, the only purpose of the considered token is to activate a new iteration of the loop; in fact, without loosing its identity the token would fail in doing this. Instead, the other $paper1$ token will go into Part B of the process. This token will cross the AND-Split gateway and the evaluation of the $paper1$'s review will start. From this point the execution proceeds, thus a new token with identity $\{(init, 0), (n, 2)\}$ enters in the game, and the marking represented by the tokens whose identity is written in bold in Figure 4.10 is reached. Now, the AND-Join gateway has two incoming tokens, one per each incoming edge, and thus evaluates their synchronization. Anyway, according to the refined

semantics, the synchronization does not take place, as the two incoming tokens have different identities. Therefore, they remain in the edges waiting for their brothers. In this way, the appropriate synchronization will take place, the tokens will go through Part C of the process, and a notification to the author with attached the review of the corresponding paper will be sent.

Extended explanations about the use of (more than one) check-in and check-out can be found in [20].

4.6 Compositionality of Safeness and Soundness

This section studies the compositionality of safeness and soundness, i.e. how the behaviour of processes affects that of the entire resulting collaboration. In particular, it is shown the interrelationship between the studied properties at collaboration and at process level.

4.6.1 On Compositionality of Safeness

Here it is shown 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). By contradiction (see Appendix C.1). □

It can be also shown 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, as in the following examples.

Running Example 7. *In the running example of the previous chapter, the collaboration is composed by two safe processes and an unsafe one. In fact, focussing on the process of the Reviewer, it is easy to see that it is not safe: if the results of the two checks are negative, two tokens arrive at the XOR-Join, producing multiple tokens on its outgoing edge. Now, considering this process together with the safe processes of the PC Chair and the Contact Author, the resulting collaboration is not safe. Indeed, the event based gateway, which waits for a review, forwards only the first token arriving on one of the two paths. As soon as a review is received, the PC Chair process proceeds and completes. Thus, due to the lack of safeness, the Reviewer will send another rejection review, but no more rejection messages arriving from the Reviewer will be considered by the PC Chair.* □

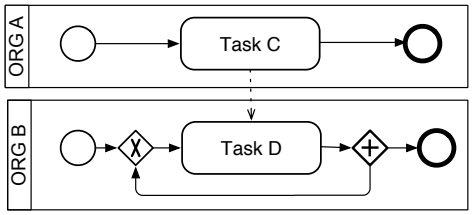


Figure 4.11: Safe Collaboration with Safe and Unsafe Processes.

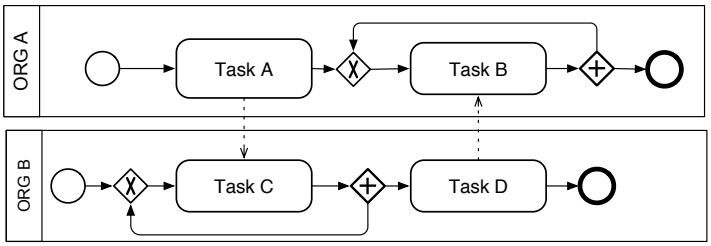


Figure 4.12: Safe Collaboration with Unsafe Processes.

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 Figure 4.11. Focussing only on the process in* ORG B *it is easy to 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, considering 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, there is no problem of safeness.* □

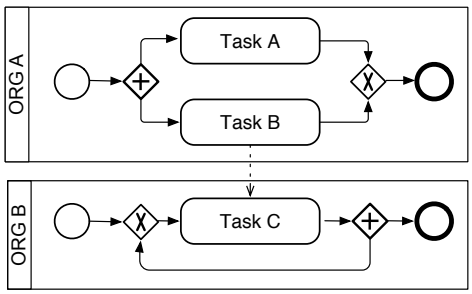


Figure 4.13: Unsafe Collaboration with Unsafe Processes.

Example 2. *In Figure 4.12 there are two unsafe processes, since each of them contains a loop capable of generating an unbounded number of tokens.*

However, considering 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 Figure 4.13 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.* □

4.6.2 On Compositionality of Soundness

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

Theorem 9. *Let C be a collaboration, if some processes in C are unsound then C is unsound.*

Proof (sketch). By contradiction (see Appendix C.1). □

On the other hand, putting together sound processes, the obtained collaboration could be either sound or unsound, since one has also to consider messages. It can happen that either a process waits for a message that will never be received or it receives more than the number of messages it is able to process. Here are some examples.

Running Example 8. *In the running example of the previous chapter, the collaboration is composed by sound processes. In fact, the PC Chair and Contact Author processes are well-structured, thus sound. Focussing on the process of the Reviewer, it is also sound since when it completes the terminate event aborts all eventually 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 Figure 4.14 there is a collaboration resulting from the composition of two sound processes. Focussing only on the processes in* ORG A *and* ORG B *it can be immediately noted 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*

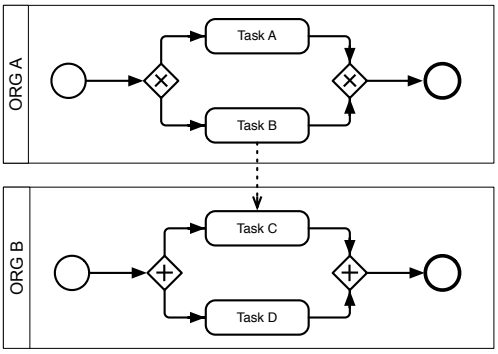


Figure 4.14: Example of Unsound Collaboration with Sound Processes.

gateway cannot be activated, thus the process of ORG B *cannot complete its execution.* □

Example 5. *Also the collaboration in Figure 4.15 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.* □

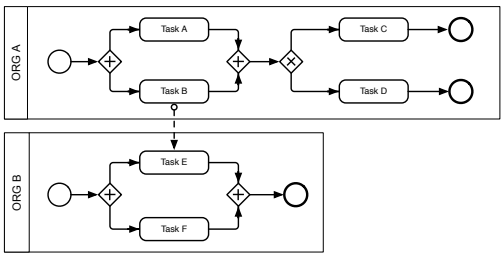


Figure 4.15: Sound Collaboration with Sound Processes.

Remark 1. *Well-structuredness and message-relaxed soundness compositionality are not considered. 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.*

4.7 Comparison with other Approaches

This section provides a formal characterisation of well-structured BPMN models. To do that, the main inspiration is the definition of well-structuredness given in [44]. Other attempts are also available in the literature. Van der Aalst et al. [98] state that a workflow net is well-structured

if the split/join constructions are properly nested. El-Saber and Boronat [35] propose a formal definition of well-structured processes, in terms of a rewriting logic, but they do not extend this definition at collaboration level.

Then safeness is considered, showing that this is a significant correctness property. Dijkman et al. [29] 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; this thesis gives instead a precise characterisation of safeness for both BPMN processes and collaborations. Other approaches introducing mapping from BPMN to formal languages, such as YAWL [24] and COWS [75], do not consider safeness, even if it is recognised as an important characteristic [17].

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 [54], and for Workflow Nets by van der Aalst [104] 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 [99]. 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 [86] in the field of the Global Interaction Nets, in order to detect errors in technology-independent collaborative business processes models. However, this approach does not apply to BPMN, which is the modelling notation aimed by the presented study. Concerning message-relaxed soundness, the work has been motivated by Puhlmann and Weske [79], who define interaction soundness, which in turn is based on lazy soundness [81]. 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. The provided framework instead natively implements the BPMN communication model via an asynchronous approach. Moreover, the interaction soundness assumes structural soundness as a necessary condition that is here relaxed. Thus, the presented investigation of properties at collaboration level provides novel insights with respect to the state-of-the-art of BPMN formal studies.

Part III

Extended Framework

Chapter 5

OR-Join Gateway

The BPMN lack of a formal semantics may represent a big issue when considering BPMN elements that have a particularly tricky behaviour, such as the OR-Join [101]. Indeed, this is a very important feature of BPMN because it is used to synchronise two or more parallel flows according to a specific (and non trivial) state on business process execution. An informal description of its behaviour can lead to different interpretations, and hence implementations. This can result on process implementations using models that do not fit with designer expectations.

This chapter aims at formally specifying the OR-Join semantics of BPMN process models. This paves the way not only to formal reasoning, but also to driven implementations of process-aware IT systems supporting the behaviour of the OR-Join compliant with BPMN 2.0. The focus on the OR-Join is due not only because of its semantic complexity, but also to its practical impact, as that is a convenient way to synchronize parallel control flows [33]. Its large use is also confirmed by the number of models containing it (316 out of 7.541 BPMN 2.0 collaborations available in the BPM Academic Initiative public repository [48]). The necessity of a faithful OR-Join semantics is also confirmed by the difficulty of replacing this construct by means of AND and XOR gateways. Indeed, there are simple process models with an OR-Join that cannot be replaced, in the sense that its synchronisation behaviour cannot be obtained by any combination of other gateways [39].

Tackling the above issues, the chapter is organised as follows: firstly, it provides a direct *global* formalisation compliant with the OR-Join semantics reported in the current BPMN 2.0 standard specification (Section 5.3); then, it also provides a *local* variant of the semantics, devised to more efficiently determine the OR-Join activation, as it depends only on information local to the considered OR-Join (Section 5.4). Thus, the global semantics is introduced as the formal reference, while the local one to be used for implementations. The soundness of the approach is given by the formal proof of

their correspondence. The gap between the two semantics is also confirmed by the comparison between them (Section 5.5). Towards the formalisation, it also possible to characterise the correctness properties considered in the thesis in this extended framework (Section 5.6), showing the effects of the OR-Join gateway on process execution.

Notably, the intricate semantics of the OR-Join has led to consider only a small subset of BPMN elements, tough able to illustrate certain difficulties when dealing with the OR-Join gateway. Indeed, this chapter focuses only on process models and abstracts from information that are not relevant for the described purpose.

Highlights. Distinctive aspects of the proposed semantics are:

- it provides a direct global formalisation compliant with the OR-Join semantics reported in the current BPMN 2.0 standard specification;
- it provides a local, more efficient, variant of the semantics;
- it allows a precise classification of BPMN models including OR-Joins, by clarifying ambiguous OR-Join behaviours.

5.1 Running Example

To better understand the motivation in considering the OR-Join gateway, this section presents a revised version of the running example presented in Section 2.2.2. The scenario in Figure 5.1 shows the process fulfilled by the Contact Author of a paper to finalise and submit an article for a scientific conference.

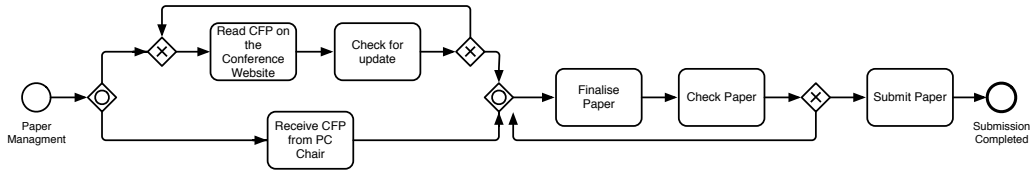


Figure 5.1: Paper Submission Process.

In order to submit a paper to a conference, firstly the author has to read the Call For Papers (CFP). He/she may either find the CFP information on the conference website or receive it by email from the PC Chair, or both. This is rendered by including related tasks in a block composed of two OR gateways: an OR-Split, used to fork the flow into two branches after a decision; and an OR-Join that acts as a synchronisation point. If the information is retrieved online, the author checks if the website is updated with the final version of the CFP. In case only the first version of the CFP is present,

the author checks for updates until the final version is produced. This is rendered by means of an XOR-Join and XOR-Split loop. Once submission information is obtained, the author can finalise the paper. Before submitting it, he/she checks the manuscript and if necessary, improves it until the final version is obtained. When the author is satisfied, the paper is submitted and the process completes.

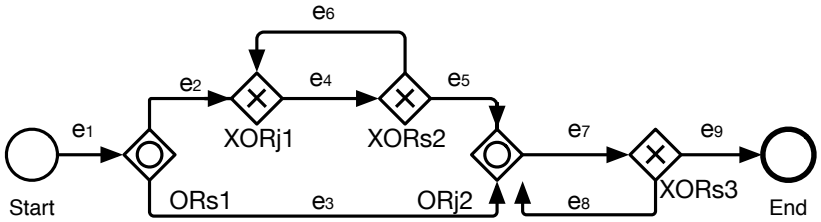


Figure 5.2: Paper Submission Process Structure.

In the rest of the chapter and for the purpose of the study tasks are intentionally left out, since they do not affect the OR-Join execution [18]. Considering the running example, its process structure is depicted in Figure 5.2. For an easier readability and understandability of how the semantics works, labels are assigned to each BPMN element.

5.2 Towards the OR-Join Formal Definition

This section discusses in detail the semantics of BPMN 2.0 OR-Join as provided in the OMG specification and clarifies it by means of some examples. Then, some preliminary notions are introduced.

5.2.1 From BPMN 2.0 Specification to Process Execution

The OR-Join semantics is quite complex, both from the definition point of view and from the computational point of view, in terms of determining whether an OR-Join is active. In this thesis, the characteristics of the OR-Join are distilled, from a detailed reading of the BPMN specification which is reported in Figure 5.3 where, as a matter of terminology, Inclusive Gateway stands for OR-Join, while Sequence Flow for sequence edge.

From the standard it is clear that the OR-Join has a *non-local semantics* and its activation may depend on the marking evolution considering the whole diagram.

More in detail, given an OR-Join with a token in at least one of its incoming edges, it has to wait for a token that is in a path ending in a

---

The Inclusive Gateway is activated if:

- At least one incoming Sequence Flow has at least one token and
- For every directed path formed by sequence flow that:
  - (i) starts with a Sequence Flow *f* of the diagram that has a token,
  - (ii) ends with an incoming Sequence Flow of the inclusive gateway that has no token,
  - (iii) does not visit the Inclusive Gateway.
- There is also a directed path formed by Sequence Flow that:
  - (iv) starts with *f*,
  - (v) ends with an incoming Sequence Flow of the inclusive gateway that has a token,
  - (vi) does not visit the Inclusive Gateway.

Figure 5.3: OR-Join Semantics According to the OMG Standard BPMN 2.0 (p. 436).

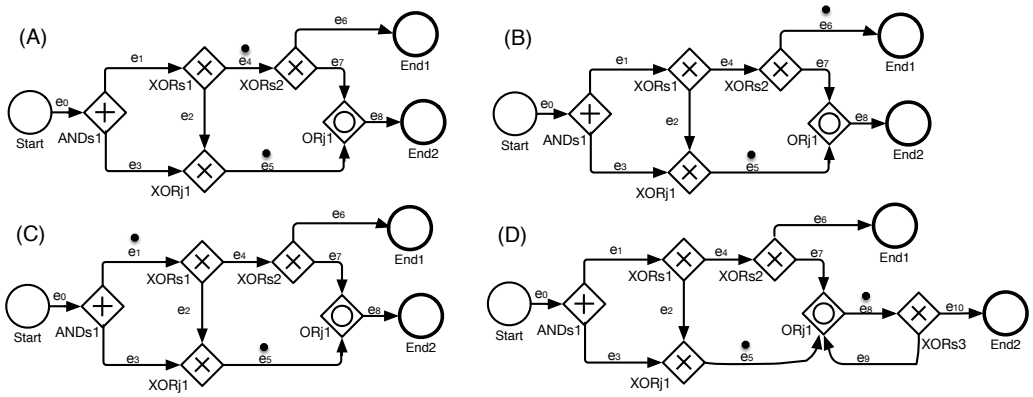


Figure 5.4: OR-Join Activation.

empty incoming edge of such OR-Join that does not visit the OR-Join itself. However, if this token is also in a path ending in a non-empty incoming edge, the OR-Join is activated and the execution can proceed without waiting for the synchronization.

Example 6. *In order to illustrate how the BPMN OR-Join specification works into practice, the example in Figure 5.4 (A) is used. In this case ORj1 has an incoming token in* e_5*, but it is not activated because it has to wait for the token in* e_4 *corresponding to f in the definition in Figure 5.3. Indeed, there is not another path from* e_4 *to* e_5*. However, if the token in* e_4 *moves to* e_6*, as in Figure 5.4 (B), the execution of ORj1 resumes, because now there is no marked path ending in* e_7*. Moreover, moving the token in* e_4 *back to* e_1*, as in Figure 5.4 (C), quite surprisingly ORj1 is activated, since this token*

can follow the path leading to e_5*. In this case, the OR-Join behaviour is quite anomalous; this is due to the fact that the model is unsafe.* □

Example 7. *To illustrate the effects of the condition "does not visit the* Inclusive Gateway*" in Figure 5.3, a variant of the previous process is considered. Here ORj1 is enclosed in a cycle in Figure 5.4 (D). Also in this case ORj1 is activated; indeed, although the token in* e_8 *is in a path ending in an empty edge incoming in ORj1 (i.e.,* e_9*), since this path visits ORj1 it is ignored.* □

5.2.2 Preliminaries

To define the formal semantics of a BPMN model some information needs to be extracted from the model by means of a pre-processing step. This information consists of: *(i.)* paths from each OR-Join backward to the start event (and their suffix sub-paths) that do not visit the inclusive gateway; *(ii.)* sequence edges involved in a cycle; and *(iii.)* dependences between OR-Joins. Only models with one start event are considered; this is not a limitation as in this setting each model can be rendered in this form.

For the purpose of the pre-processing, a process model is considered as a direct graph $G = (V, A)$ where: V is a set of *vertices*, ranged over by v and consisting of start events, end events, and gateways; and A is a set of *arrows*, consisting of triples (v_1, e, v_2) with $v_1 \neq v_2$ and $\mathsf{e} \in \mathbb{E}$, where $\mathbb{E}$ is the set of all (sequence) edges in a model. Since edges are uniquely identified in a BPMN model, it results that for each (v_1, e, v_2) in A there exists no triple $(v_1', \mathsf{e}', v_2')$ in A with $\mathsf{e}' = \mathsf{e}$. This allows to write, when convenient, (v_1, e, v_2) as e. Moreover, it is assumed that an OR-Join vertex is uniquely identified by the name of its outgoing edge.

A *path* in G, denoted by p, is a non-empty sequence of edges in A, where the third element of a triple is equal to the first of the next triple in the sequence, if any. A path that ends in its starting vertex is called *cycle*. For example, in the model in Figure 5.2 there are the following cycles: $(\mathsf{e}_7, \mathsf{e}_8), (\mathsf{e}_4, \mathsf{e}_6)$. Given a path p of the form $(v_0, \mathsf{e}_0, v_1), \ldots, (v_{k-1}, \mathsf{e}_{k-1}, v_k)$, notations $\mathsf{first}(p)$ and $\mathsf{last}(p)$ indicate the starting edge e_0 and the ending edge e_{k-1} of p, respectively.

Moreover, $\mathbb{P}$ refers to the set of all the paths in G and $\mathcal{P} : \mathbb{E} \to 2^{\mathbb{P}}$ is a function that, given as input an edge $\mathsf{e} \in \mathbb{E}$ returns the set of all paths ending in the OR-Join uniquely identified by e and starting from all vertices between the start event and the OR-Join, which do not visit the considered OR-Join. Notably, this function returns a finite set of paths, because cycles within paths are not repeated. While computing $\mathcal{P}$, it is possible also to compute the set $\mathbb{C} \subseteq \mathbb{E}$ of edges included in a cycle. Concerning the example in Figure 5.2, $\mathcal{P}(\mathsf{e}_7) = \{(\mathsf{e}_5), (\mathsf{e}_3), (\mathsf{e}_1, \mathsf{e}_3), (\mathsf{e}_1, \mathsf{e}_2, \mathsf{e}_4, \mathsf{e}_5), (\mathsf{e}_1, \mathsf{e}_2, \mathsf{e}_4, \mathsf{e}_6, \mathsf{e}_5)\}$, and $\mathbb{C} = \{\mathsf{e}_4, \mathsf{e}_6, \mathsf{e}_7, \mathsf{e}_8, \}$.

Finally, to properly formalise the OR-Join semantics in presence of vicious circles (i.e., to keep blocked the execution, see discussion above), it is necessary to detect for each OR-Join if there are other OR-Joins from which it depends. This is expressed as a boolean predicate $noDep : \mathbb{E} \rightarrow \{true, false\}$, which taken as input an edge e identifying an OR-Join, it holds if no other OR-Join mutually depends from the OR-Join uniquely identified by its outgoing edge e.

The pre-processing execution relies on existing graph theory procedures. Given the graph $G = (V, A)$ representing the process model, the proposed pre-processing generates all possible paths between two nodes by combining minimal cycles with simple paths. More in detail, a simple path is basically a list of edges where nodes are visited at most once, while a minimal cycle is a simple path forming a cycle. Hence, the combination of such elements is done by adding each cycle to each path, if and only if the path and the cycle have a common node. This procedure is used to determine all the possible paths through which a token can reach the OR-Join and the OR-Join dependences. To do that the *jGraphT* (www.jgrapht.org) Java library is used since it is able to manage graphs. Thanks to this library it is possible to capture cycles with a customised implementation of the Szwarcfiter and Lauer algorithm [91] and paths by using a Dijkstra-like algorithm [30]. The code of the pre-processing is available at https://bitbucket.org/proslabteam/or-joinpreprocessing.

5.3 Formal Framework

According to the OMG standard the semantics of the OR-Join requires global information about the state of the whole model. Thus, a direct, one-to-one, formalisation of this description has to be given with a *global* style, i.e., it is based on a notion of state storing information about tokens distribution over the whole model. This section presents the global formalisation. Specifically, it first presents the syntax and operational semantics.

5.3.1 Syntax

To enable a formal treatment of BPMN models including the OR-Join, a BNF syntax of the model structure is defined in Figure 5.5. To simplify the formal treatment, in the proposed grammar only simple start and end events are considered. Moreover, also the event-based gateway is omitted. Considering the OR-Join gateway, the correspondence between the textual notation used here and the graphical notation of BPMN is as follows:

- $\mathsf{orSplit}(\mathsf{e}_i, E_o)$ represents an OR-Split gateway with incoming edge e_i and outgoing edges E_o;

$$P ::= \mathsf{start}(\mathsf{e}_o) \mid \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}) \mid \mathsf{andSplit}(\mathsf{e}_i, E_o) \mid \mathsf{andJoin}(E_i, \mathsf{e}_o) \mid \mathsf{xorSplit}(\mathsf{e}_i, E_o) \mid \mathsf{xorJoin}(E_i, \mathsf{e}_o) \mid \mathsf{orSplit}(E_i, \mathsf{e}_o) \mid \mathsf{orJoin}(E_i, \mathsf{e}_o) \mid P_1 \| P_2$$

Figure 5.5: Syntax of BPMN Process Structures.

$$\langle \mathsf{start}(\mathsf{e}), \sigma_0 \rangle \rightarrow_G inc(\sigma_0, \mathsf{e}) \quad (G\text{-}Start)$$

$$\langle \mathsf{end}(\mathsf{e}, \mathsf{e}'), \sigma \rangle \rightarrow_G inc(dec(\sigma, \mathsf{e}), \mathsf{e}') \quad \sigma(\mathsf{e}) > 0 \quad (G\text{-}End)$$

$$\langle \mathsf{andSplit}(\mathsf{e}, E), \sigma \rangle \rightarrow_G inc(dec(\sigma, \mathsf{e}), E) \quad \sigma(\mathsf{e}) > 0 \quad (G\text{-}AndSplit)$$

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

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

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

$$\langle \mathsf{orSplit}(\mathsf{e}, E_1 \sqcup E_2), \sigma \rangle \rightarrow_G inc(dec(\sigma, \mathsf{e}), E_1) \quad \sigma(\mathsf{e}) > 0 \quad E_1 \neq \varnothing \quad (G\text{-}OrSplit)$$

$$\langle \mathsf{orJoin}(E_1 \sqcup E_2, \mathsf{e}), \sigma \rangle \rightarrow_G inc(dec(\sigma, E_1), \mathsf{e}) \quad \forall \mathsf{e}' \in E_1.\sigma(\mathsf{e}') > 0 \quad \forall \mathsf{e}' \in E_2.\sigma(\mathsf{e}') = 0 \quad E_1 \neq \varnothing \quad \forall p_1 \in \Pi.\exists\, p_2 \in \Pi_{p_1} \quad (G\text{-}OrJoin)$$

$$\frac{\langle P_1, \sigma \rangle \rightarrow_G \sigma'}{\langle P_1 \parallel P_2, \sigma \rangle \rightarrow_G \sigma'} \ (G\text{-}Int_1) \qquad \frac{\langle P_2, \sigma \rangle \rightarrow_G \sigma'}{\langle P_1 \parallel P_2, \sigma \rangle \rightarrow_G \sigma'} \ (G\text{-}Int_2)$$

Figure 5.6: BPMN Global Semantics.

- $\mathsf{orJoin}(E_i, \mathsf{e}_o)$ represents an OR-Join gateway with incoming edges E_i and outgoing edge e_o.

Notice, the one-to-one correspondence between the syntax used here to represent an OR gateway and the graphical notation of BPMN is reported in detail in the Appendix A.2.

5.3.2 Semantics

This section formalises this global perspective of the BPMN semantics. The proposed operational semantics is given in terms of configurations of the form $\langle P, \sigma, \mathcal{P} \rangle$, where: P is a process structure; σ is the execution state, storing for each edge the current number of tokens marking it; and $\mathcal{P}$ is the function that associates to each OR-Join gateway all paths that are incoming to it, not visiting it, and starting from marked edges (it results from pre-processing, see Section 5.2.2). The *inital state*, where all edges are unmarked, is denoted by σ_0; formally, $\sigma_0(\mathsf{e}) = 0 \ \ \forall \mathsf{e} \in \mathbb{E}$.

The reduction relation over configurations, written $\rightarrow_G$ and defined by the rules in Figure 5.6, formalises the execution of a process in terms of edge marking evolution. Since such execution only affects the process state, for

the sake of presentation, the structure and $\mathcal{P}$ are omitted from the target configuration of the transition. Moreover, since $\mathcal{P}$ is exploited only by the OR-Join rule, it will also be omitted from the source configuration. Thus, $\langle P, \sigma, \mathcal{P}\rangle \rightarrow_G \langle P, \sigma', \mathcal{P}\rangle$ shall be usually written as $\langle P, \sigma\rangle \rightarrow_G \sigma'$.

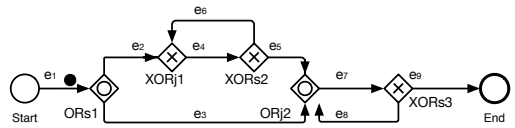
The operational rules for the OR-Split and the OR-Join gateway are as follows. Rule *G-OrSplit* is activated when there is a token in the incoming edge of an OR-Split gateway, which is then removed while a token is added in some outgoing edges (at least one). Notably, the rule makes use of operator $\sqcup$, denoting the disjoint union of sets, i.e. $E_1 \sqcup E_2$ stands for $E_1 \cup E_2$ if $E_1 \cap E_2 = \varnothing$, it is undefined otherwise.

Rule *G-OrJoin* defines the semantics of the OR-Join gateway. The operator $\sqcup$ is used to split the set of edges incoming in the OR-Join into two disjoint sets, E_1 and E_2, such that one contains marked edges ($\forall \mathsf{e}' \in E_1.\sigma(\mathsf{e}') > 0$) and the other one contains unmarked edges ($\forall \mathsf{e}' \in E_2.\sigma(\mathsf{e}') = 0$). To make clear the correspondence with the BPMN 2.0 specification, the BPMN standard is quoted in the description of the rule. *"The Inclusive Gateway is activated if"* the conditions for the rule applications are satisfied. Thus, the requirement *"At least one incoming Sequence Flow has at least one token"* is represented by condition $E_1 \neq \varnothing$. The second requirement *"For every directed path formed by Sequence Flow that (i)... (ii)... (iii)... There is also a directed path formed by Sequence Flow that (iv)... (v)... (vi)"* is represented by the condition $\forall p_1 \in \Pi.\exists\, p_2 \in \Pi_{p_1}$, where Π is the set of paths satisfying (i), (ii) and (iii), while the sets Π_p, one for each path p in Π, contain paths satisfying (iv), (v) and (vi). Formally, they are defined as $\Pi = \{p \in \mathcal{P}(\mathsf{e}) \,|\, \sigma(\mathsf{first}(p)) > 0 \wedge \mathsf{last}(p) \in E_2\}$ and $\Pi_p = \{p' \in \mathcal{P}(\mathsf{e}) \,|\, \mathsf{first}(p') = \mathsf{first}(p) \wedge \mathsf{last}(p') \in E_1\}$. In particular, a path p in Π is such that: "*(i) starts with a Sequence Flow f of the diagram that has a token*" ($\sigma(\mathsf{first}(p)) > 0$), "*(ii) ends with an incoming Sequence Flow of the inclusive gateway that has no token*" ($\mathsf{last}(p) \in E_2$), and "*(iii) does not visit the Inclusive Gateway*" (ensured by definition of $\mathcal{P}$). Instead, given a path p in Π, a path p' in Π_p is such that: "*(iv) starts with f*" ($\mathsf{first}(p') = \mathsf{first}(p)$, as f is the first edge of p), "*(v) ends with an incoming Sequence Flow of the inclusive gateway that has a token*" ($\mathsf{last}(p') \in E_1$), and "*(vi) does not visit the Inclusive Gateway*" (ensured again by definition of $\mathcal{P}$).

Running Example 9. *The initial configuration of the process in Figure 5.2 is $\langle P, \sigma_0\rangle$ where:*

$$
\begin{aligned}
P \;=\; & \mathsf{start}(\mathsf{e}_1) \parallel \mathsf{orSplit}(\mathsf{e}_1, \{\mathsf{e}_2, \mathsf{e}_3\}) \parallel \mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_6\}, \mathsf{e}_4) \\
& \parallel \mathsf{xorSplit}(\mathsf{e}_4, \{\mathsf{e}_5, \mathsf{e}_6\}) \parallel \mathsf{orJoin}(\{\mathsf{e}_3, \mathsf{e}_5, \mathsf{e}_8\}, \mathsf{e}_7) \parallel \mathsf{xorSplit}(\mathsf{e}_7, \{\mathsf{e}_8, \mathsf{e}_9\}) \\
& \parallel \mathsf{end}(\mathsf{e}_9, \mathsf{e}_{10})
\end{aligned}
$$

and $\sigma_0(\mathsf{e}_i) = 0 \,\forall\, i = 1, \ldots, 10$.



(a) Initial configuration.

(b) Step 1.

(c) Step 2.

(d) Step 3.

(e) Step 4.

(f) Step 5.

(g) Step 6.

Figure 5.7: Running Example.

Given the initial configuration by applying rule G-Start the execution of the process starts by marking with a token the edge e_1 *(Figure 5.7a). Rule G-OrSplit can be then applied; it moves the token from* e_1 *to one (or more) outgoing edges of the OR-Split, say both (Figure 5.7b). On the top branch rule G-XorJoin can be applied producing a token on edge* e_4 *(Figure 5.7c).*

Supposing that this token comes back once into the cycle, before reaching edge e_5 *(Figure 5.7d, 5.7e, 5.7f), all premises of rule G-OrJoin are satisfied:* $E_1 = \{\mathsf{e}_3, \mathsf{e}_5\} \neq \emptyset$*, and the condition based in the universal quantification trivially holds as* $\Pi = \emptyset$*, since all paths with a token at the beginning and no token at the end, e.g.* $(\mathsf{e}_3, \mathsf{e}_7, \mathsf{e}_8)$*, do visit the OR-Join, thus violating the requirement (iii). Therefore, the rule can be applied, the token in* e_3 *and* e_5 *are synchronised and a new token is produced on* e_7 *(Figure 5.7g). From there, the execution simply proceeds according to the semantics of the XOR gateway.* □

5.4 Local Semantics of BPMN OR-Join

The OR-Join semantics presented in the previous section perfectly fits with the informal definition given in the BPMN 2.0 specification. However, the evaluation of the OR-Join gateway activation (formalised by the premises of rule *G-OrJoin*) requires a global view of the process marking. From a practical perspective, this may complicate the implementation of the process control flow, also considering that the semantics of all other BPMN constructs is *local*, i.e. it relies only on the information about the marking of incoming and outgoing edges. Therefore, this section proposes an alternative, yet equivalent, semantics of BPMN, including the OR-Join construct, that is local, devised to more efficiently determine the OR-Join activation.

For the local semantics, only *safe* models [100] are considered. Recall, safeness requires a model to not activate an edge more than once at the same time. This assumption is not too restrictive, since safeness is recognised as one of the most important correctness criteria for business process models [29]. The lack of this property, in fact, may cause issues concerning process execution, related e.g. to the proper termination of processes or to erroneous synchronisations among concurrent control flows [20] (see Section 4.5).

5.4.1 Syntax

To enable local treatment of the BPMN semantics, roughly speaking the global state information of a process is spread over the edges of its structure, resulting on a *Marked Process*. Formally, the syntax of marked processes, denoted by M, is defined in Figure 5.8. The only difference between the syntax of a marked process and a process structure is that in the former an edge is also characterised by a *type* T, indicating if it is part of a cycle (c) or not (nc), and by a *status* Σ, denoting whether a token is marking the edge (*live* status denoted by l), or it may still arrive (*wait* status denoted by w), or will not arrive (*dead* status denoted by d). Notably, as explained

$$
\begin{array}{lcl}
M & ::= & \mathsf{start}(\mathsf{e}.T.\Sigma) \mid \mathsf{end}(\mathsf{e}.T.\Sigma, \mathsf{e}'.T.\Sigma) \mid \mathsf{andSplit}(\mathsf{e}.T.\Sigma, E) \mid \\
 & & \mathsf{andJoin}(\mathsf{e}.T.\Sigma, E) \mid \mathsf{xorSplit}(\mathsf{e}.T.\Sigma, E) \mid \mathsf{xorJoin}(\mathsf{e}.T.\Sigma, E) \mid \\
 & & \mathsf{orSplit}(\mathsf{e}.T.\Sigma, E) \mid \mathsf{orJoin}(\mathsf{e}.T.\Sigma, E) \mid M_1 \,\|\, M_2 \\
T & ::= & \mathsf{c} \mid \mathsf{nc} \qquad \Sigma ::= \mathsf{l} \mid \mathsf{w} \mid \mathsf{d}
\end{array}
$$

Figure 5.8: BPMN Syntax of Marked Processes.

in Section 5.2.2, edge types are statically determined in the pre-processing. With abuse of notation, the edge set notation E extends to marked edges.

5.4.2 Semantics

The operational semantics does not need to consider any more configurations with a state, but it is directly given in terms of marked processes. Formally, the operational semantics is defined by means of a labelled transition relation $M \xrightarrow{\ell}_L M'$, meaning that "the marked process M performs a transition labelled by ℓ and becomes M' in doing so". Labels ℓ are used to propagate the effect of marking updates, resulting from the evolution of a subterm of the process, to the other subterms. They are triples of the form $(\mathsf{w} : E_1, \mathsf{d} : E_2, \mathsf{l} : E_3)$, indicating the edges whose status must be set to w, d and l, respectively. For the sake of simplicity, within labels, sets E_j contain just edge names (without type and status). Moreover, to improve readability, a field of the triple is omitted when the associated edge set is empty, and brackets { and } are removed in case of singleton; for example, the label $(\mathsf{w} : \varnothing, \mathsf{d} : \varnothing, \mathsf{l} : \{\mathsf{e}\})$ is written $\mathsf{l} : \mathsf{e}$.

To define the labelled transition relation, a few auxiliary functions are necessary. First, $setLive(E)$, $setDead(E)$ and $setWait(E)$ are exploited to change the status of gateway edges to l, d and w, respectively. They are inductively defined on the structure of E as follows.

- $setLive(\varnothing) = \varnothing$ and $setLive(\{\mathsf{e}.T.\Sigma\} \cup E) = \{\mathsf{e}.T.\mathsf{l}\} \cup setLive(E)$
- $setDead(\varnothing) = \varnothing$ and $setDead(\{\mathsf{e}.T.\Sigma\} \cup E) = \{\mathsf{e}.T.\mathsf{d}\} \cup setDead(E)$
- $setWait(\varnothing) = \varnothing$ and $setWait(\{\mathsf{e}.T.\Sigma\} \cup E) = \{\mathsf{e}.T.\mathsf{w}\} \cup setWait(E)$

Similarly, to check if the edges in E have live (resp. dead, resp. wait) status, the boolean function $isLive(E)$ (resp. $isDead(E)$, resp. $isWait(E)$) is exploited. It is inductively defined as following on the structure of E.

- $isLive(\varnothing) = true$
- $isLive(\{\mathsf{e}.T.\Sigma\} \cup E) = \begin{cases} true & \text{if } \Sigma = \mathsf{l} \wedge isLive(E) \\ false & \text{otherwise.} \end{cases}$
- $isDead(\varnothing) = true$

- $isDead(\{\mathsf{e}.T.\Sigma\} \cup E) = \begin{cases} true & \text{if } \Sigma = \mathsf{d} \wedge isDead(E) \\ false & \text{otherwise.} \end{cases}$
- $isWait(\varnothing) = true$
- $isWait(\{\mathsf{e}.T.\Sigma\} \cup E) = \begin{cases} true & \text{if } \Sigma = \mathsf{w} \wedge isWait(E) \\ false & \text{otherwise.} \end{cases}$

Finally, to distinguish the type of edges in E the boolean functions $isC(E)$ and $isNC(E)$ are used. They are inductively defined on the structure of E as following.

- $isC(\varnothing) = true$
- $isC(\{\mathsf{e}.T.\Sigma\} \cup E) = \begin{cases} true & \text{if } T = \mathsf{c} \wedge isC(E) \\ false & \text{otherwise.} \end{cases}$
- $isNC(\varnothing) = true$
- $isNC(\{\mathsf{e}.T.\Sigma\} \cup E) = \begin{cases} true & \text{if } T = \mathsf{nc} \wedge isNC(E) \\ false & \text{otherwise.} \end{cases}$

Now, rules defining the evolution of the different types of tokens need to be introduced. In particular, Figure 5.9 presents rules related the live token, while in Figure 5.10 rules related the dead token. Notably, the initial status of a marked process M is identified by means of the boolean predicate $isInit(M)$, which holds when all edges of M have status w.

Definition 15 (Initial status of marked processes). *Let M be a marked process, then $isInit(M)$ is inductively defined on the structure of its argument process as follows:*

- $isInit(\mathsf{start}(\mathsf{e}.T.\Sigma))$ *if* $isWait(\mathsf{e}.T.\Sigma)$
- $isInit(\mathsf{andSplit}(\mathsf{e}.T.\Sigma, E))$ *if* $isWait(\{\mathsf{e}.T.\Sigma\} \cup E)$
- $isInit(\mathsf{andJoin}(\mathsf{e}.T.\Sigma, E))$ *if* $isWait(\{\mathsf{e}.T.\Sigma\} \cup E)$
- $isInit(\mathsf{xorSplit}(\mathsf{e}.T.\Sigma, E))$ *if* $isWait(\{\mathsf{e}.T.\Sigma\} \cup E)$
- $isInit(\mathsf{xorJoin}(\mathsf{e}.T.\Sigma, E))$ *if* $isWait(\{\mathsf{e}.T.\Sigma\} \cup E)$
- $isInit(\mathsf{orSplit}(\mathsf{e}.T.\Sigma, E))$ *if* $isWait(\{\mathsf{e}.T.\Sigma\} \cup E)$
- $isInit(\mathsf{orJoin}(\mathsf{e}.T.\Sigma, E))$ *if* $isWait(\{\mathsf{e}.T.\Sigma\} \cup E)$
- $isInit(\mathsf{end}(\mathsf{e}.T.\Sigma, \mathsf{e}'.T.\Sigma))$ *if* $isWait(\{\mathsf{e}.T.\Sigma\} \cup \{\mathsf{e}'.T.\Sigma\})$
- $isInit(M_1 \parallel M_2)$ *if* $isInit(M_1)$ *and* $isInit(M_2)$

$\mathsf{start}(\mathsf{e.nc.w}) \xrightarrow{\mathsf{l:e}}_L \mathsf{start}(\mathsf{e.nc.l})$		$(L\text{-}Start\text{-}NC)$
$\mathsf{end}(\mathsf{e.nc.l}, \mathsf{e'.nc.w}) \xrightarrow{(\mathsf{d:e,l:e'})}_L \mathsf{end}(\mathsf{e.nc.d}, \mathsf{e'.nc.l})$		$(L\text{-}End\text{-}NC)$
$\mathsf{andSplit}(\mathsf{e.nc.l}, E) \xrightarrow{(\mathsf{d:e,l:}E)}_L \mathsf{andSplit}(\mathsf{e.nc.d}, setLive(E))$		$(L\text{-}AndSplit\text{-}NC)$
$\mathsf{andSplit}(\mathsf{e.c.l}, E) \xrightarrow{(\mathsf{w:e,l:}E)}_L \mathsf{andSplit}(\mathsf{e.c.w}, setLive(E))$		$(L\text{-}AndSplit\text{-}C)$
$\mathsf{andJoin}(E, \mathsf{e.nc.}\Sigma) \xrightarrow{(\mathsf{d:}E\mathsf{,l:e})}_L$ $\mathsf{andJoin}(setDead(E), \mathsf{e.nc.l})$	$isLive(E)$	$(L\text{-}AndJoin\text{-}NC)$
$\mathsf{andJoin}(E_1 \sqcup E_2, \mathsf{e.c.}\Sigma) \xrightarrow{(\mathsf{w:}E_1\mathsf{,d:}E_2\mathsf{,l:e})}_L$ $\mathsf{andJoin}(setWait(E_1) \sqcup setDead(E_2), \mathsf{e.c.l})$	$isLive(E_1 \sqcup E_2)$, $isC(E_1)$, $isNC(E_2)$	$(L\text{-}AndJoin\text{-}C)$
$\mathsf{xorSplit}(\mathsf{e.nc.l}, \{\mathsf{e'.nc.}\Sigma\} \sqcup E) \xrightarrow{(\mathsf{d:}\{\mathsf{e}\}\sqcup E\mathsf{,l:e'})}_L$ $\mathsf{xorSplit}(\mathsf{e.nc.d}, \{\mathsf{e'.nc.l}\} \sqcup setDead(E))$		$(L\text{-}XorSplit\text{-}NC)$
$\mathsf{xorSplit}(\mathsf{e.c.l}, \{\mathsf{e'.c.}\Sigma\} \sqcup E_1 \sqcup E_2) \xrightarrow{(\mathsf{w:e,l:e'})}_L$ $\mathsf{xorSplit}(\mathsf{e.c.w}, \{\mathsf{e'.c.l}\} \sqcup E_1 \sqcup E_2)$	$isC(E_1)$, $isNC(E_2)$	$(L_1\text{-}XorSplit\text{-}C)$
$\mathsf{xorSplit}(\mathsf{e.c.l}, \{\mathsf{e'.nc.}\Sigma\} \sqcup E_1 \sqcup E_2) \xrightarrow{(\mathsf{d:}\{\mathsf{e}\}\sqcup E_1 \sqcup E_2\mathsf{,l:e'})}_L$ $\mathsf{xorSplit}(\mathsf{e.c.d}, \{\mathsf{e'.nc.l}\} \sqcup setDead(E_1) \sqcup setDead(E_2))$	$isC(E_1)$, $isNC(E_2)$	$(L_2\text{-}XorSplit\text{-}C)$
$\mathsf{xorJoin}(\{\mathsf{e'.nc.l}\} \sqcup E, \mathsf{e.nc.}\Sigma) \xrightarrow{(\mathsf{d:}\{\mathsf{e'}\}\sqcup E\mathsf{,l:e})}_L$ $\mathsf{xorJoin}(\{\mathsf{e'.nc.d}\} \sqcup setDead(E), \mathsf{e.nc.l})$		$(L\text{-}XorJoin\text{-}NC)$
$\mathsf{xorJoin}(\{\mathsf{e'.c.l}\} \sqcup E_1 \sqcup E_2, \mathsf{e.c.}\Sigma) \xrightarrow{(\mathsf{w:e'\mathsf{,d:}}E_2\mathsf{,l:e})}_L$ $\mathsf{xorJoin}(\{\mathsf{e'.c.w}\} \sqcup E_1 \sqcup setDead(E_2), \mathsf{e.c.l})$	$isC(E_1)$, $isNC(E_2)$	$(L_1\text{-}XorJoin\text{-}C)$
$\mathsf{xorJoin}(\{\mathsf{e'.nc.l}\} \sqcup E_1 \sqcup E_2, \mathsf{e.c.}\Sigma) \xrightarrow{(\mathsf{d:}\{\mathsf{e'}\}\sqcup E_2\mathsf{,l:e})}_L$ $\mathsf{xorJoin}(\{\mathsf{e'.nc.d}\} \sqcup E_1 \sqcup setDead(E_2), \mathsf{e.c.l})$	$isC(E_1)$, $isNC(E_2)$	$(L_2\text{-}XorJoin\text{-}C)$
$\mathsf{orSplit}(\mathsf{e.nc.l}, E_1 \sqcup E_2) \xrightarrow{(\mathsf{d:}\{\mathsf{e}\}\sqcup E_2\mathsf{,l:}E_1)}_L$ $\mathsf{orSplit}(\mathsf{e.nc.d}, setLive(E_1) \sqcup setDead(E_2))$	$E_1 \neq \varnothing$	$(L\text{-}OrSplit\text{-}NC)$
$\mathsf{orSplit}(\mathsf{e.c.l}, E_1 \sqcup E_2) \xrightarrow{(\mathsf{w:e,l:}E_1)}_L$ $\mathsf{orSplit}(\mathsf{e.c.w}, setLive(E_1) \sqcup E_2)$	$E_1 \neq \varnothing$, $isC(E_1)$, $isNC(E_2)$	$(L_1\text{-}OrSplit\text{-}C)$
$\mathsf{orSplit}(\mathsf{e.c.l}, E_1 \sqcup E_2) \xrightarrow{(\mathsf{d:}\{\mathsf{e}\}\sqcup E_2\mathsf{,l:}E_1)}_L$ $\mathsf{orSplit}(\mathsf{e.c.d}, setLive(E_1) \sqcup setDead(E_2))$	$E_1 \neq \varnothing$, $isNC(E_1)$, $isC(E_2)$	$(L_2\text{-}OrSplit\text{-}C)$
$\mathsf{orJoin}(E_1 \sqcup E_2, \mathsf{e.nc.}\Sigma) \xrightarrow{(\mathsf{d:}E_1\sqcup E_2\mathsf{,l:e})}_L$ $\mathsf{orJoin}(setDead(E_1 \sqcup E_2), \mathsf{e.nc.l})$	$E_1 \neq \varnothing$, $isLive(E_1)$, $isDead(E_2)$	$(L\text{-}OrJoin\text{-}NC)$
$\mathsf{orJoin}(E_1 \sqcup E_2, \mathsf{e.c.}\Sigma) \xrightarrow{(\mathsf{w:}E_1\mathsf{,d:}E_2\mathsf{,l:e})}_L$ $\mathsf{orJoin}(setWait(E_1) \sqcup setDead(E_2), \mathsf{e.c.l})$	$E_1 \neq \varnothing$, $isLive(E_1)$, $isDead(E_2)$, $isC(E_1)$, $isNC(E_2)$, $noDep(\mathsf{e})$	$(L_1\text{-}OrJoin\text{-}C)$
$\mathsf{orJoin}(E_1 \sqcup E_2, \mathsf{e.c.}\Sigma) \xrightarrow{(\mathsf{w:}E_2\mathsf{,d:}E_1\mathsf{,l:e})}_L$ $\mathsf{orJoin}(setDead(E_1) \sqcup setWait(E_2), \mathsf{e.c.l})$	$E_1 \neq \varnothing$, $isLive(E_1)$, $isDead(E_2)$, $isNC(E_1)$, $isC(E_2)$, $noDep(\mathsf{e})$	$(L_2\text{-}OrJoin\text{-}C)$

Figure 5.9: BPMN Local Semantics - Evolution of Live Tokens.

Considering live tokens, the semantics behaves as following. Start and end events have edges with non-cyclic type, as according to the BPMN standard a start event cannot have an incoming edge and the end event cannot have outgoing edges (the completing edge is a spurious edge that cannot be connected to other elements). In particular, rule *L-Start-NC* annotates the edge e outgoing from the start event with l when the process is in the initial status (in fact, the edge has a w status before the transition), the corresponding label l : e is produced. Rule *L-End-NC* is activated when the incoming edge has an l status; it produces a d status on the incoming edge and a l status on the completing edge. Then, there are two rules for the AND-Split according to the type of incoming and outgoing edges. In particular, rule *L-AndSplit-NC* is applied when there is a live status on the non-cyclic incoming edge of an AND-Split (and as a consequence all the outgoing edges are of type nc). As result of its application, it annotates with d the incoming edge and with l the outgoing edges. Instead, rule *L-AndSplit-C* is applied when the cyclic incoming edge of an AND-Split gateway has the live status. In this case, of course, some of the outgoing edges are included in a cycle. As result of its application, the rule annotates with w the incoming edge (to allow it to be set to live again) and with l the outgoing edges. Similarly, also the AND-Join gateway has two cases: rule *L-AndJoin-NC* produces a live status on the outgoing edge, that is of non-cyclic type, and a dead status on the non-cyclic incoming edges of the AND-Join, only when all the incoming edges are labelled with l ($isLive(E)$). Rule *L-AndJoin-C* produces a live status on the cyclic outgoing edge, a dead status on the incoming edges of type nc ($isNC(E_2)$) and a wait status on the other incoming edges, only when all the incoming edges are labelled with l ($isLive(E)$). Moreover, this rule makes use of the operator $\sqcup$ to distinguish non-cyclic edges from cyclic ones. For the XOR-Split gateway there are three different cases according to the type of outgoing edges. In detail, rule *L-XorSplit-NC*, applied when there is a live status on the non-cyclic incoming edge, annotates with l one of the outgoing edges and with d the others. Instead, when at least one of the outgoing edges of the XOR-Split is of type nc there are two rules, according to the type of the chosen outgoing edge. In particular, rule L_1-*XorSplit-C*, applied when a live status is available in the incoming edge, annotates with l one of the outgoing edges; if that edge is of type c all the other edges are annotated with w, otherwise rule L_2-*XorSplit-C* is applied, producing a wait status only on that edges of type c and a dead status on the others. Similarly, there are three rules also for the XOR-Join gateway: they distinguish the case in which the outgoing edge is of non-cyclic type, from the case in which it is of type c. More in detail, rule *L-XorJoin-C* is activated every time there is a live status on one of the incoming edges, of non-cyclic type, changing all incoming edges status with d and the outgoing edge status with

l. Rule L_1-*XorJoin-C* is activated every time there is a live status on one of the incoming edges of type c changing the outgoing edge status with l and the status of the incoming edges of type c (resp. nc) with w (resp. d). Rule L_2-*XorJoin-C* is activated when there is a live status on an incoming edge of type nc and produces a live status on the outgoing edge, a dead status on all the incoming edges of type nc, while does not change the wait status of the incoming edges of cyclic type. Considering the OR-Split gateway there are three rules $L = L$-*OrSplit-NC*, $L_1 = L_1$-*OrSplit-C* and $L_2 = L_2$-*OrSplit-C*, that differ for the type of the incoming/outgoing edges. In particular, rule L is activated when there is live status on the non-cyclic incoming edge (and as a consequence all the outgoing edges are of type nc); its application changes the status of the incoming edge in d and annotates with l at least one of the outgoing edges and all the other are setted to d. Rules L_1 and L_2 differ for the type of the chosen outgoing edges: both of them are activated when there is a live status on the incoming edge of type c, but L_1 changes this status to w since the chosen outgoing edges belonging to E_1 are of type c. Instead L_2 changes the status of the incoming edge to d and annotates with l the chosen outgoing edges, that in this case are of type nc, and with d all the others. Similarly, considering the OR-Join there are three rules $L' = L$-*OrJoin-NC*, $L'_1 = L_1$-*OrJoin-C*, $L'_2 = L_2$-*OrJoin-C*. In particular, L' is applied when the outgoing edge is of non-cyclic type, while L'_1 and L'_2 when it is of type c. In these latter cases, the boolean predicate $noDep(\mathsf{e})$, defined in Section 5.2.2, is used to ensure that in case of vicious circles ($noDep(\mathsf{e}) = false$) the rules cannot be applied, thus enforcing a deadlocked behaviour as prescribed by the BPMN 2.0 standard. In detail, rule L' is activated when some of the incoming edges ($isLive(E_1)$) have a live status, while the others have a dead status ($isDead(E_2)$). Its application annotates with l the outgoing edge and with dead all the incoming edges. Rule L'_1, instead, is applied when the incoming edges with live status ($isLive(E_1)$) are of cyclic type ($isC(E_1)$) and there is no dependence with other OR-Joins. Its application annotates with l the outgoing edge, with dead the incoming edges of type nc and with wait the incoming edges of type c. Rule L'_2 acts in a similar way.

The rules described so far are not enough for properly expressing the OR-Join behaviour. Other rules are indeed needed to propagate the dead status information. These rules are shown in Figure 5.10. They are applied when the incoming edges of a gateway are annotated with d, and simply propagate this information to the outgoing edges. More in detail, regarding the split gateways, the dead status is propagated to all outgoing edges when the incoming edge is annotated with d. Considering the join gateways instead, the dead status is propagated only when all incoming edges are annotated with d. In particular, concerning the AND-Join gateway, this means that in case there is a deadlock upstream of an incoming edge, the outgoing edge

will remain in the wait status forever.

$$\mathsf{andSplit}(\mathsf{e}.T.\mathsf{d}, E) \xrightarrow{\mathsf{d}:E}_L \mathsf{andSplit}(\mathsf{e}.T.\mathsf{d}, setDead(E)) \qquad (D\text{-}AndSplit)$$

$$\mathsf{andJoin}(\mathsf{e}.T.\Sigma, E) \xrightarrow{\mathsf{d}:\mathsf{e}}_L \mathsf{andJoin}(\mathsf{e}.T.\mathsf{d}, E) \quad isDead(E) \quad (D\text{-}AndJoin)$$

$$\mathsf{xorSplit}(\mathsf{e}.T.\mathsf{d}, E) \xrightarrow{\mathsf{d}:E}_L \mathsf{xorSplit}(\mathsf{e}.T.\mathsf{d}, setDead(E)) \qquad (D\text{-}XorSplit)$$

$$\mathsf{xorJoin}(\mathsf{e}.T.\Sigma, E) \xrightarrow{\mathsf{d}:\mathsf{e}}_L \mathsf{xorJoin}(\mathsf{e}.T.\mathsf{d}, E) \quad isDead(E) \quad (D\text{-}XorJoin)$$

$$\mathsf{orSplit}(\mathsf{e}.T.\mathsf{d}, E) \xrightarrow{\mathsf{d}:E}_L \mathsf{orSplit}(\mathsf{e}.T.\mathsf{d}, setDead(E)) \qquad (D\text{-}OrSplit)$$

$$\mathsf{orJoin}(\mathsf{e}.T.\Sigma, E) \xrightarrow{\mathsf{d}:\mathsf{e}}_L \mathsf{orJoin}(\mathsf{e}.T.\mathsf{d}, E) \quad isDead(E) \quad (D\text{-}OrJoin)$$

Figure 5.10: BPMN Local Semantics - Dead Status Propagation.

Finally, *M-StatusUpd₁* and *M-StatusUpd₂* rules allow the interleaving of the process element evolution.

$$\frac{M_1 \xrightarrow{\ell}_L M_1'}{M_1 \parallel M_2 \xrightarrow{\ell}_L M_1' \parallel M_2 \star \ell} \quad (M\text{-}StatusUpd_1)$$

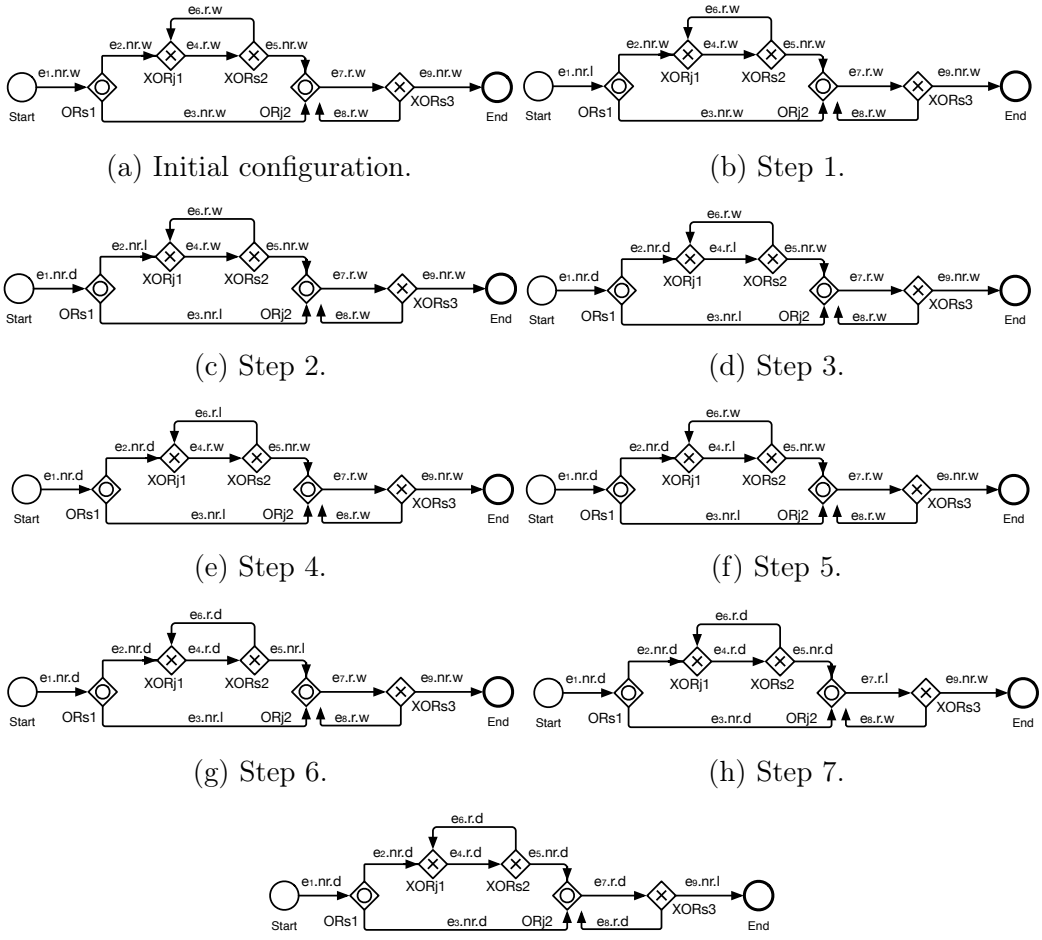
$$\frac{M_2 \xrightarrow{\ell}_L M_2'}{M_1 \parallel M_2 \xrightarrow{\ell}_L M_1 \star \ell \parallel M_2'} \quad (M\text{-}StatusUpd_2)$$

They rely on the status updating function $M \star \ell$, which returns a process obtained from M by updating the status of its edges according to the labelled sets they belong to in ℓ. Formally, this function is inductively defined on the structure of process M and, with abuse of notation, extends to terms of the form $\mathsf{e}.T.\Sigma$ and E as follows:

- $\varnothing \star \ell \;=\; \varnothing$
- $\mathsf{e}.T.\Sigma \star (\mathsf{w} : E_1, \mathsf{d} : E_2, \mathsf{l} : E_3) \;=\; \begin{cases} \mathsf{e}.T.\mathsf{w} & \text{if } \mathsf{e} \in E_1 \\ \mathsf{e}.T.\mathsf{d} & \text{if } \mathsf{e} \in E_2 \\ \mathsf{e}.T.\mathsf{l} & \text{if } \mathsf{e} \in E_3 \\ \mathsf{e}.T.\Sigma & \text{otherwise.} \end{cases}$
- $(\{\mathsf{e}.T.\Sigma\} \sqcup E) \star \ell = \{\mathsf{e}.T.\Sigma \star \ell\} \sqcup (E \star \ell)$

Then, in each base case of the inductive definition of the status updating function, the function is simply applied to the edges $\mathsf{e}.T.\Sigma$ of process nodes. The definition cases are reported below:

$$
\begin{array}{lcl}
\mathsf{start}(\mathsf{e}.T.\Sigma) \star \ell & = & \mathsf{start}(\mathsf{e}.T.\Sigma \star \ell) \\
\mathsf{end}(\mathsf{e}.T.\Sigma, \mathsf{e}'.T.\Sigma) \star \ell & = & \mathsf{end}(\mathsf{e}.T.\Sigma \star \ell, \mathsf{e}'.T.\Sigma \star \ell) \\
\mathsf{andSplit}(\mathsf{e}.T.\Sigma, E) \star \ell & = & \mathsf{andSplit}(\mathsf{e}.T.\Sigma \star \ell, E \star \ell) \\
\mathsf{andJoin}(\mathsf{e}.T.\Sigma, E) \star \ell & = & \mathsf{andJoin}(\mathsf{e}.T.\Sigma \star \ell, E \star \ell) \\
\mathsf{xorSplit}(\mathsf{e}.T.\Sigma, E) \star \ell & = & \mathsf{xorSplit}(\mathsf{e}.T.\Sigma \star \ell, E \star \ell) \\
\mathsf{xorJoin}(\mathsf{e}.T.\Sigma, E) \star \ell & = & \mathsf{xorJoin}(\mathsf{e}.T.\Sigma \star \ell, E \star \ell) \\
\mathsf{orSplit}(\mathsf{e}.T.\Sigma, E) \star \ell & = & \mathsf{orSplit}(\mathsf{e}.T.\Sigma \star \ell, E \star \ell) \\
\mathsf{orJoin}(\mathsf{e}.T.\Sigma, E) \star \ell & = & \mathsf{orJoin}(\mathsf{e}.T.\Sigma \star \ell, E \star \ell) \\
(M_1 \parallel M_2) \star \ell & = & M_1 \star \ell \parallel M_2 \star \ell
\end{array}
$$



(a) Initial configuration. (b) Step 1.

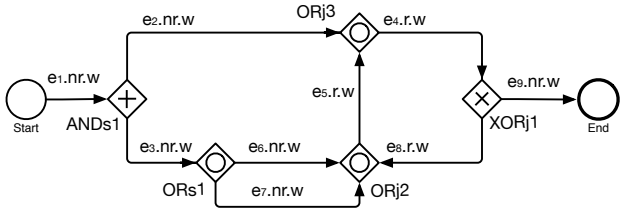
(c) Step 2. (d) Step 3.

(e) Step 4. (f) Step 5.

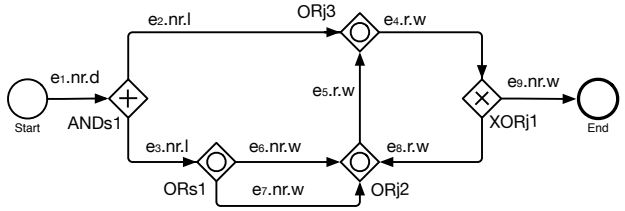
(g) Step 6. (h) Step 7.

(i) Step 8.

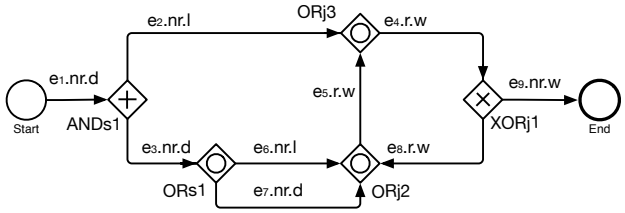
Figure 5.11: Running Example.



(a) Initial marking.



(b) Step 1.



(c) Step 2.

Figure 5.12: Example of Vicious Circle.

Now it is shown how the local semantics works on the running example considering a possible evolution from the initial state.

Running Example 10. *Figure 5.11a shows the initial configuration of the running example introduced in Section 5.2, that is when all edges are annotated with* w*. Then, the application of rule L-Start-NR produces a live status for edge* e_1 *(Figure 5.11b). Supposing now that the OR-Split gateway* $ORs1$ *produces a live status for* e_3 *and* e_2 *(Figure 5.11c), then the live status of* e_2 *triggers the XOR-Join gateway* $XORj1$ *that produces a live status for* e_4 *(Figure 5.11d). Supposing now, that the first time, the XOR-Split gateway* $XORs2$ *annotates with* l *edge* e_6 *by means of rule* L_1*-XorSplit-C (Figure 5.11e), then this re-activates the* $XORj1$ *(Figure 5.11f) that produces again a live status for* e_4*. Supposing now that rule* L_2*-XorSplit-C is applied, annotating with* l *edge* e_5 *and with dead* e_4 *and* e_6 *(Figure 5.11g), then the OR-Join gateway* $ORj2$ *has two incoming edges annotated with* l *and one with* w*. Since the edge marked with* w *is of cyclic type and the considered OR-Join does not depend from other OR-Joins, it can proceed by producing a live status for* e_7 *(Figure 5.11h). After the synchronisation, if the XOR-Split* $XORs3$ *evaluates as true the condition on* e_9*, producing a live status for that*

edge e_9 *and a dead status for the other outgoing edge* e_8 *(Figure 5.11i).* □

To clarify the semantics in presence of vicious circles, another example is reported below.

Example 8. *Figure 5.12a shows the initial configuration of a process model exhibiting a vicious circle, i.e. two OR-Joins mutually dependent on each others. Thanks to the behaviour of the AND-Split a live status is produced on its outgoing edges* e_2, e_3 *(Figure 5.12b). Then in Step 2 the OR-Split* $ORs1$ *produces a live status on edge* e_6 *and a dead status on edges* e_7*; therefore both* $ORj2$ *and* $ORj3$ *has an incoming edge* e_2 *(resp.* e_6*) annotated with* l*, while the other incoming edges are in a dead (*e_7*) or wait status (*$\mathsf{e}_5, \mathsf{e}_8$*) (Figure 5.12c). In particular, the ones annotated with* w *are of cyclic type* c*. This would allow the OR-Join to execute if there would not be dependencies with other OR-Joins. However, since for instance* $noDep(\mathsf{e}_5) = false$*, the OR-Join* $ORj2$ *is not activated, thus the process is blocked.* □

The section concludes by proving the correspondence between the provided global and local semantics. In order to do that it is first necessary to illustrate the correspondence between the syntax used in the global formalisation and that used in the local version. The local notation is achieved by applying σ to the structure P, that is by distributing the token information included in σ on the edges of P. Notably, considering only safe processes it follows $0 \leqslant \sigma(\mathsf{e}) \leqslant 1$. Moreover, auxiliary notations t and nl are used to denote an *undefined type*, which can be either c or nc, and a *not live* status, which can be either w or d. Formally:

Definition 16 (**Syntax correspondence**). *Let* $\langle P, \sigma \rangle$ *be a process configuration, then* $P \cdot \sigma$ *is inductively defined on the structure of* P *as follows:*

- $\mathsf{start}(\mathsf{e}) \cdot \sigma = \mathsf{start}(\mathsf{e.t.}(\mathsf{e} \cdot \sigma))$
- $\mathsf{end}(\mathsf{e}, \mathsf{e}') \cdot \sigma = \mathsf{end}(\mathsf{e.t.}(\mathsf{e} \cdot \sigma), \mathsf{e}'.\mathsf{t.}(\mathsf{e}' \cdot \sigma))$
- $\mathsf{andSplit}(\mathsf{e}, E) \cdot \sigma = \mathsf{andSplit}(\mathsf{e.t.}(\mathsf{e} \cdot \sigma), (E \cdot \sigma))$
- $\mathsf{andJoin}(E, \mathsf{e}) \cdot \sigma = \mathsf{andJoin}((E \cdot \sigma), \mathsf{e.t.}(\mathsf{e} \cdot \sigma))$
- $\mathsf{xorSplit}(\mathsf{e}, E) \cdot \sigma = \mathsf{orJoin}(\mathsf{e.t.}(\mathsf{e} \cdot \sigma), (E \cdot \sigma))$
- $\mathsf{xorJoin}(E, \mathsf{e}) \cdot \sigma = \mathsf{orJoin}((E \cdot \sigma), \mathsf{e.t.}(\mathsf{e} \cdot \sigma))$
- $\mathsf{orSplit}(\mathsf{e}, E) \cdot \sigma = \mathsf{orJoin}(\mathsf{e.t.}(\mathsf{e} \cdot \sigma), (E \cdot \sigma))$
- $\mathsf{orJoin}(E, \mathsf{e}) \cdot \sigma = \mathsf{orJoin}((E \cdot \sigma), \mathsf{e.t.}(\mathsf{e} \cdot \sigma))$
- $(P_1 \parallel P_2) \cdot \sigma = P_1 \cdot \sigma \parallel P_2 \cdot \sigma$

where

$$\mathsf{e}\cdot\sigma = \begin{cases} \mathsf{l} & \text{if } \sigma(\mathsf{e}) = 1; \\ \mathsf{nl} & \text{otherwise.} \end{cases} \qquad \varnothing\cdot\sigma = \varnothing \qquad (\{\mathsf{e}\}\cup E)\cdot\sigma = \{\mathsf{e.t}.(\mathsf{e}\cdot\sigma)\}\cup(E\cdot\sigma)$$

and for each $\mathsf{e.t}$ *we have that* $\mathsf{t} = \begin{cases} \mathsf{c} & \text{if } \mathsf{e}\in\mathbb{C}; \\ \mathsf{nc} & \text{otherwise.} \end{cases}$

with $\mathbb{C}$ *be the set of edges included in a cycle in* $\langle P,\sigma\rangle$ *(see Section 5.2.2).*

According to the above definition, a term $P\cdot\sigma$ represents a class of marked processes, i.e. all those processes with the same marking for what concerns the live status, but possibly different markings for the other two status (information that indeed is not considered at all in the global semantics). Therefore, to state that marked processes belong to a given class the relation $\equiv$ is used. Its meaning is as follows: $M\equiv P\cdot\sigma$ means that M is syntactical equivalent to $P\cdot\sigma$, up to an instantiation of nl occurrences in $P\cdot\sigma$.

Finally, the results rely on the notion of *reachable* configuration/processes. In fact, the considered syntaxes are too liberal, as they allow terms that cannot be obtained (by means of transitions) from a process in its initial state.

Definition 17 (**Reachable process configuration/marked process**). *A process configuration* $\langle P,\sigma\rangle$ *(resp. marked process* M*) is* reachable *if there exists* $\langle P,\sigma'\rangle$ *(resp. process* M'*) such that* $\sigma'=\sigma_0$ *(resp.* $isInit(M')$*) and* $\langle P,\sigma'\rangle \rightarrow_G{}^*\sigma$ *(resp.* $M' \xrightarrow{\ell}_L{}^* M$*).*

Now, the formal results state that each step of the global semantics corresponds to one or more steps of the local semantics (Theorem 10) and vice versa (Theorem 11). Their proofs are given by induction on the derivation of the transitions and are fully reported in Appendix C.2. Theorem 10 relies on the following Lemma concerning the evolution of processes up to syntax correspondence.

Lemma 5. *Given a marked process* M*, if* $M\equiv P\cdot\sigma \xrightarrow{\ell}_L M'\equiv P\cdot\sigma'$ *then, given* $M''\equiv P'\cdot\sigma$ *for some* P*, we have that* $M''\star\ell = P'\cdot\sigma'$*.*

Proof (sketch). By structural induction of P. □

Theorem 10. *Let* $\langle P,\sigma\rangle$ *be a reachable process configuration, if* $\langle P,\sigma\rangle\rightarrow_G\sigma'$ *then for all reachable marked process* M *such that* $M\equiv P\cdot\sigma$*,* $M\xrightarrow{\ell}_L{}^+ M'$ *and* $M'\equiv P\cdot\sigma'$*.*

Proof (sketch). By induction on the derivation of $\langle P, \sigma \rangle \rightarrow_G \sigma'$ □

Theorem 11. *Let M be a reachable marked process, with $M \equiv P \cdot \sigma$, if $M \stackrel{s}{\Rightarrow}_L M'$, where $s = \ell_1, \ldots, \ell_n$ is such that $\forall i = 1, \ldots, n-1$, ℓ_i is of the form $(\mathsf{d} : E_2)$ and ℓ_n is of the form $\ell_n = (\mathsf{w} : E_1, \mathsf{d} : E_2, \mathsf{l} : E_3)$ with $E_3 \neq \varnothing$ then $\langle P, \sigma \rangle \rightarrow_G \sigma'$ and $M' \equiv P \cdot \sigma'$.*

Proof (sketch). By induction on the derivation of $M \stackrel{s}{\Rightarrow}_L M'$ □

5.5 Global vs. Local Semantics

Proved that the two semantics are equivalent under the safeness assumption according to Theorems 10 and 11, this section compares the two semantics by conducting some experiments.

Given a fragment of a process model ending in an OR-Join, that will be called OR-Join model fragment (Figure 5.13), it can be observed that the number of applications of the OR-Join rule depends on the number and on the length of the branches joined by the gateway in this fragment.

In particular, Figure 5.13 generalised the dimension of an OR-Join model fragment, showing a block composed by $k+1$ incoming branches of different lengths, each of which has a token. More precisely, there is a path p_0 that has a token on the ending edge ($\sigma(\mathsf{last}(p_0)) = 1$), while the others p_i, $i = 1...k$, have a token on the starting edge ($\sigma(\mathsf{first}(p_i)) = 1$).

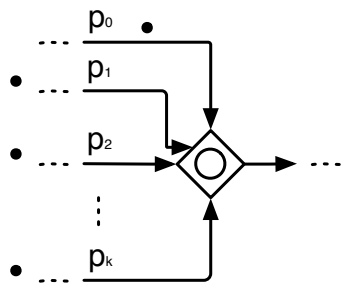


Figure 5.13: OR-Join Model Fragment.

Considering the two semantics applied to a model with the structure subsumed in the proposed fragment, due to the presence of a token in the incoming edge of the OR-Join, it can be observed a different number of invocations of *L-OrJoin-NC* and *G-OrJoin* rules. In particular, the global rule will be applied every time the other tokens perform a step in the k branches, i.e. in total $\sum_{i=1}^{k} max\_length(p_i)$ where the function max_length provides the length of the longest path. Instead the local rule will be applied only when all the other tokens reach the gateway, hence in total k times (one time for branch).

This may result in a scenario where the number of applications of the global rule may grows infinitely. It is the case of the proposed running example, where an OR-Join could wait for a token, stacked inside a loop in one of the path incoming to the OR-Join.

Considering the running example, in Figure 5.7, the number of incoming branches of the OR-join is equal to two. Even if it influences the number of applications of both global and local semantic rules it is considered as a constant. Indeed, adding incoming branches to the OR-join, the number of applications of both the global and the local semantic rules increases alike.

The experiments consider model executions in which both the OR-Join branches are active. These executions differ for the number of cycles (i is the cycle number). Each executed model is denoted by the parameter i that increases form 0 to 100 in step of 10.

Experimental results are showed in Table 5.1. Each row provides the number of loop repetitions that increases form 0 to 100 in step of 10 (i column), the number of applications of the global semantics (global calls column), the total time spent by the global semantics (global time column), the number of application of the local semantics (local calls column), the total time spent by the local semantics (local time column). The first row represents the base case where the model instance does not contain cycles. In this case, the global rule *G-OrJoin* is applied four times, while the local rule *L-OrJoin-NC* is invoked just twice, like in all the other cases. In this case, the execution times result quite similar. Differently, considering the other rows in Table 5.1, the local semantics keeps approximatively the same execution time, hence a constant progress, while the global semantics time increases linearly. With i equal to 10, the global semantics lasts two times and half as long as the local one. While increasing the number of cycles until 100 this gap raises to more than 12 times. This increase is also shown up clearly on the chart in Figure 5.14, focussing only on time.

i	Global		Local	
	calls	**time**	**calls**	**time**
0	4	0,372	2	0,425
10	22	1,18	2	0,477
20	42	1,91	2	0,532
30	62	3,21	2	0,503
40	82	3,45	2	0,503
50	102	3,54	2	0,466
60	122	3,58	2	0,555
70	142	3,92	2	0,519
80	162	4,16	2	0,479
90	182	4,86	2	0,503
100	202	5,60	2	0,462

Table 5.1: Experiment Results.

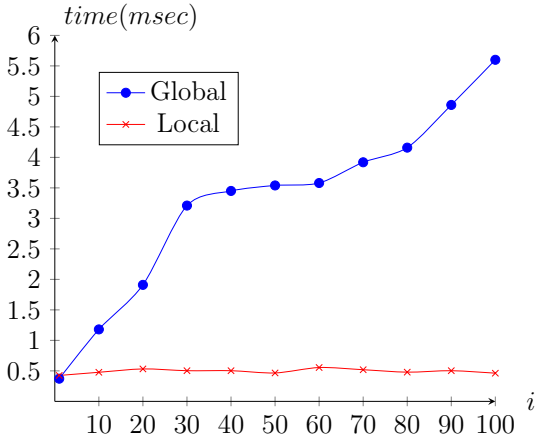


Figure 5.14: Experiment Trend.

Notably, looking at the execution of a single model, the difference in terms of time is not very significant, due to the fact that each additional invocation

of the global semantics lasts less than a millisecond. Anyway, shifting the attention on business processes execution engines in which several executions of the same model as well as different models are running at the same time, the gap between the semantics becomes more and more revealing.

From the experiment, it turns out that having a global semantics means that a process is busy waiting for a condition to be satisfied. This can be avoided by putting a process to sleep and waking it up when the appropriate execution state is reached, that is by means of a local semantics.

This semantics fosters a compositional, hence more scalable, approach for enacting processes with OR-Joins. Indeed, the global semantics has been introduced as the formal reference, while the local one to be used for implementations.

5.6 Properties of BPMN Processes

This section provides 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. Notably, having demonstrated the correspondence between the global and the local semantics (Theorems 10 and 11), the considered properties are defined only with respect to the global formalisation.

5.6.1 Well-Structured BPMN Processes

Advantages of structuring BPMN models have been discussed for years. Having structured models can be a benefit in a framework considering BPMN models including the OR-Join gateway. In fact, in well-structured models one can have a priori knowledge of the number of tokens the OR-Join has to wait for by looking at the number of tokens produced by the corresponding OR-Split gateway.

The formal characterisation of well-structured BPMN processes relies on the usual functions $in(P)$ and $out(P)$, which determine the incoming and outgoing sequence edges of a process element P (their full definition is relegated to Appendix B). Since the definition of well-structuredness extends the one given in Section 4.1.1 only the new cases are reported.

Definition 18 (Well-structured processes)**.** *A process P is* well-structured *(written $isWS(P)$) if P has the following form:*

$$\mathsf{start(e)} \quad \| \; P' \; \| \quad \mathsf{end(e'', e''')}$$

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

1. $isWSCore(\mathsf{orSplit}(\mathsf{e}, E) \parallel \mathsf{orJoin}(E', \mathsf{e}''))where E = E'$

2. $$\frac{\forall j \in [1..n] \;\; isWSCore(P_j), \;\; in(P_j) \subseteq E, \;\; out(P_j) \subseteq E'}{isWSCore(\mathsf{orSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{orJoin}(E', \mathsf{e}''))}$$

According to Definition 18, case 1 means that a process fragment starting with an OR-Split and closing with an OR-Join is well-structured. Case 2 means that a composite process starting with an OR-Split and closing with an OR-Join is well-structured core if each edge of the split is connected to a given edge of the join by means of well-structured core processes.

Running Example 11. *Considering the running example in Figure 5.1 and according to the above definitions, the process is not well-structured, due to the presence of the XOR split gateway not combined with a XOR join gateway.* □

5.6.2 Safe BPMN Processes

Among the considered properties, safeness plays an important role in the business process domain. This is particularly evident when considering processes including the OR join gateway, whose behaviour is quite anomalous in presence of an unsafe model. In fact, the local semantics is defined under the safeness assumption.

Before providing a formal characterisation of safe BPMN processes, the following definition determining the safeness of a process in a given state needs to be introduced.

Definition 19 (Current state safe process)**.** *A process configuration* $\langle P, \sigma \rangle$ *is current state safe (*cs-safe*) if and only if* $\forall \mathsf{e} \in edgesEl(P) \,.\, \sigma(\mathsf{e}) \leqslant 1$.

The definition of safe processes requires that cs-safeness is preserved along the computations. $\rightarrow_G^*$ is used to denote the reflexive and transitive closure of $\rightarrow_G$.

Definition 20 (Safe processes)**.** *A process* P *is* safe *if and only if, given* σ_0*, such that the process configuration is in the initial state* $\langle P, \sigma_0 \rangle$*, for all* σ' *such that* $\langle P, \sigma_0 \rangle \rightarrow_G^* \sigma'$ *we have that* $\langle P, \sigma' \rangle$ *is cs-safe.*

Running Example 12. *Considering again the running example, it is easily to see that the process is safe, since there is no process fragment capable of producing more than one token.* □

5.6.3 Sound BPMN Processes

The tricky informal semantics of the OR-Join gateway can lead to different interpretation of its behaviour, affecting also the classification of models including the OR-Join with respect to some correctness properties, such as soundness. Thus, providing a formal OR-Join semantics and soundness definition is crucial to allow a correct classification of BPMN models including the OR-Join gateway.

The definition of sound processes relies on the notion of current state sound process. It, in turns, relies on the definition of function $marked(\sigma,end(P))$, which refers to the set of completing edges ($end(P)$) with at least one token.

Definition 21 (Current state sound process)**.** *A process configuration $\langle P, \sigma \rangle$ is current state sound (*cs-sound*) if and only if $\forall \mathsf{e} \in marked(\sigma,end(P))$. $\sigma(\mathsf{e}) = 1$ and $\forall \mathsf{e} \in edges(P) \backslash end(P)$. $\sigma(\mathsf{e}) = 0$.*

Definition 22 (Sound process)**.** *A process P is* sound *if and only if, given σ_0, such that the process configuration is in the initial state $\langle P, \sigma_0 \rangle$, for all σ' such that $\langle P, \sigma_0 \rangle \rightarrow_G^* \sigma'$ we have that there exists σ'' such that $\langle P, \sigma' \rangle \rightarrow_G^* \sigma''$, and $\langle P, \sigma'' \rangle$ is cs-sound.*

Running Example 13. *The process in the running example in Figure 5.1 is sound. In fact, the OR-Join is not blocked, thus the process can reach a marking where only the end event is marked with one token.* □

5.7 Classification Results

The extension of the formal framework with the introduction of the OR-Join gateway does not change the classification results obtained for the core BPMN elements (Section 4.3). Nevertheless, a formalisation of this element, faithfully compliant with the standard, is crucial also to provide a precise classification of BPMN models. In fact, the tricky behaviour of the OR-Join gateway, without a precise semantics, can lead to different interpretations, affecting also the violation of general correctness criteria, such as soundness. It may happen that the same model can either belong to the class of sound models or to the class of unsound model according to the different interpretations of the OR-Join behaviour. In particular, these differences regard the treatment of mutually dependent OR-Joins (the so-called "vicious circles") and of deadlock upstream an OR-Join. As already proved, from a faithful translation of the standard, it results that in these cases the OR-Joins are blocked, resulting in unsound processes.

5.8 Comparison with other Approaches

The particular behaviour defined for the OR-Join in BPMN 2.0 has triggered a stream of research on providing a formal semantics for this construct. Indeed, a precise semantics enables to overcome misunderstandings and allows to reason on model properties by means of formal methods techniques. This section reports the most relevant work in this direction. Specifically, first available OR-Join formalisations are discussed and then an investigation on correctness properties of models including the OR-Join is reported.

5.8.1 OR-Join Formalisations

Most of the previous attempts to formalise the semantics of the OR-Join [107, 31, 93, 18, 37] are based on earlier versions of the BPMN standard, which provide different semantics for the OR-Join and do not fit with the current 2.0 standard. Moreover, also when the current version of the standard is considered, different interpretations of the OR-Join behaviour, not always faithful to the specification, have been given. In particular, these differences regard the treatment of mutually dependent OR-Joins (the so-called 'vicious circles') and of deadlock upstream an OR-Join. In fact, from a faithful translation of the standard, it results that mutually dependent OR-Joins are blocked, since an OR-Join is not able to recognise a deadlock on a path leading to it, thus it will wait forever.

This section discusses the most significant related works. In Table 5.2 a comparison among the state-of-the-art approaches resolving the OR-Join issue is summarised. It compares them with respect to: *(i)* the BPMN version used for process specification, *(ii)* the language used to give the OR-Join semantics, *(iii)* the approach used to give the semantics (local or globally specified), *(iv)* the type of marking (the distinction is between the base case with a simple token, from those approaches where the token is enriched with more structural information such as boolean information, colours or token set). When available the table summarises also: *(v)* the safeness or soundness assumption proposed by each approach, *(vi)* how each approach threats the vicious circle (the distinction is between the cases in which the mutually dependent OR-Joins wait for each other reporting the term blocked, from the case in which the OR-Joins can fire using the term unlocked), and *(vi)* how it manages the deadlock upstream an OR-Join (the distinction is between the cases in which the deadlock is detected and resolved using the term unlocked from the case the deadlock is not resolved by the semantics using the term blocked) The symbol (-) is used when no information regarding the considered aspect is provided.

Dumas et al. [31] base their work on BPMN 1.0 and on the definition of the Synchronisation Merge pattern to which the specification refers to.

Paper	Year	BPMN Version	Language	Semantics	Marking	Restrictions	Vicious Circles	Deadlock Upstream
[31]	2007	BPMN 1.0	Algorithm	Local	Base	-	Unlocked	Unlocked
[93]	2009	BPMN 1.0	ASM	Global	Token Sets	-	-	-
[107]	2010	BPMN 1.0	WF Graph	Global	Base	Safeness	Blocked	-
[108]	2010	BPMN 2.0 draft	WF Graph	Global	Base	Safeness	Blocked	-
[18]	2011	BPMN 2.0 beta 1	Direct	Global	Base	-	-	-
[37]	2016	BPMN 2.0 draft	WF Graph	Local	3 Colours	Soundness	-	-
[53]	2015	BPMN 2.0	WF Graph	Local	Base	Soundness	Unlocked	-
This thesis	2018	BPMN 2.0	Direct	Global	Base	-	Blocked	Blocked
This thesis	2018	BPMN 2.0	Direct	Local	Status	Safeness	Blocked	Blocked

Table 5.2: Review on the OR-Join Semantics.

They provide a local semantics given throughout an algorithm with simple token representing the state of the execution. The approach does not impose any restriction and it is able to unlock mutually dependent OR-Joins and to detect deadlocks upstream. Thalheim et al. [93] refer to the BPMN 1.0 specification and make use of Abstract State Machines to introduce the global semantics of the OR-Join. Adopting a token-based view of workflow semantics, to threat the OR-Join they introduce sets of tokens, which are viewed as a coherent group when a join fires. Völzer [107] proposes a non-local semantics for the OR-Join in the BPMN 1.0 specification (2006) using workflow graphs. The semantics is given via simple tokens working under the safeness restriction. In case of vicious circles he argues that the intended meaning is not clear and hence they should be sort out by static analysis. Nevertheless, there are cases where mutually dependent OR-Joins create a deadlock in his semantics. This approach is then improved in [108], which quotes the 2010 version of the specification. Under the same assumptions of the previous work, the new approach presents an implemenatation that requires only constant time to decide whether an OR-Join is enabled in a given state. However, the cost grows linearly with the number of present OR-Joins. Carbone et al. [18] refer to BPMN 2.0 - Beta 1, providing a global semantics directly in terms of a subset of BPMN. The approach does not imposes any restriction and in particular, concerning the vicious circle, they argue that, since informally BPMN specification does not include the resolution strategy and their work is a faithful translation, they do not consider it. Fahland and Völzer [37] study dynamic versions of the classical flexibility constructs skip and block and define a formal semantics for them. This study gives rise to a simple and fully local semantics for inclusive gateways, based on the 2010 BPMN version. It makes use of coloured tokens and requires a model to be sound. They do not consider the vicious circle; moreover there can be scenarios where the OR-Join semantics could lead to deadlocks. The above approaches rely on past versions of the BPMN standard, which provide different semantics for the OR-Join with respect to the current 2.0 version. Thus, they cannot be applied as they are to the standard BPMN 2.0. Moreover, concerning vicious circles and deadlock upstream considered by some of those works, this thesis checkes how they are dealt with the current specification and, to be completely faithful with it, it simply applies the same solution. Indeed, in the current description of the OR-Join semantics in Figure 5.3, it does not seem to be any ambiguity about these two issues. The OR-Join is able to detect neither a vicious circle nor a deadlock upstream, thus in both cases its execution is blocked forever. An example is shown in the situation depicted in Figure 5.15, where ANDj1 is deadlocked because no token will never arrive in e_5 to synchronise with the one in e_4.

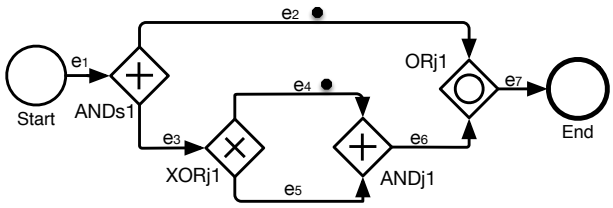


Figure 5.15: OR-Join with Deadlock Upstream.

The OR-Join activation condition "At least one incoming Sequence Flow has at least one token" is satisfied, as there is a token in e_2. Regarding the other condition, the sequence edge f can only be e_4; now, for the path formed by e_4 and e_6 there is no other path starting with e_4 and ending with a marked edge incoming to the OR-Join (like e_2). Thus, the OR-Join is not activated and will never be, as the waited token in e_4 will never move. The situation is similar in case of vicious circles.

Only few works rely on the current version of BPMN. However, they are not fully compliant with the OMG standard.

Prinz et al. [53] propose a formalisation of the OR-Join semantics referred to the current version of the standard. However, they limit the work on *sound* workflow graphs, which identify a quite restricted class of BPMN processes [114]. In this regard, they propose a soundness definition in models including the OR-Join gateway. However, the proposed semantics does not fit with the standard as, for instance, it avoids vicious circles by determining which OR-Join in a circle has to wait and which one has to proceed. Thus, also soundness verification is affected.

Summing up, differently from previous attempts to formalise the OR-Join semantics the provided semantics results from a faithful and direct translation of the current version of the standard. This permits to resolve OR-Join issues without leaving room for ambiguity and to define correctness properties fitting with the standard.

As just seen, in formalising the OR-Join semantics different interpretations have been given. The same has happened also for what concerns its implementation. In fact, studying the implementation of the OR-Join used by some popular BPMN management tools it can be observed that most of them relax, simplify or even avoid it. In particular, this work checked: Camunda [16], Flowable [1], Stadust [4], jBPM [84], Process Maker [2], Sydle [5] and Signavio [3]. These BPMN tools provide their own

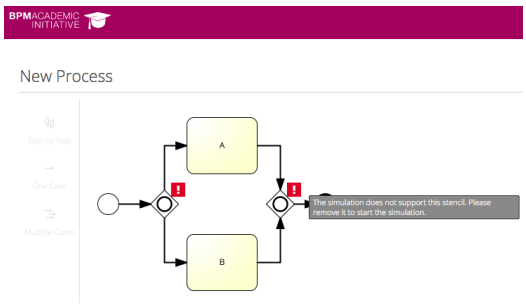


Figure 5.16: Signavio Simulation Error.

interpretation of the BPMN standard, typically relaxing the OR-Join semantics. More specifically, Camunda and Flowable take advantage from the Activiti OR-Join implementation that in some cases keeps blocked a waiting token differently from what prescribed in the specification. A similar behaviour is followed by Stadust. Differently, jBPM, Process Maker and Sydle handle only relaxed process structures in which the OR-Join merges branches created by an OR-Split, and then enforce a simplified behaviour. Last but not least, Signavio, and in particular its simulation feature, does not support the OR gateway neither in split nor in join as shown in Figure 5.16.

5.8.2 Reasoning on Correctness Properties

The common way for reasoning about correctness of BPMN models is to define a formal semantics of the modelling language. Afterwards, formal reasoning methods can be applied on the obtained formal model. Usually BPMN formalisations rely on mappings into well-founded formalism, such as Petri nets [29, 50, 83, 8]. None of the available mappings considers the OR-Join in processes with arbitrary topology. Kunze and Weske [50], propose a mapping of the OR-Join in well-structured processes. However, concerning well-structuredness, none of the proposed definitions [44, 35, 98] formally defines well-structured processes considering the case in which OR-Split and OR-Join gateways are properly nested.

Among the available formalisations including the OR-Join gateway, most of them works under the safeness or soundness assumption [107, 53]. Specifically, Völzer [107] considers unsoundness as a modelling error, since the lack of synchronisation (namely unsafeness) can cause undesired race conditions for the OR-join enabledness in their semantics. Prinz et al. [53] describe a general definition for soundness of processes guaranteeing the absence of deadlocks and no lack of synchronisation for each possible OR-join semantics. In both cases, the OR-Join semantics is given in operation of the property definitions. Differently, here first a semantics compliant with the BPMN standard is provided, and then, based on that semantics, correctness properties are properly defined. A relevant work in this direction is the one by Wynn et al. [114] that propose new verification techniques to determine the correctness of business processes with cancellation and OR-joins. They present these techniques in the context of the workflow language YAWL but they argue that the results also apply to other languages, such as BPMN. However, also the proposed approach relies on the use of mapping from a business model to a reset net, thus inheriting the constraints of the target language.

Chapter 6

Sub-Processes

Modelling inter-organisational informative systems generally requires to provide descriptions of the system at different levels of abstraction. On the one hand, the model designer can better manage the complexity he/she has to handle going from abstract descriptions of the system towards more detailed ones, so to focus from time to time on specific aspects of the system or parts of it. On the other hand, going from a detailed description to an abstract one permits to have models less crowded and more understandable to the interested readers, thus keeping the information flow in line with general comprehension capabilities [19].

The formalisation provided for the BPMN core elements is able to capture both the process and the collaboration level. In addition, within a process, the sub-process element can be used to represent a compound activity that can be expanded or collapsed at designer convenience, in order to regulate the abstraction level of the model. However, the combined usage of messages and sub-processes in a collaboration model can lead to intricate and cumbersome behaviours, which may hide undesired situations not easily identifiable by the designer. In the business process modelling domain such undesirable behaviours are typically related to the violation of general correctness criteria, such as safeness and soundness. Notably, a sub-process is not syntactic sugar that can be removed via a sort of macro expansion, i.e. via simply substituting the sub-process element by its internal content. Moreover, due to its behaviour the sub-process can impact on the classification of the models, both at the process and collaboration level.

Thus, this chapter extends the semantics already given for the core subset of BPMN elements by considering sub-processes. To illustrate the sub-process effects on the classification first a modified version of the running example is presented (Section 6.1), then the formal framework is extended taking into account distinctive characteristics introduced by sub-process elements (Section 6.2), finally business process properties are defined on this

framework (Section 6.3) and it is shown the sub-process impact on the BPMN classification (Section 6.4).

Highlights. Distinctive aspects of this chapter are:

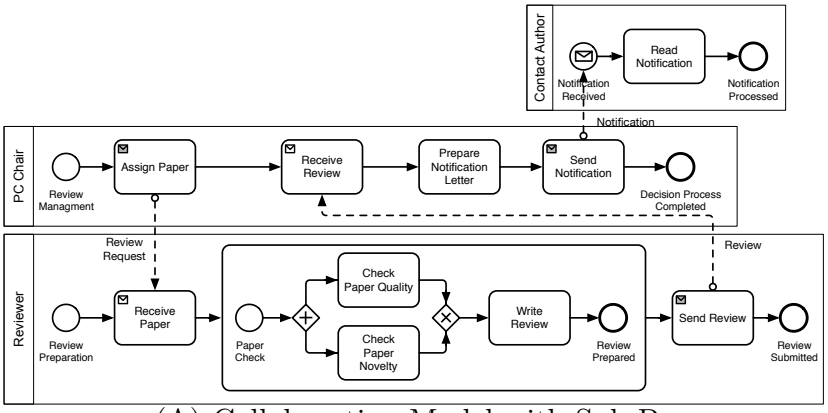
- it defines a formal framework for rigorously characterising the semantics of BPMN models including sub-processes, thus adding a new level of abstraction;
- on top of the defined semantics it formally characterised the considered correctness properties, underling the impact of sub-processes on the classification results;
- the resulting classification represents an advance in classifying BPMN models, as it is derived from a faithful formalisation of the sub-process behaviour, as reported in the current standard specification.

6.1 Running Example

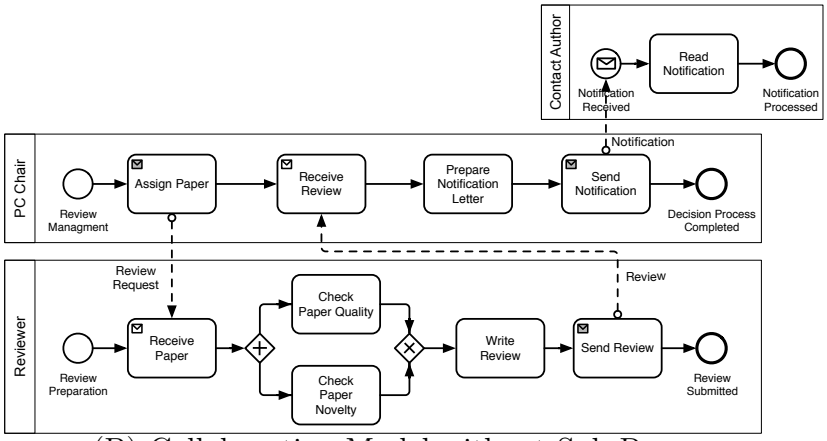
According to the BPMN standard a sub-process element is *"an activity that encapsulates a process that is in turn modelled by activities, gateways, events, and sequence flows"* [68, Sec. 13.2.4, pp.430]. Once activated, a sub-process instance remains in the execution state as long as the encapsulated process is running. Then, only when none of its internal activity is active the sub-process completes [68, pp. 431], and as a result the control is passed to the sequence edge outgoing from the sub-process element. A good effect of the standard definition refers to the possibility to check the quality of a model at different levels of abstraction when it includes sub-processes, which, when collapsed, can be considered as a "normal" task consuming an incoming token when it starts, and then it will produces an outgoing token when it ends.

To illustrate the behaviour of the sub-process elements Figure 6.1 reports two exemplificative models concerning the reviewing process for a scientific conference. As in Section 3.1 simplification are in place. In particular, the two collaboration diagrams combine the activities of a single **PC Chair**, a single **Reviewer** and the **Contact Author**. In the process specified in Figure 6.1 (A) the reviewing phase, including the checks on the paper and the writing of the review, is embedded into a sub-process, while Figure 6.1 (B) shows the same process where the sub-process has been flattened into the main process.

Notably, differently from what could be expected, according to the definition provided by the standard the two processes lead to different process completion. In particular, in the process in Figure 6.1 (A) only one token is propagated after the execution of the sub-process, thus only one token



(A) Collaboration Model with Sub-Process



(B) Collaboration Model without Sub-Process

Figure 6.1: Paper Reviewing Collaboration Model

reaches the end event of the including process. A different result can be observed for the flattened process model in Figure 6.1 (B). In this case the two tokens generated by the AND-Split gateway will both reach the end event.

6.2 Formal Framework

This section extends the formalisation already defined in Section 3.2 by including the sub-process element, which was overlooked in the previous semantics definition. Specifically, it presents the syntax and operational semantics.

6.2.1 Syntax

Form the syntactic point of view, syntax of processes in Section 3.2.1 needs to be extended with a specific construct representing the sub-process element:

$$P \ ::= \ \ldots \mid \mathsf{subProc}(\mathsf{e}_i, P, \mathsf{e}_o)$$

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

Here in the following the auxiliary functions already defined in Section 3.2.1 are used. In particular, since this section considers process including nested sub-processes, to refer to the enabling edges of the start events of the current level it resorts to function $start(P)$. Recall, it is assumed 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. Similarly functions $end(P)$ and $end(C)$ are defined on the structure of processes and collaborations in order to refer to end events in the current layer.

Running Example 14. *The BPMN model in Figure 6.1(A) is expressed in the syntax as the following collaboration structure (at an unspecified step of execution):*

$$\mathsf{pool}(\mathsf{PC\ Chair}, P_{pc}) \parallel \mathsf{pool}(\mathsf{Reviewer}, P_r) \parallel \mathsf{pool}(\mathsf{Contact\ Author}, P_{ca})$$

with P_r, P_{pc} and P_{ca} expressed as follows, where for simplicity the edges are identified in progressive order e_i (with $i = 0 \ldots 25$):

$$\begin{aligned}
P_r &= \mathsf{start}(\mathsf{e}_0, \mathsf{e}_1) \parallel \mathsf{taskRcv}(\mathsf{e}_1, \mathsf{Review\ Request}, \mathsf{e}_2) \parallel \mathsf{subProc}(\mathsf{e}_2, P, \mathsf{e}_{13}) \parallel \\
&\quad \mathsf{taskSnd}(\mathsf{e}_{12}, \mathsf{Review}, \mathsf{e}_{13}) \parallel \mathsf{end}(\mathsf{e}_{13}, \mathsf{e}_{14}) \\
P &= \mathsf{start}(\mathsf{e}_3, \mathsf{e}_4) \parallel \mathsf{andSplit}(\mathsf{e}_4, \{\mathsf{e}_5, \mathsf{e}_6\}) \parallel \mathsf{task}(\mathsf{e}_5, \mathsf{e}_7) \parallel \\
&\quad \mathsf{task}(\mathsf{e}_6, \mathsf{e}_8) \parallel \mathsf{andJoin}(\{\mathsf{e}_7, \mathsf{e}_8\}, \mathsf{e}_9) \parallel \mathsf{task}(\mathsf{e}_9, \mathsf{e}_{10}) \parallel \mathsf{end}(\mathsf{e}_{10}, \mathsf{e}_{11}) \\
P_{pc} &= \mathsf{start}(\mathsf{e}_{15}, \mathsf{e}_{16}) \parallel \mathsf{taskSnd}(\mathsf{e}_{16}, \mathsf{Review\ Request}, \mathsf{e}_{17}) \parallel \\
&\quad \mathsf{taskRcv}(\mathsf{e}_{17}, \mathsf{Review}, \mathsf{e}_{18}) \parallel \\
&\quad \mathsf{task}(\mathsf{e}_{18}, \mathsf{e}_{19}) \parallel \mathsf{taskSnd}(\mathsf{e}_{19}, \mathsf{Notification}, \mathsf{e}_{20}) \parallel \mathsf{end}(\mathsf{e}_{20}, \mathsf{e}_{21}) \\
P_{ca} &= \mathsf{startRcv}(\mathsf{e}_{22}, \mathsf{Notification}, \mathsf{e}_{23}) \parallel \mathsf{task}(\mathsf{e}_{23}, \mathsf{e}_{24}) \parallel \mathsf{end}(\mathsf{e}_{24}, \mathsf{e}_{25})
\end{aligned}$$

Moreover, considering the functions defined on the structure it follows: $participant(\mathsf{pool}(\mathsf{PC\ Chair}, P_{pc}) \parallel \mathsf{pool}(\mathsf{Reviewer}, P_r) \parallel \mathsf{pool}(\mathsf{Contact\ Author}, P_{ca})) = \{P_{pc}, P_r, P_{ca}\}$, $start(P_r) = \{\mathsf{e}_0\}$, $start(P) = \{\mathsf{e}_3\}$ *and* $end(P) = \{\mathsf{e}_{11}\}$ $end(P_r) = \{\mathsf{e}_{14}\}$. *Finally,* $edges(P_r) = \{\mathsf{e}_0, ..., \mathsf{e}_{14}\}$ *and* $edges(P) = \{\mathsf{e}_3, ..., \mathsf{e}_{11}\}$, $edgesEl(P_r) = \{\mathsf{e}_1, ..., \mathsf{e}_{13}\}$, $edgesEl(P) = \{\mathsf{e}_4, ..., \mathsf{e}_{10}\}$ *The others are defined in a similar way.* □

The one-to-one correspondence between the syntax used here to represent a sub-process and the graphical notation of BPMN, that is exemplified by means of the running example in Figure 6.1(A), is also reported in detail in the Appendix A.3.

6.2.2 Semantics

The syntax presented so far permits to describe the mere structure of a collaboration and a process. To describe the semantics of collaboration models, the structural information needs to be enriched with a notion of execution state, defining the current marking of sequence and message edges. As previously, the operational semantics is directly defined on BPMN in terms of LTS, whose definition relies on an auxiliary LTS on the behaviour of processes.

Now, to formalise the behaviour of the sub-process element it is only sufficient to include in process and collaboration configurations information about the completion of the sub-process. To check it the boolean predicate $completed(P, \sigma)$ is exploited. 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" [68, pp. 426, 431]. Thus, the sub-process completion can be formalised as follows.

Definition 23. *Let P be a process included in the sub-process the predicate $completed(P, \sigma)$ holds if one of the following conditions is satisfied:*

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

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.

Rules for the sub-process element are reported in Figure 6.2. Rule *P-SubProcStart* is activated only when there is a token in the incoming edge of the sub-process, which is then moved to the enabling edge of the start event in the sub-process body. Then, the sub-process behaves according to the behaviour of the elements it contains according to the rule *P-SubProcEvolution*. When the sub-process completes rule *P-SubProcEnd* is activated. It removes all the tokens from the sequence edges of the sub-process body[1], and adds a token to the outgoing edge of the sub-process. Rule *P-SubProcKill* deals

[1]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.

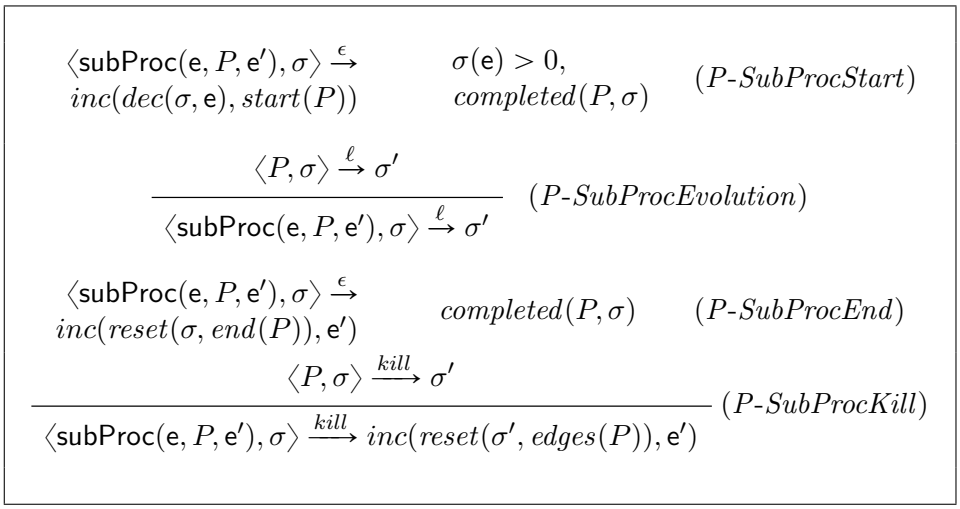


Figure 6.2: BPMN Sub-Process Semantics.

with a sub-process element observing a killing action in its behaviour due to an occurrence of a terminate event. This rule is activated only when there is a token in the incoming edge of termination event. Then, the sub-process stops its internal behaviours and passes the control to the upper layer, indeed the rule removes all the tokens in the sub-process and adds a token to the outgoing edge of the sub-process.

6.3 Properties of BPMN Collaborations

This section provides a rigorous characterisation of the key properties studied in this work, i.e. well-structuredness, safeness and soundness, in BPMN models including the sub-process element. These properties are characterised both at the process and collaboration level.

6.3.1 Well-Structured BPMN Collaborations

To be able to consider both structured and unstructured models this section provides a formal definition of well-structured BPMN models including the sub-process element. This definition naturally extends the one already given in Section 4.1.1, which relies on the auxiliary functions $in(P)$ and $out(P)$ (their definition is relegated to Appendix B) and on the definition of well-structured core, given via the boolean predicate $isWSCore(\cdot)$.

Recall, a process P is well-structured (written $isWS(P)$) if P is expressed as a (core) process included between any possible combination of different types of the considered start and end events. Moreover, it is required that the

core process satisfies the predicate $isWSCore(\cdot)$, which is defined as follows (only the new cases are here reported).

Definition 24 (Well-structured core processes)**.** *Predicate* $isWSCore(\cdot)$ *is inductively defined on the structure of its argument as follows:*

$$\frac{isWSCore(P_1'), in(P_1') = \{\mathsf{e}''\}, out(P_1') = \{\mathsf{e}'''\}}{}$$

$1(a).\ isWSCore(\mathsf{subProc}(\mathsf{e}, \mathsf{start}(\mathsf{e}', \mathsf{e}'') \parallel P_1' \parallel \mathsf{end}(\mathsf{e}''', \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}}))$
$1(b).\ isWSCore(\mathsf{subProc}(\mathsf{e}, \mathsf{start}(\mathsf{e}', \mathsf{e}'') \parallel P_1' \parallel \mathsf{terminate}(\mathsf{e}'''), \mathsf{e}^{\mathsf{iv}}))$
$1(c).\ isWSCore(\mathsf{subProc}(\mathsf{e}, \mathsf{start}(\mathsf{e}', \mathsf{e}'') \parallel P_1' \parallel \mathsf{endSnd}(\mathsf{e}''', \mathsf{m}, \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}}))$
$1(d).\ isWSCore(\mathsf{subProc}(\mathsf{e}, \mathsf{startRcv}(\mathsf{m}, \mathsf{e}'') \parallel P_1' \parallel \mathsf{end}(\mathsf{e}''', \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}}))$
$1(e).\ isWSCore(\mathsf{subProc}(\mathsf{e}, \mathsf{startRcv}(\mathsf{m}, \mathsf{e}'') \parallel P_1' \parallel \mathsf{terminate}(\mathsf{e}'''), \mathsf{e}^{\mathsf{iv}}))$
$1(f).\ isWSCore(\mathsf{subProc}(\mathsf{e}, \mathsf{startRcv}(\mathsf{m}, \mathsf{e}'') \parallel P_1' \parallel \mathsf{endSnd}(\mathsf{e}''', \mathsf{m}, \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}}))$

According to Definition 24 a sub-process is a well structured core if it includes a well-structured core process.

Well-structuredness naturally extends to collaborations, by requiring each process involved in a collaboration to be well-structured.

Running Example 15. *Considering the proposed running example in Figure 6.1 (A) and according to the above definitions, processes* P_{pc} *and* P_{ca} *are well-structured, while process* P_r *is not well-structured, due to the presence of the unstructured process fragment formed by two AND-Split and XOR-Join. Thus, the overall collaboration is not well-structured.* □

6.3.2 Safe BPMN Collaborations

The other relevant property considered in the thesis is safeness. In the new formal framework, including the sub-process element, the formal definition of safe BPMN processes and collaborations presented in Section 4.1.2 is still valid. Recall, it is based on definition determining the safeness of a process in a given state (cs-safeness), and requires that cs-safeness is preserved along the computations.

Running Example 16. *Considering again the running example depicted in Figure 6.1 (A), processes* P_{pc} *and* P_{ca} *are safe since there are not process fragments capable of producing more than one token. Process* P_r *instead is not safe. In fact, when the sub-process is started, it first runs in parallel tasks Check Paper Quality and Check Paper Novelty, then executes two times task Write Review. Thus, the whole process is not safe. Moreover, also the resulting collaboration is not safe.* □

6.3.3 Sound BPMN Collaborations

The use of sub-processes can affect the satisfaction of the soundness property, due to the BPMN notion of sub-process completion. In fact, affecting the completion of the whole process, the sub-process element impacts on the soundness property. However, the definition of sound processes and collaborations given in Section 4.1.3 are still valid in the new extended framework. Recall, it is based on definition determining the soundness of a process in a given state (cs-sound), and requires that this property is preserved along the computations. Also message-relaxed soundness naturally extends to the new framework.

Running Example 17. *Considering the running example in Figure 6.1 (A). It is easily to see that the collaboration in Figure 6.1 (B) is both unsound and message-relaxed unsound since in process P_{pc}, due to the fragment formed by the AND-Split and XOR-Join gateway, there will be a configuration with two tokens on the end event and a pending message. Now, considering the model in Figure 6.1 (A) obtained by Figure 6.1 (B) enclosing within a sub-process the part of the model generating multiple tokens, it can be observed a positive effect on the soundness of the model: the collaboration is both sound and message-relaxed sound.* □

6.4 Classification Results

The ability of the proposed formal framework to properly treat sub-processes without any restrictions enables to provide a more precise classification of the BPMN models as synthesised by the Euler diagrams in Figure 6.3(a) and Figure 6.3(b).

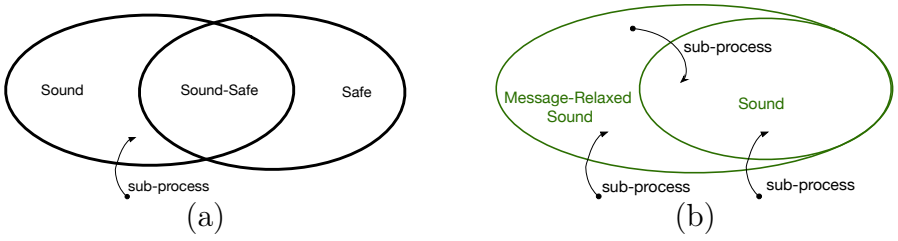


Figure 6.3: Reasoning at Process (a) and Collaboration (b) Level.

In particular, Figure 6.3(a) underlines reasoning that can be done at process level on soundness. Here it emerges how the sub-process element can affect model soundness, as properly using it, it may render sound a model that was unsound, as shown in Figure 6.1 (A) of the running example.

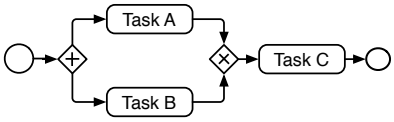


Figure 6.4: Unsound Process.

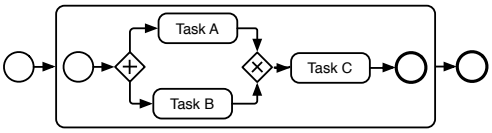


Figure 6.5: Sound Process with an Unsound Sub-Process.

Also the use of sub-processes can impact on the satisfaction of the soundness property. Figure 6.5 shows a simple process model where the unsound process in Figure 6.4 is included in the sub-process.

Notice, this reasoning is not affected by safeness and, in particular, it cannot be extended to collaborations. In fact, as discussed in Section 4.6, when composing two sound processes the resulting collaboration could be either sound or unsound.

Interesting situations also arise when focussing on the collaboration level, as highlighted in Figure 6.3(b). Worth to notice is the possibility to transform, with a small change, an unsound collaboration into a sound one.

Figure 6.6, Figure 6.7 and Figure 6.8 report a simple example showing the impact of sub-processes. Also in this case the models are rather similar, but according to the classification the result is completely different. The collaboration model in Figure 6.6 is both unsound and message-relaxed unsound since when ORG A there is a configuration with two tokens on the end event and a pending message. Now, considering the model obtained by Figure 6.6 introducing a sub-process the resulting collaboration in Figure 6.7 is unsound and message-relaxed sound, since the use of the sub-process mitigates the causes of message-relaxed unsoundness. In fact there will be only the issue of a pending message, since *Task C* sends two messages and only one will be consumed by *Task D*. Differently, Figure 6.8 shows that enclosing within a sub-process only the part of the model generating multiple tokens results in a positive effect on the soundness of the model. The collaboration is both sound and message-relaxed sound.

6.5 Relationships among Properties

This section studies how the introduction of the sub-process elements impacts on the relationships among the considered properties. In particular it investigates 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 C.3.

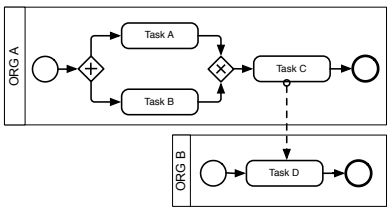


Figure 6.6: Example of Unsound and Message-Relaxed Unsound.

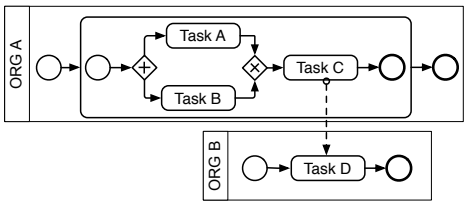


Figure 6.7: Example of Message-Relaxed Sound and Unsound Collaboration.

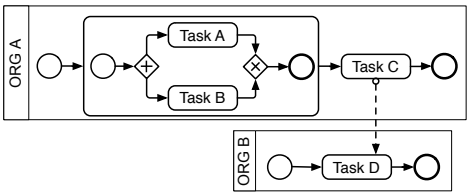


Figure 6.8: Example of Message-Relaxed Sound and Sound Collaboration.

6.5.1 Well-structuredness vs. Safeness in BPMN

Considering well-structuredness and safeness Section 4.4.1 proved that all well-structured models are safe (Theorem 1), and that the reverse does not hold. This results is still valid when introducing the sub-process element. To show that, first it is proved that a process in the initial state is cs-safe (Lemma 1). Then, it is shown that cs-safeness is preserved by the evolution of well-structured core process elements (Lemma 6) 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). This last notion, in its own turn, needs the definition of initial state for a core process element (see Appendix B). The proof follows the one already provided for the core set of BPMN elements with the exception of including the sub-process element. The new relevant cases regard the following Lemma and are shown in Appendix C.3.

Lemma 6. *Let* $isWSCore(P)$*, and let* $\langle P, \sigma \rangle$ *be a core reachable and cs-safe process configuration, if* $\langle P, \sigma \rangle \xrightarrow{\ell} \sigma'$ *then* $\langle P, \sigma' \rangle$ *is cs-safe.*

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

6.5.2 Well-structuredness vs. Soundness in BPMN

Considering the relationship between well-structuredness and soundness Section 4.4.1 proved that a well-structured process is always sound (Theorem 3), but there are sound processes that are not well-structured. To show that this result is still valid in the new framework, first it is demostrated that a reachable well-structured core process element can always complete its execution (Lemma 7). This latter Lemma 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. The proof is equal to the one reported in Section 7.4.2, with the exception of the following Lemma.

Lemma 7. *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). By induction on the structure of well-structured core process. □

6.5.3 Safeness vs. Soundness in BPMN

Considering the relationship between safeness and soundness it results that there are unsafe models that are sound. This is a peculiarity of BPMN, thank to its capability to support also (unsafe) sub-processes.

Reasoning at the process level, the following Theorem holds.

Theorem 12. *Let P be a process, P is unsafe does not imply P is unsound.*

Proof (sketch). By contradiction. □

Considering now the collaboration level, it results that there are unsafe collaborations that could be either sound or unsound, as proved by the following Theorem.

Theorem 13. *Let C be a collaboration, C is unsafe does not imply C is unsound.*

Proof (sketch). By contradiction. □

Running Example 18. *In the collaboration in the running example (Figure 6.1 (A)), process P_{pc} are safe (and sound) and P_{ca} while process P_r is not safe but sound. In fact, the sub-process completes only when all the generated tokens arrive at its end event, and only one token is moved to the including process. Thus, the collaboration is not safe but it is both sound and message-relaxed sound.* □

6.6 Compositionality of Safeness and Soundness

This section studies safeness and soundness compositionality at process level, by considering the compositionality of sub-processes. Specifically, it investigates how the sub-processes behaviour impacts on the safeness and soundness of processes including them.

6.6.1 On Compositionality of Safeness

In the following it is shown that the composition of unsafe sub-processes always results in un-safe processes, but the vice versa does not hold. There are also un-safe processes including safe sub-processes when the unsafeness does not depend from the behaviour of the sub-process.

Theorem 14. *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). By contradiction (see Appendix C.3). □

Running Example 19. *The running example in Figure 6.1 (A) includes a sub-process. As shown in Section 6.3.2 processes P_{pc} and P_{ca} are safe while process P_r is not safe. Thus, the unsafeness of the sub-process impacts on the safeness of the including process.*

6.6.2 On Compositionality of Soundness

Considering soundness in a multi-layer structure, it results that the composition of unsound sub-processes does not result in un-sound processes. There are also sound processes including unsound sub-processes. In fact, when putting unsound sub-processes together in a process, this could be either sound or unsound.

Theorem 15. *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). By contradiction (see Appendix C.3). □

6.7 Comparison with other Approaches

Much effort has been devoted to formalise the BPMN standard. However, less attention has been paid to provide a direct formalisation taking into account message exchange and sub-process behaviour, as well their impact

on collaborations. This section discusses the most relevant related work. First the available formalisations are discussed and then an investigation on correctness properties is reported.

6.7.1 Sub-Processes Formalisations

Regarding the formalisation of BPMN models, the notion of sub-process has not been extensively studied yet. Among the others, Borger and Thalheim [14], Christiansen et al. [18], El-Saber and Boronat [35], and Kossak et al. [47] provide a direct formalisation for a core subset of BPMN elements. However, none of them is able to reason on collaborations including at the same time sub-processes, messages and their interplay. Differently, in [47] the authors present a sub-process semantics. However, differently from what prescribed by the BPMN standard, when an end event of the sub-process is reached the flow immediately moves back up to the higher-level parent process. Moreover, the paper skips the problems derived by the combined usage of messages and sub-processes, by saying that the "*semantics, however, does not change with the graphical depiction, that is, a collapsed sub-process must have the same semantics as when it is expanded*".

Others contributions provide a mapping toward well known formal formalism (e.g., process algebras and Petri Nets). All these approaches suffer from issues related to encodings. In fact, the semantics given by translation is based on the low-level details of the encoding without considering the actual features and constructs of BPMN. This may make the formalisation inaccurate, since translations usually rely on assumptions and language abstractions. In particular, Dijkman et al. [29] propose a mapping from BPMN to Petri Nets. The paper introduces also sub-processes saying that its behaviour is not clear in the BPMN specification. Hence, they restrict the mapping to sub-processes with a single start event and a single end event only. Moreover, such a mapping does not work properly if there are one or more activities within the sub-process which may be executed multiple times. Hence they consider only safe sub-processes, while in the given approach no restrictions are made.

6.7.2 Reasoning on Correctness Properties

Concerning verification, different properties have been defined in the context of Petri Nets, mainly focussing on the verification of correctness properties related to the control flow of the business process, without considering communication aspects [29, 26, 104, 80, 100]. Among the others, Dijkman et al. [29] provide a mapping from BPMN to Petri Nets. In such work, safeness for a BPMN models means that any activity will ever be enabled or executed more than once concurrently. Safeness is discussed in relation to

sub-processes as a restriction of the considered models. However, definitions are just given using natural language. Moreover, the mapping does not permit to distinguish sequence and message edges. Correspondingly analysis activities based on such encoding are not able to differentiate issues concerning the control flow from those concerning the message flow. In this direction a relevant work is the one by Puhlmann and Weske [81]. The authors propose a classification of BPMN models based on different notions of soundness. Their formalisation is also able to represent the sub-process element as a node that references another process graph. However, they argue that such composed process graphs can always be flattened, thus, they consider only flat process graphs. Among the direct formalisations, El-Saber and Boronat [35] provide a formal characterisation of well-structured processes. Their formal framework includes also the sub-process element. However, restricting only to well-structured processes, other relevant properties are not taken into account.

Chapter 7

Multiple Instances and Data

Clearly understanding of interactions and data exchanges is particularly important, especially when multiple instances of interacting participants are involved. In this regard, BPMN collaboration diagrams result to be an effective way to reflect how multiple participants cooperate to reach a shared goal. Also in this context, the BPMN lack of formal semantics becomes a prominent issue. To overcome this problem, several formalisations have been proposed, mainly focussing on the control flow perspective (e.g., [29, 24, 112, 14, 105]). Less attention has been paid to provide a formal semantics capturing the interplay between control features, message exchanges, and data. These perspectives are strongly related, especially when a participant interacts with multi-instance participants. In fact, to achieve successful collaboration interactions, it is required to deliver the messages arriving at the receiver side to the appropriate instances. As messages are used to exchange data between participants, the BPMN standard fosters the use of the content of the messages themselves to correlate them with the corresponding instances. Thus, the data perspective plays a crucial role when considering multi-instance collaborations. Despite this, no formal semantics that considers all together these key aspects of BPMN collaboration models has been yet proposed in the literature.

This chapter defines a formal semantics for BPMN collaborations considering control flow elements, multi-instance pools, data objects and data-based decision gateways (Section 7.2). The movement from a pure control-flow perspective to more complex models where also data and decisions are explicitly considered has also called for methods and techniques able to guarantee the correctness of such models. Therefore, also correctness properties are defined in multi-instance collaborative scenarios (Section 7.3), to check if the classification results obtained for the core subset of BPMN elements are still valid in this extended framework. Finally, an animator tool faithfully implementing the proposed formal semantics is presented (Section 7.6).

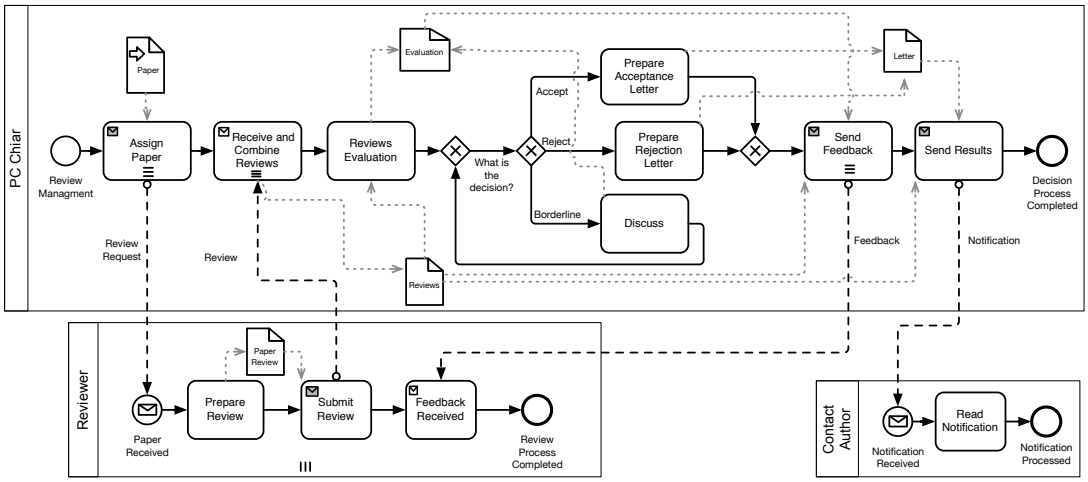


Figure 7.1: Paper Reviewing Collaboration Model.

Highlights. The main contributions of this chapter are:

- it defines a direct formal semantics for BPMN collaborations considering control flow elements, multi-instance pools, data objects and data-based decision gateways, thus providing a precise understanding of ambiguous and loose points of the standard;
- it defines the correctness properties in this data-aware setting, confirming the obtained classification results.

7.1 Running Example

To precisely deal with multiple instances in BPMN collaboration models, it is necessary to take into account the data flow. Indeed, the creation of process *instances* can be triggered by the arrival of *messages*, which contain data. Within a process instance, data is stored in *data objects*, used to drive the instance execution. Values of data objects can be used to fill the content of outgoing messages, and vice versa, the content of incoming messages can be stored in data objects. To clarify the interplay between such concepts the reviewing process presented in Section 2.2.2 is considered. The example concerns the management of a single paper, which is revised by three reviewers; of course, the management of all papers submitted to the conference requires to enact the collaboration for each paper.

In this scenario, data support is crucial to precisely render the message exchanges between participants, especially because multiple instances of the *Reviewer* process are created. In fact, messages coming into this pool might start a new process instance, or be routed to existing instances already underway. Messages and process instances must contain enough information to determine, when a message arrives at a pool, if a new process instance is

needed or, if not, which existing instance will handle it. To this aim, BPMN makes use of the concept of *correlation*: it is up to each single message to provide the information that permits to associate the message with the appropriate (possibly new) instance. This is achieved by embedding values, called *correlation data*, in the content of the message itself. Pattern-matching is used to associate a message to a distinct receiving task or event. In the considered example, every time the chair sends back a feedback to a reviewer, the message must contain information (in the presented case reviewer name and paper title) to be correlated to the correct process instance of *Reviewer*.

According to the BPMN standard, data objects do not have any direct effect on the sequence flow or message flow of processes, since tokens do not flow along data associations [68, p. 221]. However, this statement is questionable. Indeed, on the one hand, the information stored in data objects can be used to drive the execution of process instances, as they can be referred in the conditional expressions of XOR gateways to take decisions about which branch should be taken. On the other hand, data objects can be connected in input to tasks. In particular, the standard states that *"the Data Objects as inputs into the Tasks act as an additional constraint for the performance of those Tasks. The performers [...] cannot start the Task without the appropriate input"* [68, p. 183]. In both cases, a data object has an implicit indirect effect on the execution, since it can drive the decision taken by a XOR gateway or act as a guard condition on a task. For instance, in the running example, according to the value of the *Evaluation* data object, the conditional expression *What is the decision?* is evaluated and a branch of the XOR-Split gateway is chosen. As another example, the task *Send Results* can be executed only if an acceptance or rejection letter is stored in the *Letter* data object.

Concerning the content of data objects, the standard left underspecified its structure, in order to keep the notation independent from the kind of data structure required from time to time. Here a generic record structure is considered, assuming that a data object is just a list of fields, characterised by a name and the corresponding value. Of course, a field can be used to represent the state of a data object. More complex XML-like structures, which are out of the scope of the thesis, can be anyway rendered resorting to nesting. The structure in terms of fields of the data objects used in the running example is specified in Figure 7.2(a). Messages are structured as well; the structure of the messages specified in the example is shown in Figure 7.2(b). Values can be manipulated and inserted into data object fields via assignments performed by tasks.

Guards, assignments, and structure of data objects and messages are not explicitly reported in the graphical representation of the BPMN model, but are defined as attributes of the involved BPMN elements. Information on

(a)	Paper {title, contact, authors, body} Reviews {title, reviewers, scores, bodies} Evaluation {title, decision} Letter {title, evaluation} PaperReview {title, score, body}
(b)	ReviewRequest {title, body} Notification {title, contact, authors, evaluation, scores, bodies} Review {reviewerName, title, score, body} Feedback {reviewerName, title, evaluation}

Figure 7.2: Structures of Data Objects (a) and Messages (b) of the Paper Reviewing Example.

their definition and functioning are provided in Section 7.2.

7.2 Formal Framework

This section formalises the semantics of BPMN collaborations supporting multiple instances. The formalisation focuses on those BPMN elements, informally presented in the previous section, that are strictly needed to deal with multiple instantiation of collaborations, namely multi-instance pools, message exchange events and tasks, and data objects; additionally, in order to define meaningful collaborations, also some core BPMN elements are considered, whose formalisation has been given in Section 3.2

7.2.1 Syntax

Figure 7.3 reports the BNF syntax defining the textual notation of BPMN collaboration models. This syntax only describes the structure of models, without taking into account all those aspects that come into play to describe the model semantics, such as token distribution and messages. In the proposed grammar, the non-terminal symbols C, P and A represent *Collaboration Structures*, *Process Structures* and *Data Assignments*, respectively. The first two syntactic categories directly refer to the corresponding notions in

$$
\begin{array}{lcl}
C & ::= & \mathsf{pool}(\mathsf{p}, P) \;\mid\; \mathsf{miPool}(\mathsf{p}, P) \;\mid\; C_1 \parallel C_2 \\
P & ::= & \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \;\mid\; \mathsf{startRcv}(\mathsf{m}:\tilde{\mathsf{t}}, \mathsf{e}_o) \;\mid\; \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}) \\
 & \mid & \mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}:\tilde{\mathsf{exp}}, \mathsf{e}_{cmp}) \;\mid\; \mathsf{terminate}(\mathsf{e}_i) \;\mid\; \mathsf{andSplit}(\mathsf{e}_i, E_o) \\
 & \mid & \mathsf{xorSplit}(\mathsf{e}_i, G) \;\mid\; \mathsf{andJoin}(E_i, \mathsf{e}_o) \;\mid\; \mathsf{xorJoin}(E_i, \mathsf{e}_o) \\
 & \mid & \mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1:\tilde{\mathsf{t}}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_h:\tilde{\mathsf{t}}_h, \mathsf{e}_{oh})) \\
 & \mid & \mathsf{task}(\mathsf{e}_i, \mathsf{exp}, A, \mathsf{e}_o) \;\mid\; \mathsf{taskRcv}(\mathsf{e}_i, \mathsf{exp}, A, \mathsf{m}:\tilde{\mathsf{t}}, \mathsf{e}_o) \\
 & \mid & \mathsf{taskSnd}(\mathsf{e}_i, \mathsf{exp}, A, \mathsf{m}:\tilde{\mathsf{exp}}, \mathsf{e}_o) \;\mid\; \mathsf{empty}(\mathsf{e}_i, \mathsf{e}_o) \\
 & \mid & \mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}:\tilde{\mathsf{t}}, \mathsf{e}_o) \;\mid\; \mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}:\tilde{\mathsf{exp}}, \mathsf{e}_o) \;\mid\; P_1 \parallel P_2 \\
A & ::= & \epsilon \;\mid \mathsf{d.f} ::= \mathsf{exp}, A
\end{array}
$$

Figure 7.3: BNF Syntax of BPMN Collaboration Structures.

BPMN, while the latter refers to list of assignments used to specify updating of data objects. The terminal symbols, denoted by the sans serif font, are the typical elements of a BPMN model, i.e. pools, events, tasks and gateways.

There is no direct syntactic representation of *Data Objects*. The evolution of their state during the model execution is a semantic concern (described later in this section). Thus, syntactically, only the connections between data objects and the other elements are relevant. They are rendered by references to data objects within *expressions*, used to check when a task is ready to start (graphically, the task has a connection incoming from the data object), to update the values stored in a data object (graphically, the task has a connection outgoing to the data object), and to drive the decision of a XOR-Split gateway. The standard is quite loose in specifying what is the actual structure of data objects; a generic record structure is considered, assuming that the data object is just a list of fields, characterised by a name and the corresponding value. Specifically, the field f of the data object named d is accessed via the usual notation $\mathsf{d.f}$. It is assumed that different pools use data objects with different names (this can be easily achieved by prefixing a data object name with the name of the enclosing pool).

Intuitively, a BPMN collaboration model is rendered in this syntax as a collection of (single-instance and multi-instance) pools, each one specifying a process. Formally, a collaboration C is a composition, by means of the $\|$ operator, of pools either of the form $\mathsf{pool}(\mathsf{p}, P)$ (for single-instance pools) or $\mathsf{miPool}(\mathsf{p}, P)$ (for multi-instance pools), where p is the name that uniquely identifies the pool, and P is the enclosed process. At process level, as usual, $\mathsf{e} \in \mathbb{E}$ uniquely denotes a *sequence edge*, while $E \in 2^{\mathbb{E}}$ a set of edges. Notably, $|E| > 1$ when E is used in joining and splitting gateways; similarly, an event-based gateway contains at least two message events, i.e. $h > 1$ in each $\mathsf{eventBased}$ term. For the convenience of the reader, e_i refers to the edge incoming in an element, e_o to the outgoing edge, e_{enb} to the (spurious) edge denoting the enabled status of a start event and e_{cmp} to the (spurious) edge denoting the completed status of end events. Function $edges(P)$ will be used to get the set of all edges used in the process P.

In the considered data-based setting, messages may carry values. Therefore, a sending action specifies a list of expressions whose evaluation will return a tuple of values to be sent, while a receiving action specifies a template to select matching messages and possibly assign values to data object fields. Formally, a *message* is a pair $\mathsf{m} : \tilde{\mathsf{v}}$, where $\mathsf{m} \in \mathbb{M}$ is the (unique) message name (i.e., the label of the message edge), and $\tilde{\mathsf{v}}$ is a tuple of values, with $\mathsf{v} \in \mathbb{V}$ and $\tilde{\cdot}$ denoting tuples (i.e., $\tilde{\mathsf{v}}$ stands for $\langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$). Sending actions have as argument a pair of the form $\mathsf{m} : \tilde{\mathsf{exp}}$. The precise syntax of *expressions* is deliberately not specified, it is just assumed that they contain, at least, values v and data object fields $\mathsf{d.f}$. Receiving actions have

as argument a pair of the form $\mathsf{m} : \tilde{\mathsf{t}}$, where $\tilde{\mathsf{t}}$ denotes a *template*, that is a sequence of expressions and formal fields used as pattern to select messages received by the pool. Formal fields are data object fields identified by the ?-tag (e.g., ?$\mathsf{d.f}$) and are used to bind fields to values. In order to store the received values and allow their reuse, each message in the receiving process is associated a data object, whose name coincides with the message name. Data objects are associated to a task by means of a conditional expression, which is a guard enabling the task execution, and a list of *assignments* A, each of which assigns the value of an expression to a data field. When there is no data object as input to a task, the guard is simply true, while if there is no data object in output to a task the list of assignments is empty (ϵ).

The XOR-Split gateway specifies *guard conditions* in its outgoing edges, used to decide which edge to activate according to the values of data objects. This is formally rendered as a function $G : \mathbb{E} \rightarrow \mathbb{EXP}$ mapping edges to conditional expressions, where $\mathbb{EXP}$ is the set of all expressions that includes the distinguished expression $\mathsf{default}$ referring to the *default sequence edge* outgoing from the gateway (it is assigned to at most one edge). When convenient, function G will be considered as a set of pairs $(\mathsf{e}, \mathsf{exp})$.

The correspondence between the syntax used here to represent multi-instance collaborations and the graphical notation of BPMN is exemplified by means of the running example in Figure 7.4 and Figure 7.5, while the detailed one-to-one correspondence [1] is available in Appendix A.4. Notably, in the textual notation there is no direct representation of the sequential multi-instance task, which is anyway simply rendered as a macro where the task is enclosed in a *for* loop (expressed by means of a pair of XOR join and split gateways, and an additional data object c_i for the loop counter). Moreover, to properly manage lists of reviewers, scores and review bodies in the PC Chair process, fields with vector-like data structure are used, equipped with the typical $\mathsf{add}()$ and $\mathsf{next}()$ functionalities. It is assumed an $\mathsf{evaluate}()$ expression to combine review scores in a decision. Finally, to simplify the definition of well-structured processes (given later), as usual, an *empty* task is included in the syntax.

It is worth noticing that the following assumptions are in place. First, when a process is instantiated by means of a message start event, then this is the only starting event in the process. Second, processes of multi-instance pools can be instantiated only by a message start event.

[1] Notably, in the textual representation there is some information (messages content, receiving templates, data object assignments, etc.) that is not reported in the graphical notation. In fact, for the sake of understandability, according to the BPMN standard these technical details of collaborations are not part of the graphical representation, but they are part of the low-level XML representation. This information is explicitly reported in the textual representation as it is needed to properly define the execution semantics of the collaboration models.

Overall paper reviewing collaboration scenario:
$\mathsf{pool}(\mathsf{PC\ Chair}, P_{pc}) \parallel \mathsf{miPool}(\mathsf{Reviewer}, P_r) \parallel \mathsf{pool}(\mathsf{Contact\ Author}, P_{ca})$

Reviewer process :

$P_r = \mathsf{startRcv}(\mathsf{ReviewRequest}:\tilde{t_2}, e_{15}) \parallel \mathsf{task}(e_{15}, \mathsf{true}, A_6, e_{16}) \parallel$
$\mathsf{taskSnd}(e_{16}, \mathsf{exp}_7, \epsilon, \mathsf{Review}:\tilde{exp}_8, e_{17}) \parallel$
$\mathsf{taskRcv}(e_{17}, \mathsf{true}, \epsilon, \mathsf{Feedback}:\tilde{t_3}, e_{18}) \parallel \mathsf{end}(e_{18}, e_{cmp})$

Templates, *expressions*, *assignments* :

$\tilde{t_2} = \langle ?\mathsf{ReviewRequest.title}, ?\mathsf{ReviewRequest.body}\rangle$
$A_6 = \mathsf{PaperReview.title} := \mathsf{ReviewRequest.title},$
$\mathsf{PaperReview.score} := \mathsf{assignscore}(\mathsf{ReviewRequest.body}),$
$\mathsf{PaperReview.body} := \mathsf{writeReview}(\mathsf{ReviewRequest.body})$
$\mathsf{exp}_7 = \mathsf{PaperReview.score} \neq \mathsf{null}\ \mathsf{and}\ \mathsf{PaperReview.body} \neq \mathsf{null}$
$\tilde{exp}_8 = \langle \mathsf{myName}(), \mathsf{PaperReview.title}, \mathsf{PaperReview.score}, \mathsf{PaperReview.body}\rangle$
$\tilde{t_3} = \langle \mathsf{myName}(), \mathsf{ReviewRequest.title}, ?\mathsf{Feedback.evaluation}\rangle$

PC Chair process :

$P_{pc} = \mathsf{start}(e_{enb}, e_1) \parallel \mathsf{xorJoin}(\{e_1, e_1'''\}, e_1') \parallel$
$\mathsf{taskSnd}(e_1', c_1.c \neq \mathsf{null}, c_1.c := c_1.c + 1, \mathsf{ReviewRequest}:\tilde{exp}_1, e_1'') \parallel$
$\mathsf{xorSplit}(e_1'', \{(e_1''', c_1.c \leqslant 3), (e_2, \mathsf{default})\}) \parallel \mathsf{xorJoin}(\{e_2, e_2'''\}, e_2') \parallel$
$\mathsf{task}(e_6, \mathsf{true}, A_3, e_9) \parallel \mathsf{task}(e_7, \mathsf{true}, A_4, e_{10}) \parallel$
$\mathsf{task}(e_8, \mathsf{true}, A_5, e_{11}) \parallel \mathsf{xorJoin}(\{e_9, e_{10}\}, e_{12}) \parallel \mathsf{xorJoin}(\{e_{12}, e_{12}'''\}, e_{12}') \parallel$
$\mathsf{taskSnd}(e_{12}', \mathsf{exp}_3, c_3.c := c_3.c + 1, \mathsf{Feedback}:\tilde{exp}_4, e_{12}'') \parallel$
$\mathsf{xorSplit}(e_{12}'', \{(e_{12}''', c_3.c \leqslant 3), (e_{13}, \mathsf{default})\}) \parallel$
$\mathsf{taskSnd}(e_{13}, \mathsf{exp}_5, \epsilon, \mathsf{Notification}:\tilde{exp}_6, e_{14}) \parallel$
$\mathsf{end}(e_{14}, e_{cmp})$

Templates, *expressions*, *assignments* :

$\tilde{exp}_1 = \langle \mathsf{Paper.title}, \mathsf{Paper.body}\rangle$
$A_1 = \mathsf{Reviews.title} := \mathsf{Paper.title},$
$\mathsf{Reviews.reviewers} := \mathsf{add}(\mathsf{Reviews.reviewers}, \mathsf{Review.reviewerName}),$
$\mathsf{Reviews.scores} := \mathsf{add}(\mathsf{Reviews.scores}, \mathsf{Review.score}),$
$\mathsf{Reviews.bodies} := \mathsf{add}(\mathsf{Reviews.bodies}, \mathsf{Review.body})$
$\tilde{t_1} = \langle ?\mathsf{Review.reviewerName}, \mathsf{Paper.title}, ?\mathsf{Review.score}, ?\mathsf{Review.body}\rangle$
$\mathsf{exp}_2 = \mathsf{Reviews.scores} \neq \mathsf{null}\ \mathsf{and}$
$\mathsf{Reviews.bodies} \neq \mathsf{null}\ \mathsf{and}\ \mathsf{Reviews.reviewers} \neq \mathsf{null}$
$A_2 = \mathsf{Evaluation.title} := \mathsf{Paper.title},$
$\mathsf{Evaluation.decision} = \mathsf{evaluate}(\mathsf{Reviews.scores})$
$A_3 = \mathsf{Letter.title} := \mathsf{Paper.title},\ \mathsf{Letter.evaluation} := \mathsf{accept}$
$A_4 = \mathsf{Letter.title} := \mathsf{Paper.title},\ \mathsf{Letter.evaluation} := \mathsf{reject}$
$A_5 = \mathsf{Evaluation.title} := \mathsf{Paper.title},$
$\mathsf{Evaluation.decision} := \mathsf{evaluate}(\mathsf{Reviews.scores})$
$\mathsf{exp}_3 = \mathsf{Reviews.reviewers} \neq \mathsf{null}\ \mathsf{and}\ \mathsf{Evaluation.decision} \neq \mathsf{null}\ \mathsf{and}\ c_3.c \neq \mathsf{null}$
$\tilde{exp}_4 = \langle \mathsf{next}(\mathsf{Reviews.reviewers}), \mathsf{Paper.title}, \mathsf{Evaluation.decision}\rangle$
$\mathsf{exp}_5 = \mathsf{Reviews.scores} \neq \mathsf{null}\ \mathsf{and}\ \mathsf{Reviews.bodies} \neq \mathsf{null}\ \mathsf{and}\ \mathsf{Letter.title} \neq \mathsf{null}$
$\mathsf{and}\ \mathsf{Letter.evaluation} = \mathsf{Evalution.score}$
$\tilde{exp}_6 = \langle \mathsf{Paper.title}, \mathsf{Paper.contact}, \ldots, \mathsf{Reviews.bodies}\rangle$

Figure 7.4: Textual Representation of the Running Example (Part I).

Reviewer process :
$P_{ca} = \mathsf{startRcv}(\mathsf{Notification}:\tilde{\mathsf{t}_4}, \mathsf{e}_{19}) \parallel \mathsf{task}(\mathsf{e}_{19}, \mathsf{true}, \epsilon, \mathsf{e}_{20}) \parallel \mathsf{end}(\mathsf{e}_{20}, \mathsf{e}_{\mathsf{cmp}})$

Templates, *expressions*, *assignments* :
$\tilde{\mathsf{t}_4} = \langle ?\mathsf{Notification.title}, ?\mathsf{Notification.contact}, \ldots, ?\mathsf{Notification.bodies} \rangle$

Figure 7.5: Textual Representation of the Running Example (Part II).

The messages received by this event will carry the input data for the new instances, i.e. no data input element is used in multi-instance pools.

7.2.2 Semantics

The syntax presented so far represents the mere structure of processes and collaborations. To describe their semantics, sequence edges are marked by means of tokens [68, p. 27]. In particular, the structural information is enriched with a notion of execution state, defined by the state of each process instance (given by the marking of sequence edges and the values of data object fields) and the store of the exchanged messages. These stateful descriptions are called process configurations and collaboration configurations.

Formally, a *process configuration* has the form $\langle P, \sigma, \alpha \rangle$, where: P is a process structure; $\sigma : \mathbb{E} \to \mathbb{N}$ is a *sequence edge state function* specifying, for each sequence edge, the current number of tokens marking it ($\mathbb{N}$ is the set of natural numbers); and $\alpha : \mathbb{F} \to \mathbb{V}$ is the *data state function* assigning values (possibly null) to data object fields ($\mathbb{F}$ is the set of data fields and $\mathbb{V}$ the set of values). σ_0 (resp. α_0) denotes the edge (resp. data) state where all edges are unmarked (resp. all fields are set to null), formally, $\sigma_0(\mathsf{e}) = 0$ $\forall \mathsf{e} \in \mathbb{E}$ and $\alpha_0(\mathsf{d.f}) = \mathsf{null}$ $\forall \mathsf{d.f} \in \mathbb{F}$. The state obtained by updating in 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 $\sigma(\mathsf{e}')$. The update of data state α is similarly defined. To simplify the definition of the operational rules, some auxiliary functions to update states are introduced. Function $inc : \mathbb{S}_\sigma \times \mathbb{E} \to \mathbb{S}_\sigma$ (resp. $dec : \mathbb{S}_\sigma \times \mathbb{E} \to \mathbb{S}_\sigma$), where $\mathbb{S}_\sigma$ is the set of edge states, updates a state by incrementing (resp. decrementing) by one the number of tokens marking an edge in the state. As usual, they are defined as $inc(\sigma, \mathsf{e}) = \sigma \cdot [\mathsf{e} \mapsto \sigma(\mathsf{e}) + 1]$ and $dec(\sigma, \mathsf{e}) = \sigma \cdot [\mathsf{e} \mapsto \sigma(\mathsf{e}) - 1]$. These functions extend in a natural way to sets E of edges as follows: $inc(\sigma, \varnothing) = \sigma$ and $inc(\sigma, \{\mathsf{e}\} \cup E)) = inc(inc(\sigma, \mathsf{e}), E)$; the cases for dec are similar. Function $reset : \mathbb{S}_\sigma \times \mathbb{E} \to \mathbb{S}_\sigma$, instead, is used to update an edge state by setting to zero the number of tokens marking an edge in the state: $reset(\sigma, \mathsf{e}) = \sigma \cdot [\mathsf{e} \mapsto 0]$. Also in this case the function extends in a natural way to sets of edges as follows: $reset(\sigma, \varnothing) = \sigma$ and $reset(\sigma, \{\mathsf{e}\} \cup E)) = reset(reset(\sigma, \mathsf{e}), E)$. The *evaluation* relation $eval \subseteq \mathbb{EXP} \times \mathbb{S}_\alpha \times \mathbb{V}$ is used to evaluate an expression

over a data state. This is a relation, not a function, because an expression may contain non-deterministic operators and, in such a case, its evaluation results in one of the possible values for that expression with respect to the given data state. Notation $eval(\mathsf{exp}, \alpha, \mathsf{v})$ states that v is one of the possible values resulting from the evaluation of the expression exp on the data state α. This relation is not explicitly defined, since the syntax of expressions is deliberately not specified; it is only assumed that $eval(\mathsf{default}, \alpha, \mathsf{v})$ implies $\mathsf{v} = \mathit{false}$ for any α. The relation extends to tuples component-wise. Finally, relation $upd \subseteq \mathbb{S}_\alpha \times \mathbb{A}^n \times \mathbb{S}_\alpha$, where $\mathbb{S}_\alpha$ is the set of data states and $\mathbb{A}$ is the set of assignments, is used to update data object values. Notation $upd(\alpha, A, \alpha')$ states that α' is one of the possible states resulting from the update of α with assignment A. The relation is inductively defined as follows: for any α, $upd(\alpha, \epsilon, \alpha)$; $upd(\alpha, \mathsf{d.f} := \mathsf{exp}, \alpha \cdot [\mathsf{d.f} \mapsto \mathsf{v}])$ with v such that $eval(\mathsf{exp}, \alpha, \mathsf{v})$; and $upd(\alpha, (A_1, A_2), \alpha'')$ with α'' such that $upd(\alpha', A_2, \alpha'')$ and α' such that $upd(\alpha, A_1, \alpha')$.

A *collaboration configuration* has the form $\langle C, \iota, \delta \rangle$, where: C is a collaboration structure; $\iota : \mathbb{P} \to 2^{\mathbb{S}_\sigma \times \mathbb{S}_\alpha}$ is the *instance state function* mapping each pool name ($\mathbb{P}$ is the set of pool names) to a multiset of instance states (ranged over by I and containing pairs of the form $\langle \sigma, \alpha \rangle$); and $\delta : \mathbb{M} \to 2^{\mathbb{V}^n}$ is a *message state function* specifying, for each message name m, a multiset of value tuples representing the messages received along the message edge labelled by m. Function δ can be updated in a way similar to σ, enabling the definition of the following auxiliary functions. Function $add : \mathbb{S}_\delta \times \mathbb{M} \times \mathbb{V}^n \to \mathbb{S}_\delta$ (resp. $rm : \mathbb{S}_\delta \times \mathbb{M} \times \mathbb{V}^n \to \mathbb{S}_\delta$), where $\mathbb{S}_\delta$ is the set of message states, allows updating a message state by adding (resp. removing) a value tuple for a given message name in the state: $add(\delta, \mathsf{m}, \tilde{\mathsf{v}}) = \delta \cdot [\mathsf{m} \mapsto \delta(\mathsf{m}) + \{\tilde{\mathsf{v}}\}]$ and $rm(\delta, \mathsf{m}, \tilde{\mathsf{v}}) = \delta \cdot [\mathsf{m} \mapsto \delta(\mathsf{m}) - \{\tilde{\mathsf{v}}\}]$, where $+$ and $-$ are the union and substraction operations on multisets. The instance state function ι can be updated in two ways: by adding a newly created instance or by modifying an existing one: $newI(\iota, \mathsf{p}, \sigma, \alpha) = \iota \cdot [\mathsf{p} \mapsto \iota(\mathsf{p}) + \{\langle \sigma, \alpha \rangle\}]$ and $updI(\iota, \mathsf{p}, I) = \iota \cdot [\mathsf{p} \mapsto I]$.

Finally, a collaboration is in the initial state when there exists a process in the collaboration whose start event is enabled, i.e. it has a token in its enabling edge, while all other sequence edges and messages edges must be unmarked and no instance of multi-instance pools has been created yet.

Definition 25 (Initial state of a process)**.** *Let $\langle P, \sigma, \alpha \rangle$ be a process configuration. Predicate $isInit(P, \sigma, \alpha)$ holds if $\sigma(start(P)) = 1$ and $\forall\ \mathsf{e} \in edges(P) \backslash start(P)\ .\ \sigma(\mathsf{e}) = 0$.*

Definition 26 (Initial state of a collaboration)**.** *Let $\langle C, \iota, \delta \rangle$ be a collaboration configuration. Predicate $isInit(C, \iota, \delta)$ holds if $\forall\ \mathsf{m} \in \mathbb{M}\ .\ \delta(\mathsf{m}) = 0$ and*

- *$\forall\, \mathsf{pool}(\mathsf{p}, P)$ in C, we have that $isInit(P, \sigma, \alpha)$ holds for $\iota(\mathsf{p}) = \{\langle \sigma, \alpha \rangle\}$;*

- $\forall\, \mathsf{miPool}(\mathsf{p}, P)$ *in* C*, we have that* $\iota(\mathsf{p}) = \varnothing$*.*

Running Example 20. *The scenario in the running example in its initial state is rendered as the collaboration configuration* $\langle(\mathsf{pool}(\mathsf{PC\ Chair}, P_{pc}) \parallel \mathsf{miPool}(\mathsf{Reviewer}, P_r) \parallel \mathsf{pool}(\mathsf{Contact\ Author}, P_{ca})), \iota, \delta\rangle$ *where:* $\iota(\mathsf{PC\ Chair}) = \{\langle\sigma, \alpha\rangle\}$ *with* $\sigma = \sigma_0 \cdot [\mathsf{e}_{enb} \mapsto 1]$ *and* $\alpha = \alpha_0 \cdot [\mathsf{Paper.title}, \ldots, \mathsf{Paper.body} \mapsto \mathrm{title}, \ldots, \mathrm{text}]$*; and* $\iota(\mathsf{Reviewer}) = \iota(\mathsf{Contact\ Author}) = \varnothing$*. The* α *function of the* p_{pc} *instance is initialised with the content of the* Paper *data input.* □

The operational semantics is defined by means of a LTS, whose definition relies on an auxiliary LTS on the behaviour of processes. The latter is a triple $\langle \mathcal{P}, \mathcal{L}, \rightarrow \rangle$ where: $\mathcal{P}$, ranged over by $\langle P, \sigma, \alpha\rangle$, is a set of process configurations; $\mathcal{L}$, ranged over by ℓ, is a set of *labels*; and $\rightarrow \subseteq \mathcal{P} \times \mathcal{L} \times \mathcal{P}$ is a *transition relation.* It will be written $\langle P, \sigma, \alpha\rangle \xrightarrow{\ell} \langle P, \sigma', \alpha'\rangle$ to indicate that $(\langle P, \sigma, \alpha\rangle, \ell, \langle P, \sigma', \alpha'\rangle) \in \rightarrow$, and say that 'the process in the configuration $\langle P, \sigma, \alpha\rangle$ can do a transition labelled by ℓ and become the process configuration $\langle P, \sigma', \alpha'\rangle$ in doing so'.

Since process execution only affects the current states, and not the process structure, for the sake of readability the structure is omitted from the target configuration of the transition. Similarly, to further improve readability, α is omitted when it is not affected by the transition. Thus, for example, a transition $\langle P, \sigma, \alpha\rangle \xrightarrow{\ell} \langle P, \sigma', \alpha\rangle$ can be written as $\langle P, \sigma, \alpha\rangle \xrightarrow{\ell} \sigma'$.

The labels used by the process transition relation are generated by the following production rules:

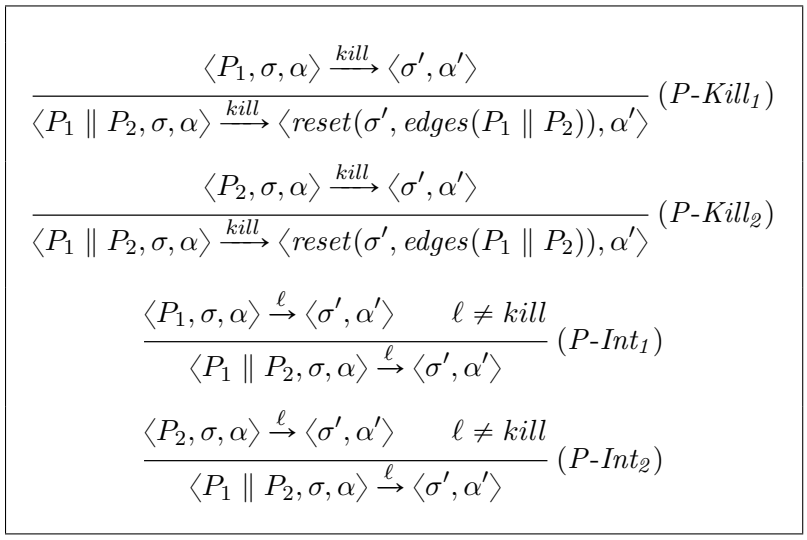
$$\ell \ ::= \ \tau \quad | \quad !\mathsf{m}:\tilde{\mathsf{v}} \quad | \quad ?\mathsf{m}:\tilde{\mathsf{et}}, A \quad | \quad \mathit{new}\,\mathsf{m}:\tilde{\mathsf{et}} \qquad \tau \ ::= \ \epsilon \mid \mathit{kill}$$

The meaning of labels is as follows. Label τ denotes an action internal to the process, while $!\mathsf{m}:\tilde{\mathsf{v}}$ and $?\mathsf{m}:\tilde{\mathsf{et}}, A$ denote sending and receiving actions, respectively. Notation $\tilde{\mathsf{et}}$ denotes an evaluated template, that is a sequence of values and formal fields. Notably, the receiving label carries information about the data assignments A to be executed, at collaboration level, after the message m is actually received. Label $\mathit{new}\,\mathsf{m}:\tilde{\mathsf{et}}$ denotes taking place of a receiving action that instantiates a new process instance (i.e., it corresponds to the occurrence of a start message event in a multi-instance pool). The meaning of internal actions is as follows: ϵ denotes an internal computation concerning the movement of tokens, while *kill* denotes taking place of the termination event.

The operational rules defining the transition relation of the processes semantics are given in Figure 7.6 and Figure 7.7. Now the rules are briefly described.

$\langle \mathsf{start}(\mathsf{e},\mathsf{e}'),\sigma,\alpha\rangle \xrightarrow{\epsilon} inc(reset(\sigma,\mathsf{e}),\mathsf{e}')$	$\sigma(\mathsf{e}) > 0$	$(P\text{-}Start)$
$\langle \mathsf{end}(\mathsf{e},\mathsf{e}'),\sigma,\alpha\rangle \xrightarrow{\epsilon} inc(dec(\sigma,\mathsf{e}),\mathsf{e}')$	$\sigma(\mathsf{e}) > 0$	$(P\text{-}End)$
$\langle \mathsf{terminate}(\mathsf{e}),\sigma,\alpha\rangle \xrightarrow{kill} dec(\sigma,\mathsf{e})$	$\sigma(\mathsf{e}) > 0$	$(P\text{-}Terminate)$
$\langle \mathsf{startRcv}(\mathsf{m}:\tilde{\mathsf{t}},\mathsf{e}),\sigma,\alpha\rangle \xrightarrow{new\ \mathsf{m}:\ \tilde{\mathsf{et}}} inc(\sigma,\mathsf{e})$	$eval(\tilde{\mathsf{t}},\alpha,\tilde{\mathsf{et}})$	$(P\text{-}StartRcv)$
$\langle \mathsf{endSnd}(\mathsf{e},\mathsf{m}:\tilde{\mathsf{exp}},\mathsf{e}'),\sigma,\alpha\rangle \xrightarrow{!\mathsf{m}:\tilde{\mathsf{v}}} inc(dec(\sigma,\mathsf{e}),\mathsf{e}')$	$\sigma(\mathsf{e}) > 0$ $eval(\tilde{\mathsf{exp}},\alpha,\tilde{\mathsf{v}})$	$(P\text{-}EndSnd)$
$\langle \mathsf{andSplit}(\mathsf{e},E),\sigma,\alpha\rangle \xrightarrow{\epsilon} inc(dec(\sigma,\mathsf{e}),E)$	$\sigma(\mathsf{e}) > 0$	$(P\text{-}AndSplit)$
$\langle \mathsf{xorSplit}(\mathsf{e},\{(\mathsf{e}',\mathsf{exp})\}\cup G),\sigma,\alpha\rangle \xrightarrow{\epsilon} inc(dec(\sigma,\mathsf{e}),\mathsf{e}')$	$\sigma(\mathsf{e}) > 0,$ $eval(\mathsf{exp},\alpha,true)$	$(P\text{-}XorSplit_1)$
$\langle \mathsf{xorSplit}(\mathsf{e},\{(\mathsf{e}',\mathsf{default})\}\cup G),\sigma,\alpha\rangle \xrightarrow{\epsilon} inc(dec(\sigma,\mathsf{e}),\mathsf{e}')$	$\sigma(\mathsf{e}) > 0,$ $\forall(\mathsf{e}_j,\mathsf{exp}_j)\in G\ .$ $eval(\mathsf{exp}_j,\alpha,false)$	$(P\text{-}XorSplit_2)$
$\langle \mathsf{andJoin}(E,\mathsf{e}),\sigma,\alpha\rangle \xrightarrow{\epsilon} inc(dec(\sigma,E),\mathsf{e})$	$\forall \mathsf{e}'\in E\ .\ \sigma(\mathsf{e}') > 0$	$(P\text{-}AndJoin)$
$\langle \mathsf{xorJoin}(\{\mathsf{e}\}\cup E,\mathsf{e}'),\sigma,\alpha\rangle \xrightarrow{\epsilon} inc(dec(\sigma,\mathsf{e}),\mathsf{e}')$	$\sigma(\mathsf{e}) > 0$	$(P\text{-}XorJoin)$
$\langle \mathsf{eventBased}(\mathsf{e},(\mathsf{m}_1:\tilde{\mathsf{t}}_1,\mathsf{e}'_1),\ldots,(\mathsf{m}_h:\tilde{\mathsf{t}}_h,\mathsf{e}'_h)),\sigma,\alpha\rangle \xrightarrow{?\mathsf{m}_j\ :\ \tilde{\mathsf{et}}_j,\epsilon} inc(dec(\sigma,\mathsf{e}),\mathsf{e}'_j)$	$\sigma(\mathsf{e}) > 0, 1\leqslant j\leqslant h$ $eval(\tilde{\mathsf{t}}_j,\alpha,\tilde{\mathsf{et}}_j)$	$(P\text{-}EventG)$
$\langle \mathsf{task}(\mathsf{e},\mathsf{exp},A,\mathsf{e}'),\sigma,\alpha\rangle \xrightarrow{\epsilon} \langle inc(dec(\sigma,\mathsf{e}),\mathsf{e}'),\alpha'\rangle$	$\sigma(\mathsf{e}) > 0,$ $eval(\mathsf{exp},\alpha,true),$ $upd(\alpha,A,\alpha')$	$(P\text{-}Task)$
$\langle \mathsf{taskRcv}(\mathsf{e},\mathsf{exp},A,\mathsf{m}:\tilde{\mathsf{t}},\mathsf{e}'),\sigma,\alpha\rangle \xrightarrow{?\mathsf{m}\ :\ \tilde{\mathsf{et}},A} inc(dec(\sigma,\mathsf{e}),\mathsf{e}')$	$\sigma(\mathsf{e}) > 0,$ $eval(\mathsf{exp},\alpha,true),$ $eval(\tilde{\mathsf{t}},\alpha,\tilde{\mathsf{et}})$	$(P\text{-}TaskRcv)$
$\langle \mathsf{taskSnd}(\mathsf{e},\mathsf{exp}',A,\mathsf{m}:\tilde{\mathsf{exp}},\mathsf{e}'),\sigma,\alpha\rangle \xrightarrow{!\mathsf{m}:\tilde{\mathsf{v}}} \langle inc(dec(\sigma,\mathsf{e}),\mathsf{e}'),\alpha'\rangle$	$\sigma(\mathsf{e}) > 0,$ $eval(\mathsf{exp}',\alpha,true),$ $upd(\alpha,A,\alpha'),$ $eval(\tilde{\mathsf{exp}},\alpha,\tilde{\mathsf{v}})$	$(P\text{-}TaskSnd)$
$\langle \mathsf{empty}(\mathsf{e},\mathsf{e}'),\sigma,\alpha\rangle \xrightarrow{\epsilon} inc(dec(\sigma,\mathsf{e}),\mathsf{e}')$	$\sigma(\mathsf{e}) > 0$	$(P\text{-}Empty)$
$\langle \mathsf{interRcv}(\mathsf{e},\mathsf{m}:\tilde{\mathsf{t}},\mathsf{e}'),\sigma,\alpha\rangle \xrightarrow{?\mathsf{m}\ :\ \tilde{\mathsf{et}},\epsilon} inc(dec(\sigma,\mathsf{e}),\mathsf{e}')$	$\sigma(\mathsf{e}) > 0$ $eval(\tilde{\mathsf{t}},\alpha,\tilde{\mathsf{et}})$	$(P\text{-}InterRcv)$
$\langle \mathsf{interSnd}(\mathsf{e},\mathsf{m}:\tilde{\mathsf{exp}},\mathsf{e}'),\sigma,\alpha\rangle \xrightarrow{!\mathsf{m}:\tilde{\mathsf{v}}} inc(dec(\sigma,\mathsf{e}),\mathsf{e}')$	$\sigma(\mathsf{e}) > 0$ $eval(\tilde{\mathsf{exp}},\alpha,\tilde{\mathsf{v}})$	$(P\text{-}InterSnd)$

Figure 7.6: BPMN Process Semantics (Part I).



$$\frac{\langle P_1, \sigma, \alpha\rangle \xrightarrow{kill} \langle \sigma', \alpha'\rangle}{\langle P_1 \parallel P_2, \sigma, \alpha\rangle \xrightarrow{kill} \langle reset(\sigma', edges(P_1 \parallel P_2)), \alpha'\rangle}\ (P\text{-}Kill_1)$$

$$\frac{\langle P_2, \sigma, \alpha\rangle \xrightarrow{kill} \langle \sigma', \alpha'\rangle}{\langle P_1 \parallel P_2, \sigma, \alpha\rangle \xrightarrow{kill} \langle reset(\sigma', edges(P_1 \parallel P_2)), \alpha'\rangle}\ (P\text{-}Kill_2)$$

$$\frac{\langle P_1, \sigma, \alpha\rangle \xrightarrow{\ell} \langle \sigma', \alpha'\rangle \qquad \ell \neq kill}{\langle P_1 \parallel P_2, \sigma, \alpha\rangle \xrightarrow{\ell} \langle \sigma', \alpha'\rangle}\ (P\text{-}Int_1)$$

$$\frac{\langle P_2, \sigma, \alpha\rangle \xrightarrow{\ell} \langle \sigma', \alpha'\rangle \qquad \ell \neq kill}{\langle P_1 \parallel P_2, \sigma, \alpha\rangle \xrightarrow{\ell} \langle \sigma', \alpha'\rangle}\ (P\text{-}Int_2)$$

Figure 7.7: BPMN Process Semantics (Part II).

Rule *P-Start* starts the execution of a (single-instance) process when it has been activated (i.e., the enabling edge e_{enb} is marked). The effect of the rule is to increment the number of tokens in the edge outgoing from the start event and to reset the marking of the enabling edge. Rule *P-End* instead 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-Terminate* is similar, but it produces a kill label used to force the termination of the process instance. Rule *P-StartRcv* starts the execution of a process by producing a label denoting the creation of a new instance and containing the information for consuming a received message at the collaboration layer (see rule *C-CreateMi* in Figure 7.8). Rule *P-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. Moreover, a send label is produced in order to deliver the produced message at the collaboration layer (see rules *C-Deliver* and *C-DeliverMi* in Figure 7.8). 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*$_1$ is applied when a token is available in the incoming edge of a XOR-Split gateway and a conditional expression of one of its outgoing edges is evaluated to *true*; the rule decrements the token in the incoming edge and increments the token in the selected outgoing edge. Notably, if more edges have their guards satisfied, one of them is non-deterministically chosen. Rule *P-XorSplit*$_2$ is applied when all guard expressions are evaluated to *false*; in this case the default edge is marked. 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-EventG* is activated when there is a token in the incoming edge and there is a message m_j to be consumed, so that the application of the rule moves the token from the incoming edge to the outgoing edge corresponding to the received message. A label corresponding to the consumption of a message is observed. Rule *P-Task* deals with tasks, possibly equipped with data objects. It is activated only when the guard expression is satisfied and there is a token in the incoming edge, which is then moved to the outgoing edge. The rule also updates the values of the data objects connected in output to the task. Rule *P-TaskRcv* is similar, but it produces a label corresponding to the consumption of a message. In this case, however, the data updates are not executed, because they must be done only after the message is actually received; therefore, the assignment are passed by means of the label to the collaboration layer (see rule *C-ReceiveMi* in Figure 7.8). Rule *P-TaskSnd* sends a message, updates the data object and moves the incoming token to the outgoing edge. The produced send label is used to deliver the message at the collaboration layer (see rule *C-DeliverMi* in Figure 7.8). Notably, here only tasks with atomic execution are considered; as shown later this requirement can be relaxed. Rules *P-Kill*$_1$ and *P-Kill*$_2$ deal with the propagation of killing action on the scope of the process instance, thus resetting the marking of the instance edges. Finally, Rules *P-Int*$_1$ and *P-Int*$_2$ deal with interleaving in a standard way for process elements.

Now, the labelled transition relation on collaboration configurations formalises the message exchange and the data update according to the process evolution. The LTS is a triple $\langle \mathcal{C}, \mathcal{L}_c, \rightarrow_c \rangle$ where: $\mathcal{C}$, ranged over by $\langle C, \iota, \delta \rangle$, is a set of collaboration configurations; $\mathcal{L}_c$, ranged over by l, is a set of *labels*; and $\rightarrow_c \subseteq \mathcal{C} \times \mathcal{L}_c \times \mathcal{C}$ is a *transition relation*. The same readability simplifications used for process configuration transitions is applied. The labels used by the collaboration transition relation are generated by the following production rules:

$$l \ ::= \ \tau \quad | \quad !\mathsf{m}:\tilde{\mathsf{v}} \quad | \quad ?\mathsf{m}:\tilde{\mathsf{v}} \quad | \quad \mathit{new}\,\mathsf{m}:\tilde{\mathsf{v}}$$

Notably, at collaboration level the receiving label just keeps track of the received message. To define the collaboration semantics, an additional auxiliary function is needed: $\mathit{match}(\tilde{\mathsf{et}}, \tilde{\mathsf{v}})$ is a partial function performing *pattern-matching* on structured data (like in [77]), thus determining if an evaluated template $\tilde{\mathsf{et}}$ matches a tuple of values $\tilde{\mathsf{v}}$. A successful matching returns a list of assignments A, updating the formal fields in the template; otherwise, the function is undefined. The rules defining the *match* function are as follows:

- $\mathit{match}(\mathsf{v}, \mathsf{v}) = \epsilon$;

- $match(?\mathsf{d.f}, \mathsf{v}) = (\mathsf{d.f} ::= \mathsf{v})$
- $match((\mathsf{et}', \tilde{\mathsf{et}}), (\mathsf{v}', \tilde{\mathsf{v}})) = match(\mathsf{et}', \mathsf{v}'), match(\tilde{\mathsf{et}}, \tilde{\mathsf{v}})$

The meaning of the rules is straightforward: an evaluated template matches against a value tuple if both have the same number of fields and corresponding fields do match; two values match only if they are identical, while a formal field matches any value.

The operational rules defining the transition relation of the collaboration semantics are given in Figure 7.8.

The first two rules deal with instance creation. In the single instance case (rule *C-Create*), an instance is created only if no instance exists for the considered pool, and there is a matching message. As result, the assignments for the received data are performed, and the message is consumed. In the multi-instance case (rule *C-CreateMi*), the created instance is simply added to the multiset of existing instances of the pool. The next six rules allow a single pool, representing organisation p, to evolve according to the evolution of one of its process instances $\langle P, \sigma, \alpha \rangle$. In particular, if the process instance performs an internal action (rules *C-Internal* or *C-InternalMi*) or a receiving/delivery action (rules *C-Receive*, *C-ReceiveMi*, *C-Deliver* or *C-DeliverMi*), the pool performs the corresponding action at collaboration layer.

As for instance creation, rules *C-Receive* and *C-ReceiveMi* can be applied only if there is at least one message action. Recall indeed that at process level the receiving labels just indicate the willingness of a process instance to consume a received message, regardless the actual presence of messages. The delivering of messages is based on the *correlation* mechanism: the correlation data are identified by the template fields that are not formal (i.e., those fields requiring specific matching values). Moreover, when a process performs a sending action, the message state function is updated in order to deliver the sent message to the receiving participant. Finally, Rules *C-Int*$_1$ and *C-Int*$_2$ permit 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.

It is worth noticing that the semantics has been defined according to a global perspective. Indeed, the overall state of a collaboration is collected by functions ι and δ of its configuration. On the other hand, the global semantics of a collaboration configuration is determined, in a compositional way, by the local semantics of the involved processes, which evolve independently from each other. The use of a global perspective simplifies *(i)* the technicalities required by the formal definition of the semantics, and *(ii)* the implementation of the animation of the overall collaboration execution. The compositional definition of the semantics, anyway, would allow to easily pass

$$\frac{\iota(\mathsf{p}) = \varnothing \qquad \langle P, \sigma_0, \alpha_0\rangle \xrightarrow{new\ \mathsf{m} : \tilde{\mathsf{et}}} \langle \sigma', \alpha'\rangle \quad \tilde{\mathsf{v}} \in \delta(\mathsf{m}) \qquad match(\tilde{\mathsf{et}}, \tilde{\mathsf{v}}) = A \quad upd(\alpha', A, \alpha'')}{\langle \mathsf{pool}(\mathsf{p}, P), \iota, \delta\rangle \xrightarrow{new\ \mathsf{m} : \tilde{\mathsf{v}}} \langle newI(\iota, \mathsf{p}, \sigma', \alpha''), rm(\delta, \mathsf{m}, \tilde{\mathsf{v}})\rangle}\ (C\text{-}Create)$$

$$\frac{\langle P, \sigma_0, \alpha_0\rangle \xrightarrow{new\ \mathsf{m} : \tilde{\mathsf{et}}} \langle \sigma', \alpha'\rangle \quad \tilde{\mathsf{v}} \in \delta(\mathsf{m}) \quad match(\tilde{\mathsf{et}}, \tilde{\mathsf{v}}) = A \quad upd(\alpha', A, \alpha'')}{\langle \mathsf{miPool}(\mathsf{p}, P), \iota, \delta\rangle \xrightarrow{new\ \mathsf{m} : \tilde{\mathsf{v}}} \langle newI(\iota, \mathsf{p}, \sigma', \alpha''), rm(\delta, \mathsf{m}, \tilde{\mathsf{v}})\rangle}\ (C\text{-}CreateMi)$$

$$\frac{\iota(\mathsf{p}) = \{\langle \sigma, \alpha\rangle\} \qquad \langle P, \sigma, \alpha\rangle \xrightarrow{\tau} \langle \sigma', \alpha'\rangle}{\langle \mathsf{pool}(\mathsf{p}, P), \iota, \delta\rangle \xrightarrow{\tau} \langle updI(\iota, \mathsf{p}, \{\langle \sigma', \alpha'\rangle\}), \delta\rangle}\ (C\text{-}Internal)$$

$$\frac{\iota(\mathsf{p}) = \{\langle \sigma, \alpha\rangle\} + I \qquad \langle P, \sigma, \alpha\rangle \xrightarrow{\tau} \langle \sigma', \alpha'\rangle}{\langle \mathsf{miPool}(\mathsf{p}, P), \iota, \delta\rangle \xrightarrow{\tau} \langle updI(\iota, \mathsf{p}, \{\langle \sigma', \alpha'\rangle\} + I), \delta\rangle}\ (C\text{-}InternalMi)$$

$$\frac{\iota(\mathsf{p}) = \{\langle \sigma, \alpha\rangle\} \qquad \langle P, \sigma, \alpha\rangle \xrightarrow{?\mathsf{m} : \tilde{\mathsf{et}}, A} \langle \sigma', \alpha'\rangle \quad \tilde{\mathsf{v}} \in \delta(\mathsf{m}) \qquad match(\tilde{\mathsf{et}}, \tilde{\mathsf{v}}) = A' \qquad upd(\alpha', (A', A), \alpha'')}{\langle \mathsf{pool}(\mathsf{p}, P), \iota, \delta\rangle \xrightarrow{?\mathsf{m} : \tilde{\mathsf{v}}} \langle updI(\iota, \mathsf{p}, \{\langle \sigma', \alpha''\rangle\}), rm(\delta, \mathsf{m}, \tilde{\mathsf{v}})\rangle}\ (C\text{-}Receive)$$

$$\frac{\iota(\mathsf{p}) = \{\langle \sigma, \alpha\rangle\} + I \qquad \langle P, \sigma, \alpha\rangle \xrightarrow{?\mathsf{m} : \tilde{\mathsf{et}}, A} \langle \sigma', \alpha'\rangle \quad \tilde{\mathsf{v}} \in \delta(\mathsf{m}) \quad match(\tilde{\mathsf{et}}, \tilde{\mathsf{v}}) = A' \quad upd(\alpha', (A', A), \alpha'')}{\langle \mathsf{miPool}(\mathsf{p}, P), \iota, \delta\rangle \xrightarrow{?\mathsf{m} : \tilde{\mathsf{v}}} \langle updI(\iota, \mathsf{p}, \{\langle \sigma', \alpha''\rangle\} + I), rm(\delta, \mathsf{m}, \tilde{\mathsf{v}})\rangle}\ (C\text{-}ReceiveMi)$$

$$\frac{\iota(\mathsf{p}) = \{\langle \sigma, \alpha\rangle\} \qquad \langle P, \sigma, \alpha\rangle \xrightarrow{!\mathsf{m} : \tilde{\mathsf{v}}} \langle \sigma', \alpha'\rangle}{\langle \mathsf{pool}(\mathsf{p}, P), \iota, \delta\rangle \xrightarrow{!\mathsf{m} : \tilde{\mathsf{v}}} \langle updI(\iota, \mathsf{p}, \{\langle \sigma', \alpha'\rangle\}), add(\delta, \mathsf{m}, \tilde{\mathsf{v}})\rangle}\ (C\text{-}Deliver)$$

$$\frac{\iota(\mathsf{p}) = \{\langle \sigma, \alpha\rangle\} + I \qquad \langle P, \sigma, \alpha\rangle \xrightarrow{!\mathsf{m} : \tilde{\mathsf{v}}} \langle \sigma', \alpha'\rangle}{\langle \mathsf{miPool}(\mathsf{p}, P), \iota, \delta\rangle \xrightarrow{!\mathsf{m} : \tilde{\mathsf{v}}} \langle updI(\iota, \mathsf{p}, \{\langle \sigma', \alpha'\rangle\} + I), add(\delta, \mathsf{m}, \tilde{\mathsf{v}})\rangle}\ (C\text{-}DeliverMi)$$

$$\frac{\langle C_1, \iota, \delta\rangle \xrightarrow{l} \langle \iota', \delta'\rangle}{\langle C_1 \parallel C_2, \iota, \delta\rangle \xrightarrow{l} \langle \iota', \delta'\rangle}\ (C\text{-}Int_1) \qquad \frac{\langle C_2, \iota, \delta\rangle \xrightarrow{l} \langle \iota', \delta'\rangle}{\langle C_1 \parallel C_2, \iota, \delta\rangle \xrightarrow{l} \langle \iota', \delta'\rangle}\ (C\text{-}Int_2)$$

Figure 7.8: BPMN Collaboration Semantics.

to a purely local perspective, where state functions are kept separate for each process.

7.3 Properties of BPMN Collaborations

In a data-aware setting, control flow needs to follow data flow, since otherwise the process would come to halt. Moreover, data flow is strongly related to message flow, especially in scenarios including multiple interacting participants. This section provides the definitions of the correctness properties considered in this thesis, by taking into account multi-instance data aware collaboration diagrams.

7.3.1 Well-Structured BPMN Collaborations

To classify BPMN models in multi-instance scenarios it is first necessary to extend the definition of well-structuredness to multi-instance collaborations including all the elements defined in the given semantics.

The formal characterisation of well-structured BPMN processes and collaborations relies on the usual functions $in(P)$ and $out(P)$, which determine the incoming and outgoing sequence edges of a process element P (the full definition is relegated to Appendix B). Moreover, the definition of well-structurendess already provided in the previous chapters naturally extends, by considering the syntactic representation the BPMN elements have in the new framework.

Definition 27 (Well-structured processes)**.** *A process P is* well-structured *(written $isWS(P)$) if P has one of the following forms:*

$$\mathsf{start}(\mathsf{e},\mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'',\mathsf{e}''') \tag{7.1}$$

$$\mathsf{start}(\mathsf{e},\mathsf{e}') \parallel P' \parallel \mathsf{terminate}(\mathsf{e}'') \tag{7.2}$$

$$\mathsf{start}(\mathsf{e},\mathsf{e}') \parallel P' \parallel \mathsf{endSnd}(\mathsf{e}'',\mathsf{m}:\tilde{\mathsf{exp}},\mathsf{e}''') \tag{7.3}$$

$$\mathsf{startRcv}(\mathsf{m}:\tilde{\mathsf{t}},\mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'',\mathsf{e}''') \tag{7.4}$$

$$\mathsf{startRcv}(\mathsf{m}:\tilde{\mathsf{t}},\mathsf{e}') \parallel P' \parallel \mathsf{terminate}(\mathsf{e}'') \tag{7.5}$$

$$\mathsf{startRcv}(\mathsf{m}:\tilde{\mathsf{t}},\mathsf{e}') \parallel P' \parallel \mathsf{endSnd}(\mathsf{e}'',\mathsf{m}:\tilde{\mathsf{exp}},\mathsf{e}''') \tag{7.6}$$

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

1. $isWSCore(\mathsf{task}(\mathsf{e},\mathsf{exp},A,\mathsf{e}'))$;
2. $isWSCore(\mathsf{taskRcv}(\mathsf{e},\mathsf{exp},A,\mathsf{m}:\tilde{\mathsf{t}},\mathsf{e}'))$;
3. $isWSCore(\mathsf{taskSnd}(\mathsf{e},\mathsf{exp},A,\mathsf{m}:\tilde{\mathsf{exp}},\mathsf{e}'))$;
4. $isWSCore(\mathsf{empty}(\mathsf{e},\mathsf{e}'))$;
5. $isWSCore(\mathsf{interRcv}(\mathsf{e},\mathsf{m}:\tilde{\mathsf{t}},\mathsf{e}'))$;
6. $isWSCore(\mathsf{interSnd}(\mathsf{e},\mathsf{m}:\tilde{\mathsf{exp}},\mathsf{e}'))$;

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

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

9. $$\frac{\forall j \in [1..n] \;\; isWSCore(P_j), \;\; in(P_j) = \mathsf{e}'_j, \;\; out(P_j) \subseteq E}{isWSCore(\mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j : \tilde{\mathsf{t}}_\mathsf{j}, \mathsf{e}'_j) | j \in [1..n]\}) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E, \mathsf{e}''))}$$

10. $$\frac{\begin{array}{c} isWSCore(P_1), isWSCore(P_2), \\ in(P_1) = \{\mathsf{e}'\}, out(P_1) = \{\mathsf{e}^{\mathsf{iv}}\}, in(P_2) = \{\mathsf{e}^{\mathsf{vi}}\}, out(P_2) = \{\mathsf{e}''\} \end{array}}{isWSCore(\mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{(\mathsf{e}^{\mathsf{v}}, \mathsf{exp}_1), (\mathsf{e}^{\mathsf{vi}}, \mathsf{exp}_2)\}))}$$

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

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

Definition 28 (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;*
- $isWS(\mathsf{miPool}(\mathsf{p}, P))$ *if P is well-structured;*
- $isWS(C_1 \parallel C_2)$ *if $isWS(C_1)$ and $isWS(C_2)$.*

Running Example 21. *Considering the proposed running example and according to the above definitions, processes P_r and P_{ca} are well-structured, while process P_{pc} is not well-structured, due to the presence of two XOR-Joins and only one XOR-Split. Thus, the overall collaboration is not well-structured.* □

7.3.2 Safe BPMN Collaborations

One of the relevant properties considered in the thesis is safeness. Also safeness can be defined in multi-instance collaborations scenario by requiring that each process instance is safe. Before providing a formal characterisation of safe BPMN processes and multi-instance collaborations, it is necessary to introduce the following definition determining the safeness of a process in a given state.

Definition 29 (Current state safe process)**.** *A process configuration* $\langle P, \sigma, \alpha \rangle$ *is current state safe (*cs-safe*) if and only if* $\forall \mathsf{e} \in edgesEl(P) \,.\, \sigma(\mathsf{e}) \leqslant 1$*.*

The definition of safe processes and multi-instance collaborations requires that cs-safeness is preserved along the computations. As usual, $\rightarrow^*$ denotes the reflexive and transitive closure of $\rightarrow$.

Definition 30 (Safe processes)**.** *A process* P *is* safe *if and only if, given* σ*,* α *such that* $isInit(P, \sigma, \alpha)$*, for all* σ' *such that* $\langle P, \sigma, \alpha \rangle \rightarrow^* \langle \sigma', \alpha' \rangle$ *we have that* $\langle P, \sigma', \alpha' \rangle$ *is cs-safe.*

Definition 31 (Safe collaborations)**.** *A collaboration* C *is* safe *if and only if, given* ι *and* δ *such that* $isInit(C, \iota, \delta)$*, for all for all* ι' *and* δ' *such that* $\langle C, \iota, \delta \rangle \rightarrow^* \langle \iota', \delta' \rangle$

- $\forall\, \mathsf{pool}(\mathsf{p}, P)$ *in* C*, given* $\iota'(\mathsf{p}) = \{\langle \sigma, \alpha \rangle\}$*, we have that* $\langle P, \sigma, \alpha \rangle$ *is cs-safe;*
- $\forall\, \mathsf{miPool}(\mathsf{p}, P)$ *in* C*, given* $\{\langle \sigma, \alpha \rangle\} \in \iota'(\mathsf{p})$*, we have that* $\langle P, \sigma, \alpha \rangle$ *is cs-safe.*

Running Example 22. *Considering again the running example depicted in Figure 7.1, all processes are safe since there are no fragment capable of generating multiple tokens. Also the overall collaboration is safe, since each of its process instances is safe.* □

7.3.3 Sound BPMN Collaborations

Regarding the soundness notion, the definition provided in Chapter 4 is referred. According to that definition, soundness informally means that if a process starts its execution with a token in the start event, it can always complete successfully. This latter requires that either there is a marking where all marked end events are marked exactly by a single token (and no token is around) or that no token is observed in the configuration. Thus, this definition is adapted here to the new framework, by requiring that each process instance is sound.

Definition 32 (Current state sound process)**.** *A process configuration* $\langle P, \sigma, \alpha \rangle$ *is current state sound (*cs-sound*) if and only if one of the following hold:*

(i) $\forall \mathsf{e} \in marked(\sigma, end(P)) \,.\, \sigma(\mathsf{e}) = 1$ *and* $\forall \mathsf{e} \in edges(P) \backslash end(P) \,.\, \sigma(\mathsf{e}) = 0$.

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

Recall, function $marked(\sigma, E)$ refers to the set of edges in E with at least one token; $edges(P)$ refers to the edges in the scope of P and function $end(P)$ refers to end events in the considered process.

Definition 33 (Sound process)**.** *A process P is* sound *if and only if, given σ α such that $isInit(P, \sigma, \alpha)$, for all σ' such that $\langle P, \sigma, \alpha\rangle \rightarrow^* \langle \sigma', \alpha'\rangle$ we have that there exist σ'', α'' such that $\langle P, \sigma', \alpha'\rangle \rightarrow^* \langle \sigma'' \alpha''\rangle$, and $\langle P, \sigma'', \alpha''\rangle$ is cs-sound.*

Definition 34 (Sound collaboration)**.** *A collaboration C is* sound *if and only if, given ι and δ such that $isInit(C, \iota, \delta)$, for all ι' and δ' such that $\langle C, \iota, \delta\rangle \rightarrow^* \langle \iota', \delta'\rangle$ we have that there exist ι'' and δ'' such that $\langle C, \iota', \delta'\rangle \rightarrow^* \langle \iota'', \delta''\rangle$ and*

- *$\forall\, \mathsf{pool}(\mathsf{p}, P)$ in C, given $\iota''(\mathsf{p}) = \{\langle \sigma, \alpha\rangle\}$, we have that $\langle P, \sigma, \alpha\rangle$ is cs-sound;*
- *$\forall\, \mathsf{miPool}(\mathsf{p}, P)$ in C, given $\{\langle \sigma, \alpha\rangle\} \in \iota''(\mathsf{p})$, we have that $\langle P, \sigma, \alpha\rangle$ is cs-sound.*

and $\forall\, \mathsf{m} \in \mathbb{M}\ .\ \delta''(\mathsf{m}) = 0$.

Now, the soundness notion can be relaxed by defining message-relaxed soundness, where no conditions are imposed on messages.

Definition 35 (Message-relaxed sound collaboration)**.** *A collaboration C is* message-relaxed sound *if and only if, given ι and δ such that $isInit(C, \iota, \delta)$, for all ι' and δ' such that $\langle C, \iota, \delta\rangle \rightarrow^* \langle \iota', \delta'\rangle$ we have that there exist ι'' and δ'' such that $\langle C, \iota', \delta'\rangle \rightarrow^* \langle \iota'', \delta''\rangle$ and*

- *$\forall\, \mathsf{pool}(\mathsf{p}, P)$ in C, given $\iota''(\mathsf{p}) = \{\langle \sigma, \alpha\rangle\}$, we have that $\langle P, \sigma, \alpha\rangle$ is cs-sound;*
- *$\forall\, \mathsf{miPool}(\mathsf{p}, P)$ in C, given $\{\langle \sigma, \alpha\rangle\} \in \iota''(\mathsf{p})$ we have that $\langle P, \sigma, \alpha\rangle$ is cs-sound.*

Running Example 23. *Considering the running example in Figure 7.1, according to the above definitions, all processes are sound. Also the overall collaboration is sound, since each of its process instances is sound.* □

7.4 Classification Results

Traditionally correctness verification considered only the control flow perspective, thus only considering the ordering relations among the activities present in the model. The capability of the proposed framework to characterize the interplay between control features, message exchanges and data allows

to formally define correctness properties in this extended complex framework. This permits to study the relationships between the studied properties. It results that the classification results obtained for the core subset of BPMN elements are still valid in the extended framework.

7.4.1 Well-structuredness vs. Safeness in BPMN

Considering well-structuredness and safeness it holds that all well-structured models are safe (Theorem 16), and that the reverse does not hold. To this aim, first it is shown that a process in the initial state is cs-safe (Lemma 8). Then, it is shown that cs-safeness is preserved by the evolution of well-structured core process elements (Lemma 9) and processes (Lemma 10). 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). This last notion, in its own turn, needs the definition of initial state for a core process element (see Appendix B).

Definition 36 (Reachable processes)**.** *A process configuration* $\langle P, \sigma, \alpha \rangle$ *is* reachable *if there exists a configuration* $\langle P, \sigma', \alpha' \rangle$ *such that* $isInit(P, \sigma', \alpha')$ *and* $\langle P, \sigma', \alpha' \rangle \rightarrow^* \langle \sigma, \alpha \rangle$.

Definition 37 (Reachable core process element)**.** *A process configuration* $\langle P, \sigma, \alpha \rangle$ *is* core reachable *if there exists a configuration* $\langle P, \sigma', \alpha' \rangle$ *such that* $isInitEl(P, \sigma', \alpha')$ *and* $\langle P, \sigma', \alpha' \rangle \rightarrow^* \langle \sigma, \alpha \rangle$.

Lemma 8. *Let* P *be a process, if* $isInit(P, \sigma, \alpha)$ *then* $\langle P, \sigma, \alpha \rangle$ *is cs-safe.*

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

Lemma 9. *Let* $isWSCore(P)$*, and let* $\langle P, \sigma, \alpha \rangle$ *be a core reachable and cs-safe process configuration, if* $\langle P, \sigma, \alpha \rangle \xrightarrow{\ell} \langle \sigma', \alpha' \rangle$ *then* $\langle P, \sigma', \alpha' \rangle$ *is cs-safe.*

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

Lemma 10. *Let* $isWS(P)$ *and let* $\langle P, \sigma, \alpha \rangle$ *be a process configuration reachable and cs-safe, if* $\langle P, \sigma, \alpha \rangle \xrightarrow{\ell} \langle \sigma', \alpha' \rangle$ *then* $\langle P, \sigma', \alpha' \rangle$ *is cs-safe.*

Proof (sketch). By case analysis on the structure of P, which is a WS process. □

Theorem 16. *Let* P *be a process, if* P *is well-structured then* P *is safe.*

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

The reverse implication of Theorem 16 is still not true. In fact there are safe processes that are not well-structured. To prove this, it is sufficient to consider the PC Chair process in the running example.

Now the previous results can be extended to collaborations.

Theorem 17. *Let C be a collaboration, if C is well-structured then C is safe.*

Proof (sketch). By contradiction. □

7.4.2 Well-structuredness vs. Soundness in BPMN

Considering the relationship between well-structuredness and soundness it results that a well-structured process is always sound (Theorem 18), but there are sound processes that are not well-structured. To prove that, first it is shown that a reachable well-structured core process element can always complete its execution (Lemma 11). This latter Lemma 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 (the definition is relegated to Appendix B).

Lemma 11. *Let $isWSCore(P)$ and let $\langle P, \sigma, \alpha \rangle$ be core reachable, then there exist σ', α' such that $\langle P, \sigma, \alpha \rangle \rightarrow^* \langle \sigma', \alpha' \rangle$ and $isCompleteEl(P, \sigma', \alpha')$.*

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

Theorem 18. *Let $isWS(P)$, then P is sound.*

Proof (sketch). By case analysis. □

The reverse implication of Theorem 18 is not true. In fact there are sound processes that are not well-structured. Moreover, it does not extend to the collaboration level. In fact, putting well-structured processes together in a collaboration, this could be either sound or unsound. This is also valid for message-relaxed soundness.

Theorem 19. *Let C be a collaboration, $isWS(C)$ does not imply C is sound.*

Proof (sketch). By contradiction. □

Theorem 20. *Let C be a collaboration, is$WS(C)$ does not imply C is message-relaxed sound.*

Proof (sketch). By contradiction. □

7.4.3 Safeness vs. Soundness in BPMN

Considering the relationship between safeness and soundness it results that there are unsafe models that are sound. Considering the process level the following Theorem holds.

Theorem 21. *Let P be a process, P is unsafe does not imply P is unsound.*

Proof (sketch). By contradiction. □

There are also unsafe collaborations that could be either sound or unsound, as proved by the following Theorem.

Theorem 22. *Let C be a collaboration, C is unsafe does not imply C is unsound.*

Proof (sketch). By contradiction. □

7.5 Compositionality of Safeness and Soundness

This section studies the compositionality of safeness and soundness, i.e. how the behaviour of processes affects that of the entire resulting collaboration. It comes out that the compositionality results obtained for the core subset of BPMN elements are still valid in the extended framework.

7.5.1 On Compositionality of Safeness

Here it is shown that safeness is compositional, that is the composition of safe processes always results in a safe collaboration.

Theorem 23. *Let C be a collaboration, if all processes in C are safe then C is safe.*

Proof (sketch). By contradiction (see Appendix C.4). □

Considering the same examples shown in Section 4.6.1 it results 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.

7.5.2 On Compositionality of Soundness

As well as for the safeness property, it is shown now that it is not feasible to detect the soundness of a collaboration by relying only on information about the soundness of the processes that compose it. However, the unsoundness of processes implies the unsoundness of the resulting collaboration. This result confirms the study already done for the core BPMN elements (Section 4.6.2).

Theorem 24. *Let C be a collaboration, if some processes in C are unsound then C is unsound.*

Proof (sketch). By contradiction (see Appendix C.4). □

On the other hand, as already shown for the core BPMN elements, putting together sound processes, the obtained collaboration could be either sound or unsound, since one has also to consider messages.

7.6 MIDA Animator

Besides being useful per se, as it provides a precise understanding of the ambiguous and loose points of the standard, a main benefit of the provided formalisation is that it paves the way for the development of tools supporting model debugging. In fact, designers are currently not assisted in figuring out the (possibly complex) behaviour of multi-instance collaborations, and just looking at their static descriptions is in general an error-prone task. This issue is clarified by resorting to the multi-instance collaboration model presented in Section 7.1. To model correctly this collaboration, the diagram must contain enough information to correlate each message with the appropriate instance. For example, if messages are not properly delivered, the feedback messages could not be properly delivered to the right *Reviewer*.

To fill this gap, a novel animator tool called MIDA (*Multiple Instances and Data Animator*) is here presented. It faithfully implements the proposed formal semantics and visualises the execution of multi-instance collaborations. Model animation plays an important role to address the issues that may arise when modelling multi-instance collaborations, such as the intricate management of the correlation mechanism or the handling of data

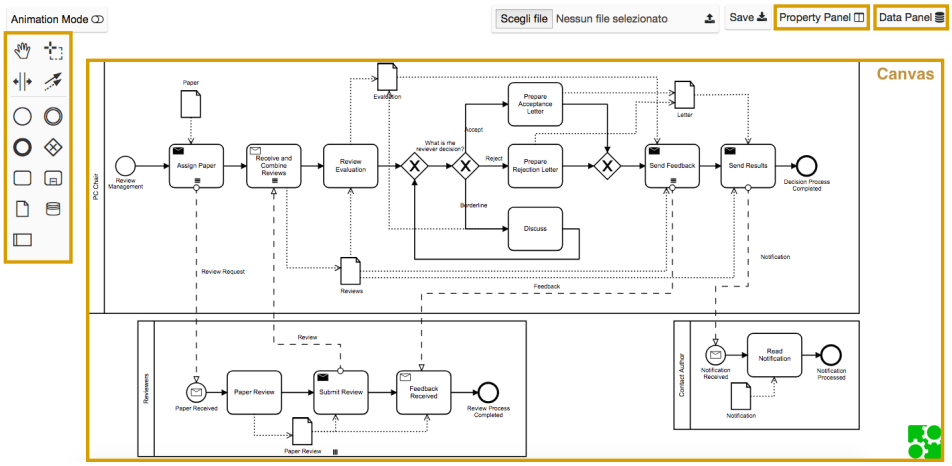


Figure 7.9: MIDA Web Interface.

and messages. Animation can enhance the understanding of business processes behaviour [41, 28], especially in presence of a faithful correspondence with a precise semantics [12]. Visualisation of model execution via an animator allows to understand the collaboration history, its current state (also in terms of data-object values) and possible future executions [61]. In the literature, few relevant contributions have been proposed: the animator by Allweyer and Schweitzer [6], and those developed by Signavio and Visual Paradigm. However, these tools do not support the interplay between multiple instances, messages and data, hence they do not allow designers to deal with the mentioned issues.

MIDA supports designers in achieving a more precise understanding of the behaviour of a collaboration by means of the visualisation of the model execution, also in terms of the values evolution of data objects and messages. MIDA animation features result helpful both in *educational contexts*, for explaining the behaviour of BPMN elements, and in practical modelling activities, for *debugging* errors that can easily arise in multi-instance collaborations.

MIDA is a web application written in JavaScript, realised by extending the Camunda *bpmn.io* token simulation plug-in [70]. As shown in Figure 7.9, the graphical interface of the tool consists of four main parts: *(i)* the canvas, where BPMN elements are composed to form a collaboration diagram; *(ii)* the palette, to insert elements in the diagram; *(iii)* the property panel, to specify attributes of the BPMN elements of the diagram; and *(iv)* the data panel, to visualise data object values. MIDA permits to locally save models in the standard format *.bpmn* and, hence, to load models previously designed.

The property panel plays a key role when modelling collaborations with MIDA, as it permits exploiting *.bpmn* XML attributes to model and save information about multi-instance characteristics, data objects and related fields, and messages. This information is written using the JavaScript syntax.

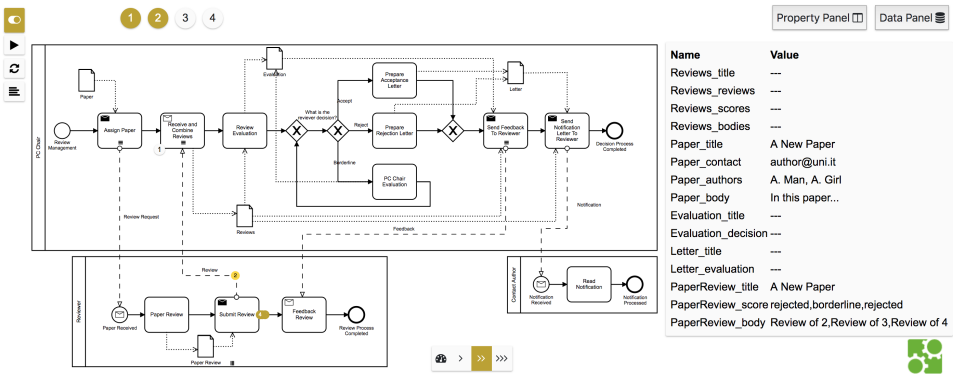


Figure 7.10: MIDA Animation.

Both pools and activities can be set as multi-instance. In the former case, a double click on the pool element opens a pop-up window that allows to specify the pool as multi-instance and to constrain the number of instances that will be executed for that pool. In the latter case, multi-instance activities are defined by selecting the corresponding marker (||| or ≡) in the element context pad and by filling the *loopCardinality* attribute with the number of activity instances to execute.

Data objects are structured in terms of fields, which are rendered in MIDA as JavaScript variables that can be initialised or left undefined. According to the BPMN standard, the access to data is represented by associations between data objects and activities. These associations can define preconditions for the execution of an activity, expressed in MIDA as an activity *guard*. The effects on data objects of an activity execution is instead specified by means of a list of *assignments*. In case of send tasks, assignments can be used also to fill message fields while, in case of receive tasks, guards also specifies correlation conditions.

The key characteristic of MIDA is the animation of collaboration models, supporting in particular the visualisation of data evolution and multiple instances execution. At any time, the animation can be paused by the user to check the distribution of tokens and the current state of data. From a practical point of view, this allows designers to debug their collaboration models. They can indeed detect undesired executions, where e.g. a control flow is blocked, and deduce the cause beyond them by possibly checking the values of the involved data. These debugging facilities are particularly useful in case of multi-instance collaborations, where data values are used to regulate the correlation of messages with instances. Anyway, like in software code debugging, the identification and fixing of bugs are still in charge of the human user.

By selecting the *Animation Mode* in the MIDA interface, a *play* button will appear over each fireable start event. Once this button is clicked, one or more instances of the desired process are activated, depending on the

multi-instance information specified in the model. This creates a new token labelled by a fresh instance identifier. Then, as shown in Figure 7.10, the token starts to cross the model according to the operational rules induced by the formal semantics.

The animation terminates once all tokens cannot move forward. In case a token remains blocked due to a data handling issue, e.g. a wrong correlation or a guard condition violation, MIDA highlights it in red as in Figure 7.11. Another example is the following: supposing that the *Discuss* task in *PC Chair* would not be in a loop, but it would have its outgoing edge directly connected to the XOR-Join in its right hand side, then after the execution of the *Discuss* task, the task *Send Feedback* would be performed, and the task *Send Results* would be activated. However, the guard of the latter task would not be satisfied, because the *Letter* data object would not be properly instantiated. This would cause a deadlock, which can be found out by using MIDA.

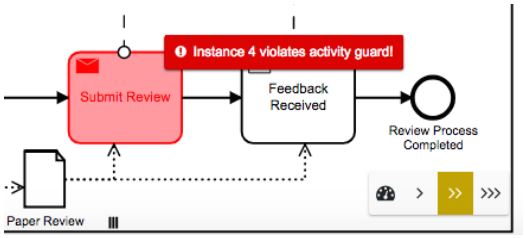


Figure 7.11: Guard Violation.

Finally, in the context of multi-instance collaborations, the choice of correlation data is an error-prone task that is a burden on the shoulders of the designers. By resorting to the example in Figure 7.1 it is possible to see how issues related to instance correlation can be detected. Considering the *Reviewer* participant in the running scenario if the template within the task for receiving the feedback would not properly specify the correlation data (e.g., $\tilde{t_3} = \langle ?\mathsf{Feedback.reviewerName}, ?\mathsf{Feedback.title}, ?\mathsf{Feedback.evaluation} \rangle$), the feedback messages could not be properly delivered. Indeed, each *Reviewer* instance would be able to match any feedback message, regardless the reviewer name and the paper title specified in the message. Thus, the feedback messages could be mixed up. Fortunately, MIDA allows to detect, and hence solve, this correlation issue. Like in code debugging, the identification of the bug is still in charge of the human user. Similarly, MIDA helps designers to detect issues concerning the exchange of messages. In fact, malformed or unexpected messages may introduce deadlocks in the execution flow, which can be easily identified by looking for blocked tokens in the animation. For instance, in the running example a feedback message without the evaluation field would be never consumed by a receiving task of the *Reviewer* instances.

The MIDA tool, as well as its source code, examples, user guide and screencast, are available at http://pros.unicam.it/mida/. In particular, the screencast shows a typical scenario where the user requires to model, animate, and debug a BPMN model. MIDA can be redistributed

7.7 Comparison with other Approaches

Recently many approaches have been proposed to formalise and verify data-aware processes. In the following these works are compared to the contribution of the thesis. In particular, first the available formalisations are discussed and then an investigation on correctness properties is reported.

7.7.1 Multiple Instances and Data Formalisations

This section discusses the most relevant attempts in formalising multiple instances and data for BPMN models.

Many works in the literature attempted to formalise the core features of BPMN. However, most of them (see, e.g., [29, 24, 112, 14, 105]) do not consider multiple instances and data. Considering these features in BPMN collaborations, relevant works are [59, 60, 43, 36]. Meyer et al. in [59] focus on process models where data objects are shared entities and the correlation mechanism is used to distinguish and refer data object instances. Use of data objects local to (multiple) instances, exchange of messages between participants, and correlation of messages are instead the focus of this thesis. In [60], the authors describe a model-driven approach for BPMN to include the data perspective. Differently from the propsed approach, they do not provide a formal semantics for BPMN multiple instances. Moreover, they do not use data in decision gateways. Moreover, Kheldoun et al. propose in [43] a formal semantics of BPMN covering features such as message-exchange, cancellation, multiple instantiation of sub-processes and exception handling, while taking into account data flow aspects. However, they do not consider multi-instance pools and do not address the correlation issue. Semantics of data objects and their use in decision gateways is instead proposed by El-Saber and Boronat in [36]. Differently from this thesis work, this formal treatment does not include collaborations and, hence, exchange of messages and multiple instances. Considering other modelling languages, YAWL [111] and high-level Petri nets [103] provide direct support for the multiple instance patterns. However, they lack support for handling data. In both cases, process instances are characterised by their identities, rather than by the values of their data, which are however necessary to correlate messages to running instances. Regarding choreographies, relevant works are [52, 45, 40]. López et al. [52] study the choreography problem derived from the synchronisation of multiple instances necessary for the management of data dependencies. Knuplesch et al. [45] introduces a data-aware collaboration approach including formal correctness criteria. However, they define the

data perspective using data-aware interaction nets, a proprietary notation, instead of the wider accepted BPMN. Improving data-awareness and data-related capabilities for choreographies is the goal of Hahn et al. [40]. They propose a way to unify the data flow across participants with the data flow inside a participant. The scope of data objects is global to the overall choreography, while here data objects have scope local to participant instances, as prescribed by the BPMN standard. Apart from the specific differences mentioned above, this work differs from the others for the focus on collaboration diagrams, rather than on choreographies. This allows to specifically deal with multiple process instantiation and messages correlation.

Finally, concerning the correlation mechanism, the BPMN standard and, hence, the thesis have been mainly inspired by works in the area of service-oriented computing (see the relationship between BPMN and WS-BPEL [67] in [68, Sec. 14.1.2]). In fact, when a service engages in multiple interactions, it is generally required to create an instance to concurrently serve each request, and correlate subsequent incoming messages to the created instances. Among the others, the COWS [77] formalism captures the basic aspects of SOC systems, and in particular service instantiation and message correlation à la WS-BPEL. From the formal point of view, correlation is realised by means of a pattern-matching function similar to that used in the provided formal semantics.

7.7.2 Reasoning on Correctness Properties

Many approaches have been proposed to formalise and verify data-aware processes, also considering data-aware extensions of soundness [62]. Concerning data-aware soundness, relevant work are [89, 11]. Sidorova et al. [89] present classical workflow nets extended with data operations as a conceptual workflow model. They develop a technique to verify may and must soundness, namely that either at least one or any refinement of a conceptual workflow model is sound. Soundness is defined according to the classical definition given for workflow nets [104], thus not considering BPMN peculiarities. Decision-aware soundness is formally defined by Batoulis and Weske in [11], where BPMN is integrated with the Decision Model and Notation (DMN). However, the contribution of decisions in the verification of soundness is local, i.e. it is limited to the interaction between decisions and their immediate outgoing sequence flows. Moreover, only the process level is taken into account. Differently, Knuplesch et al. [45] introduce a data-aware collaboration approach including formal correctness criteria. However, they define the data perspective using data-aware interaction nets, a proprietary notation, instead of the wider accepted BPMN. In particular, they focus on interaction models, that in BPMN correspond to choreography diagrams. The flow of message exchanges is specified without having any knowledge

about the partner processes, thus data exchanged via messages cannot be used within processes by decision gateways. This does not allow to capture some incorrect behaviour due to the effects of messages on data and to the correlation mechanism. Data issues are instead considered in [82, 58]. Rachdi et al. [82] propose a formal data-flow analysis of BPMN models based on Data Record concept. Their approach allows to detect the anti-patterns representing data-flow anomalies, explaining to the designer the origin of the anomaly so he/she can fix it easily in remodelling phase. However, they focus only on data, without considering other relevant properties that are instead connected to the data perspective.

Chapter 8

Formalising BPMN Patterns

The effective and efficient handling of business processes is a primary goal of organisations. For conducting a successful business, an organisation does not act alone, but it is usually involved in collaborations with other organisations. Thus, choreography modelling languages have emerged in the past years as a means for capturing and managing collaborative processes, and different evaluations of the expressive power of these languages have been made [94, 15]. In this regard, since its introduction, the Workflow Patterns (www.workflowpatterns.com) framework was specifically tailored for the purpose of process language analysis. This framework consists of patterns spanning the control-flow, the data and the resource perspective, and was gradually developed to incorporate also interactions among different participants. The importance of interactions has been underlined by many authors [102, 25, 9] and a lot of effort has been done to identify the most common interaction scenarios from a business perspective, which have been called *Service Interaction Patterns* [10]. Inter-organisational business relationships are considered as a first-class citizen in BPMN collaboration diagrams, where multiple participants interact via messages. This motivated the use of BPMN to model service interaction patterns [109], initially defined only in terms of textual descriptions. This effort provided a graphical, more intuitive, description of the patterns and allowed to assess the suitability of BPMN to express common interaction scenarios. This BPMN evaluation has also been integrated with the analysis of BPMN in terms of control-flow, resource and data patterns [94]. However, a severe issue in these studies, is that the precise behaviour of the BPMN models corresponding to some patterns may result unclear, in particular when multiple instances of the interacting participants are involved. This problem is mainly due to the fact that the BPMN standard comes without a formal semantics, which is needed in presence of tricky features. Thus, a direct formal characterisation of these patterns is crucial, as it does not leave any room for ambiguity, and increases

the potential for formal reasoning.

This chapter focuses on the workflow patterns supported by BPMN, taking as reference the BPMN evaluation results in [94], and on the BPMN service interaction patterns [15], motivated by the BPMN capability to model collaboration diagrams.

The patterns are visualised in BPMN models and formalised by means of the provided direct formal semantics for BPMN. The formalisation allows to validate the semantics, both in terms of the considered BPMN elements and of the expected semantic behaviour. In particular, it results that the extension of the semantics allows an incremental support of the considered patterns.

Specifically, first those patterns that can be modelled only by means of the core BPMN elements are considered (Section 8.1). Then, they are extended by including patterns with OR gateways (Section 8.2), sub-processes (Section 8.3) and multiple instances (Section 8.4). Finally, the results of this semantics validation are presented (Section 8.5).

Each pattern will be presented according to the following structure:

Informal Description consists of a natural language description, and a graphical representation in terms of a BPMN (collaboration) fragment.

Textual Specification provides the textual notation of the BPMN (collaboration) model.

Formal Semantics describes the operational rules applied to perform each execution step, and shows the results in terms of the execution state functions evolution.

Highlights. The main contributions of this chapter are:

- it provides a comprehensive formalisation of BPMN patterns by means of the provided direct formal semantics for BPMN models;
- It validates the semantics.

8.1 Core BPMN Patterns

This section presents and formalises the workflow patterns that can be modelled with the core BPMN elements considered in Chapter 3. These patterns include most of the Control-flow Patterns [95] and Resource Patterns [88], and many of the Service Interaction Patterns [10]. The patterns are presented in the following order: first those patterns concerning the control-flow and resource perspective and then the interaction patterns.

Sequence Pattern.

Informal Description. An activity in a process is enabled after the completion of a preceding activity in the same process.

This pattern can be modelled in BPMN as a sequence flow between two activities as shown in Figure 8.1.

Textual Specification. The core BPMN formal framework supports this patterns that in the textual notation is rendered as follows:

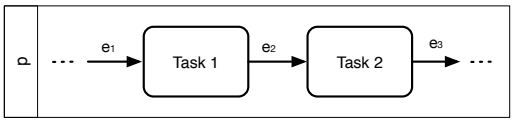


Figure 8.1: Sequence.

$$P = \mathsf{task}(\mathsf{e}_1, \mathsf{e}_2) \parallel \mathsf{task}(\mathsf{e}_2, \mathsf{e}_3) \parallel P'$$

Formal Semantics. According to the form of process P, and the current state σ, the process can evolve as follows:

- Process P moves by executing Task 1. This execution step takes place by applying rule *P-Task*, which requires the incoming edge e_1 of the task be marked by at least one token ($\sigma(\mathsf{e}_1) > 0$) The effects of the task execution are as follows: an internal action ϵ is performed, and the marking σ of the process instance is updated with the movement of one token from e_1 to e_2, that is $\sigma' = inc(dec(\sigma, \mathsf{e}_1), \mathsf{e}_2)$. Therefore, the application of rule *P-Task* produces the transition $\langle \mathsf{task}(\mathsf{e}_1, \mathsf{e}_2), \sigma \rangle \xrightarrow{\epsilon} inc(dec(\sigma, \mathsf{e}_1), \mathsf{e}_2)$ Hence, the overall process P can evolve according to the interleaving rule *P-Int₁*, that is $\langle P, \sigma \rangle \xrightarrow{\ell} \sigma'$. Now, Task 2 can evolve in the same way.

- Process P moves by executing an (unspecified) activity of P'. Thus, we have a transition $\langle P', \sigma \rangle \xrightarrow{\ell} \sigma'$, from which P can evolve by means of the symmetric rule of *P-Int₁*. This execution step, anyway, is not relevant for the pattern semantics, and hence is not discussed in more detail.

Parallel Split Pattern.

Informal Description. The divergence of a branch into two or more parallel branches each of which execute concurrently.

This pattern is captured in BPMN by an AND-Split gateway (Figure 8.2).

Textual Specification. In the textual notation the pattern is rendered as follows:

$$P = \mathsf{andSplit}(\mathsf{e}_1, \{\mathsf{e}_2, \mathsf{e}_3\}) \parallel P'$$

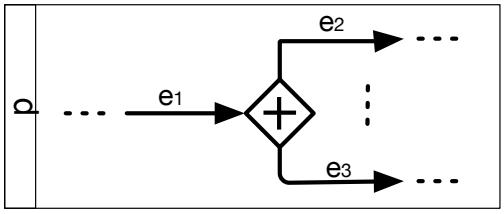


Figure 8.2: Parallel Split.

Formal Semantics. Assuming that the AND-Split gateway is enabled by a token in e_1, the process P can evolve, that is the transition $\langle \mathsf{andSplit}(\mathsf{e}, E), \sigma \rangle \xrightarrow{\epsilon} inc(dec(\sigma, \mathsf{e}_1), \{\mathsf{e}_2, \mathsf{e}_3\})$ is produced by applying rule *P-AndSplit*. Then, the process P evolves by means of the interleaving rule *P-Int*$_1$.

Synchronisation Pattern.

Informal Description. The convergence of two or more branches into a single subsequent branch such that the thread of control is passed to the subsequent branch when all input branches have been enabled.

This pattern is captured in BPMN by an AND-Join gateway (Figure 8.3).

Textual Specification. In the textual notation this pattern is rendered as follows:

$$P = \mathsf{andJoin}(\{\mathsf{e}_1, \mathsf{e}_2\}, \mathsf{e}_3) \parallel P'$$

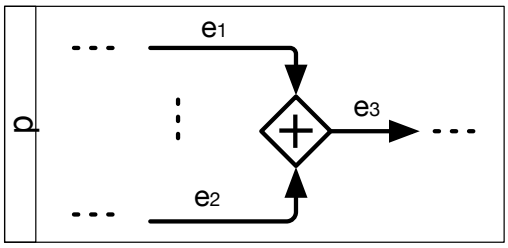


Figure 8.3: Synchronisation.

Formal Semantics. Similar to the previous pattern, the formal semantics of this pattern is determined by the application of rule for AND-Join gateway as described in Section 3.2.2.

Exclusive Choice Pattern.

Informal Description. The divergence of a branch into two or more branches, where one of the several branches is chosen.

This pattern is captured in BPMN by an XOR-Split gateway (Figure 8.4).

Textual Specification. In the textual notation the process fragment in Figure 8.4 is as follows:

$$P = \mathsf{xorSplit}(\mathsf{e}_1, \{\mathsf{e}_2, \mathsf{e}_3\}) \parallel P'$$

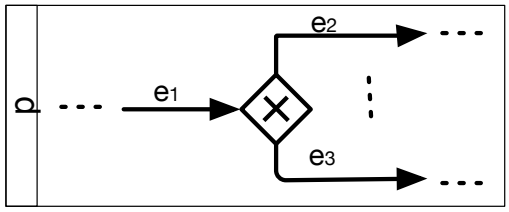


Figure 8.4: Exclusive Choice.

Formal Semantics. The formal semantics of this pattern is determined by the application of the rule *P-XorSplit*.

Simple Merge Pattern.

Informal Description. The convergence of two or more branches into a single subsequent branch without synchronization.

This pattern is captured in BPMN by an XOR-Join gateway (Figure 8.5), that expresses also the similar *Multiple Merge* pattern [94].

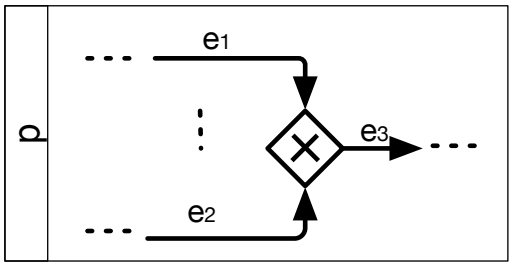


Figure 8.5: Simple Merge.

Textual Specification. This pattern can be represented using the textual notation given for the core BPMN elements as follows:

$$P = \mathsf{xorJoin}(\{\mathsf{e}_1, \mathsf{e}_2\}, \mathsf{e}_3) \parallel P'$$

Formal Semantics. Again, the formal semantics of this pattern is determined by the application of rule for XOR-Join gateway already described in Section 3.2.2.

Arbitrary Cycles Pattern.

Informal Description. The ability to represent cycles in a process model that have more than one entry or exit point.

Since the proposed BPMN formalisation does not impose any restriction on cyclic models, this pattern is fully supported by the given semantics.

Deferred Choice Pattern.

Informal Description. A divergence point in a process where one of several possible branches should be activated. The actual decision on which branch is activated is made by the environment and is deferred to the latest possible moment.

This pattern can be captured in BPMN through the Event-based gateway (Figure 8.6).

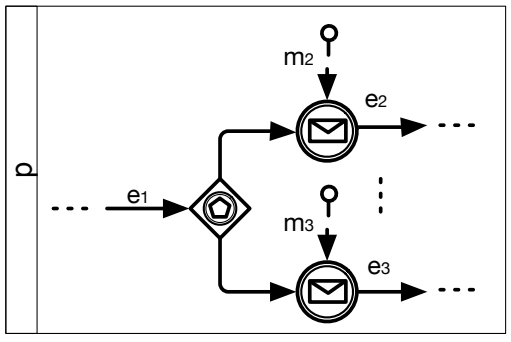


Figure 8.6: Deferred Choice.

Textual Specification. The given pattern can be represented using the core BPMN textual representation as follows:

$$P = \mathsf{eventBased}(\mathsf{e}_1, (\mathsf{m}_2, \mathsf{e}_2), (\mathsf{m}_3, \mathsf{e}_3)) \parallel P'$$

Formal Semantics. The execution steps of this pattern are determined by the application of rule for Event-based gateway already described in Section 3.2.2.

Cancel Case Pattern.

Informal Description. The cancellation of an entire process instance (i.e. all activities relating to the process instance).

A way to capture the cancel case pattern in BPMN is through the terminate event as shown in Figure 8.7.

Textual Specification. This pattern is expressed in the textual notation as follows:

$$P = \mathsf{terminate}(\mathsf{e}_n) \parallel P'$$

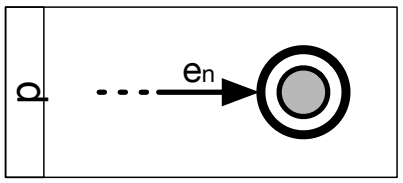


Figure 8.7: Cancel Case.

Formal Semantics. The semantics of this pattern is simply realised by applying the rule for the terminate event presented in Section 3.2.2. In detail, assuming that there is a token in the incoming edge of the terminate event $\sigma(\mathsf{e}_n) > 0$, the application rule *P-Terminate* produces the following transition $\langle \mathsf{terminate}(\mathsf{e}), \sigma \rangle \xrightarrow{kill} dec(\sigma, \mathsf{e})$. Then, the overall process P can evolve according to the killing rule *P-Kill*$_1$ that propagates the killing action in the scope of P.

Direct Allocation Pattern.

Informal Description. The ability to specify at design time the identity of the resource to which instances of a task will be distributed at runtime.

This pattern is supported by BPMN through the notion of pool, as it can be used to denote a specific business entity, as shown in Figure 8.8.

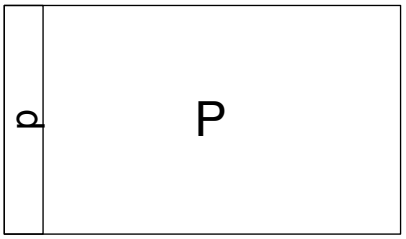


Figure 8.8: Direct Allocation.

Textual Specification. This pattern is expressed in the textual notation as follows:

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

Formal Semantics. The semantics of the pattern is simply given in terms of the semantics of the process P included in the pool.

Role-Based Allocation Pattern.

Informal Description. The ability to specify at design-time one or more roles to which instances of a task will be distributed at runtime.

Also this pattern is supported by BPMN through the notion of pool.

Automatic Execution Pattern. The ability for an instance of a task to execute without needing to utilise the services of a resource.

Since the resource allocation for the different activities is not necessarily done during design time, this pattern can be rendered by a simple BPMN process, without a partitioning of the process into pools.

Commencement on Creation Pattern.

Informal Description. The ability for a resource to commence execution on a work item as soon as it is created.

This pattern can be modelled as a BPMN task, as shown in Figure 8.9.

Textual Specification. The BPMN process fragment in Figure 8.9 is expressed in the textual notation as follows:

$$P = \mathsf{task}(\mathsf{e}_1, \mathsf{e}_2) \parallel P'$$

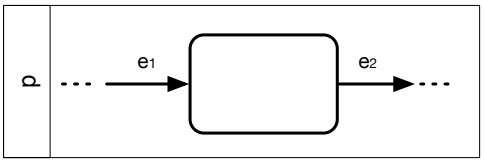


Figure 8.9: Commencement on Creation.

Formal Semantics. The semantics of the patterns is realised by applying the rule for the task execution, *P-Task*, that, as prescribed by the pattern description, requires that the incoming edge e_1 of the task is marked by at least one token ($\sigma(\mathsf{e}_1) > 0$).

Chained Execution Pattern. The ability to automatically start the next work item once the previous one has completed.

Also this pattern can be modelled in BPMN via a task and is fully supported by the core BPMN semantics. In fact, once a task execution is completed, subsequent activity(ies) receive a sequence token and are triggered

immediately when the specified condition on the required number of tokens is reached.

Send Pattern.

Informal Description. A party sends a message to another party.

This pattern can be modelled as the BPMN collaboration fragment shown in Figure 8.10. Notably, this is only a way to model it: the send task could in fact be replaced by an intermediate send event or by a message end event. However, up to some technicalities, all cases behave in the same way, thus here only one of them is reported

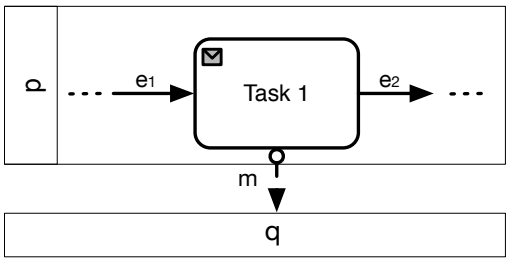


Figure 8.10: Send.

Textual Specification. The collaboration fragment in Figure 8.10 is represented in the textual notation as

$$C \;=\; \mathsf{pool}(\mathsf{p}, P) \parallel \mathsf{pool}(\mathsf{q}, Q)$$

with

$$P = \mathsf{taskSnd}(\mathsf{e}_1, \mathsf{m}, \mathsf{e}_2) \parallel P'$$

where p is the sender and q a generic receiver (represented by a black-box pool in the graphical notation, whose process Q is left unspecified in the textual one).

Formal Semantics. According to the form of process P, and the current state $\langle \sigma, \delta \rangle$ of pool p's instance, the collaboration can evolve as follows:

- Process P moves by executing Task 1. This execution step takes place by applying rule *P-TaskSnd*, which requires the incoming edge e_1 of the task be marked by at least one token ($\sigma(\mathsf{e}_1) > 0$). The effects of the task execution are as follows: the message action $!\mathsf{m}$ is produced, and the marking σ of the process instance is updated with the movement of one token from e_1 to e_2, that is $\sigma' = inc(dec(\sigma, \mathsf{e}_1), \mathsf{e}_2)$. Therefore, the application of rule *P-TaskSnd* produces the transition $\langle \mathsf{taskSnd}(\mathsf{e}_1, \mathsf{m}, \mathsf{e}_2), \sigma \rangle \xrightarrow{!\mathsf{m}} \sigma'$. Hence, the overall process P can evolve according to the interleaving rule *P-Int*$_1$, that is $\langle P, \sigma \rangle \xrightarrow{!\mathsf{m}} \sigma'$. Similarly, by applying the rule *C-Deliver*, and then the interleaving rule at collaboration level, the execution step of the overall collaboration C is represented by the transition $\langle C, \delta \rangle \xrightarrow{!\mathsf{m}} \langle \sigma', inc(\delta, \mathsf{m}) \rangle$. Its effects are:

updating the message state function ($inc(\delta, \mathsf{m})$) by adding a message to the m's message list, in order to be subsequently consumed by the receiving participant q.

- Process P moves by executing an (unspecified) activity of P'. Thus, there is a transition $\langle P', \sigma \rangle \xrightarrow{\ell} \sigma'$, from which P can evolve by means of the symmetric rule of *P-Int₁*, and the overall collaboration can then evolve accordingly. This execution step, anyway, is not relevant for the pattern semantics, and hence is not discussed in more detail.
- Process Q moves by executing an (unspecified) activity. Again this execution step is not relevant for the pattern semantics.

Relying on asynchronous communication, it is possible to formalise an unreliable and non-guaranteed delivery. The sending action in fact just updates the message state function by adding a message, without requiring this to be received.

Receive Pattern.

Informal Description. A party receives a message from another party.

This pattern can be modelled as the BPMN collaboration fragment shown in Figure 8.11. Also here the intermediate receive event could be replaced, in this case by a receive task or by a receiving start event.

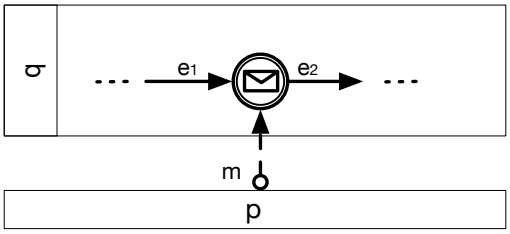


Figure 8.11: Receive.

Textual Specification. The textual representation of the collaboration fragment in Figure 8.11 has again the form C of the previous pattern, with

$$Q = \mathsf{interRcv}(\mathsf{e}_1, \mathsf{m}, \mathsf{e}_2) \parallel Q'$$

Formal Semantics. Assuming that the intermediate receive event is enabled by a token in e_1, the process Q can perform a receiving action, that is the transition $\langle \mathsf{interRcv}(\mathsf{e}_1, \mathsf{m}, \mathsf{e}_2), \sigma \rangle \xrightarrow{?\mathsf{m}} inc(dec(\sigma, \mathsf{e}_1), \mathsf{e}_2)$ is produced by applying rule *P-InterRcv*. Then, the process Q evolves by means of the interleaving rule *P-Int₁*. The produced label $?\mathsf{m}$ indicates the willingness of process Q to consume a message of type m. If present, the message is actually consumed by rule *C-Receive* at collaboration level. Indeed, this rule requires that there is a message in the m's message queue ($\mathsf{m} \in \delta(\mathsf{m})$), that is then removed.

Send/Receive Pattern.

Informal Description. Two parties, p and q, engage in two causally related interactions. In the first interaction, p sends a message (the request) to q, while in the second one p receives a message (the response) from q.

This pattern can be modelled by combining the *Send* and the *Receive* patterns, as shown in Figure 8.12.

Textual Specification. The textual representation of the collaboration fragment in Figure 8.12 has again the form C of the previous patterns, with

$$P = \mathsf{taskSnd}(\mathsf{e}_1, \mathsf{m}_1, \mathsf{e}_2) \parallel \mathsf{interRcv}(\mathsf{e}_2, \mathsf{m}_2, \mathsf{e}_3) \quad \parallel \quad P'$$
$$Q = \mathsf{interRcv}(\mathsf{e}_4, \mathsf{m}_1, \mathsf{e}_5) \parallel \mathsf{taskSnd}(\mathsf{e}_5, \mathsf{m}_2, \mathsf{e}_6) \quad \parallel \quad Q'$$

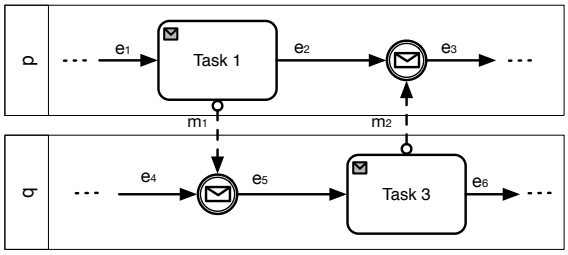


Figure 8.12: Send/Receive.

Formal Semantics. The execution steps of this pattern are realised by combining the semantic rules for the *Send* and *Receive* patterns. In detail: let us suppose that there is a token in the incoming edge of $\mathsf{Task\ 1}$ ($\sigma(\mathsf{e}_1) > 0$); by applying rule *P-TaskSnd* it follows that $\langle \mathsf{taskSnd}(\mathsf{e}_1, \mathsf{m}_1, \mathsf{e}_2), \sigma \rangle \xrightarrow{\mathsf{!m}} inc(dec(\sigma, \mathsf{e}_1), \mathsf{e}_2)$. Then, P evolves by performing a sending action, by means of the interleaving rule *P-Int$_1$*, that is $\langle P, \sigma, \alpha \rangle \xrightarrow{\mathsf{!m}} \sigma'$. At the collaboration layer, by applying rule *C-Deliver*, the message m_1 is delivered to q. Now, on the receiving party, assuming that there is a token on e_4, by applying rules *P-InterRcv* and *P-Int$_1$*, it results that $\langle Q, \sigma_2 \rangle \xrightarrow{\mathsf{?m}} inc(dec(\sigma_2, \mathsf{e}_4), \mathsf{e}_5)$. The observed label indicates the willingness to receive a message of type m_1. Thus, at collaboration level, rule *C-Receive* can be applied to allow process Q to actually consume the sent request message. Now, $\mathsf{Task\ 3}$ is enabled and, by proceeding in a specular way, Q can send the response message m_2 and P can consume it.

Racing Incoming Messages Pattern.

Informal Description. A party expects to receive one among a set of messages. These messages may be structurally different (i.e. different types) and may come from different categories of partners. The way a message is processed depends on its type and/or the category of partner from which it comes.

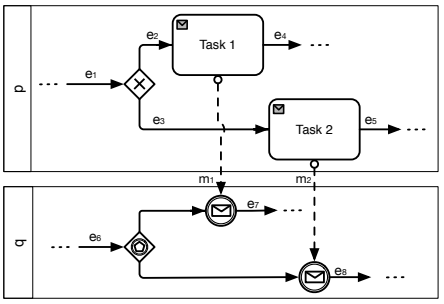


Figure 8.13: Racing Incoming Messages (a).

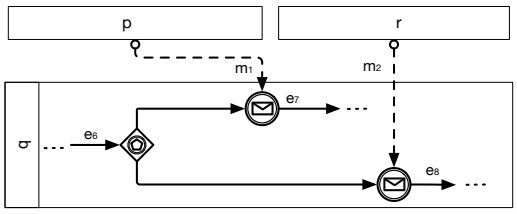


Figure 8.14: Racing Incoming Messages (b).

This pattern can be modelled in BPMN by using in the receiving participant an event-based gateway connected to receiving events. Messages can be expected from one participant (Figure 8.13) or they can arrive from different participants (Figure 8.14).

Textual Specification. Considering first the case in which messages arrive from one participant (Figure 8.13), in the textual notation the diagram is rendered as the collaboration of the usual form C, with

$$\begin{aligned} P &= \mathsf{xorSplit}(\mathsf{e}_1, \{\mathsf{e}_2, \mathsf{e}_3\}) \parallel \mathsf{taskSnd}(\mathsf{e}_2, \mathsf{m}_1, \mathsf{e}_4) \parallel \\ &\quad \mathsf{taskSnd}(\mathsf{e}_3, \mathsf{m}_2, \mathsf{e}_5) \parallel P' \\ Q &= \mathsf{eventBased}(\mathsf{e}_6, (\mathsf{m}_1, \mathsf{e}_7), (\mathsf{m}_2, \mathsf{e}_8)) \parallel Q' \end{aligned}$$

The case in which messages arrive from two different participants (Figure 8.14) is rendered in the textual notation as $C = \mathsf{pool}(\mathsf{p}, P'') \parallel \mathsf{pool}(\mathsf{r}, R) \parallel \mathsf{pool}(\mathsf{q}, Q)$, where process Q is as the above one, while P'' and R are left unspecified (because they are included in black-box pools).

Formal Semantics. Considering the case in which messages arrive from a single participant, and assuming that a token is available in the incoming edge of the XOR-Split gateway of P ($\sigma(\mathsf{e}_1) > 0$), then rule *P-XorSplit* can be applied and the token is moved to the edge e_2, hence enabling Task 1. Formally, this step corresponds to the transition $\langle P, \sigma \rangle \xrightarrow{\epsilon} \langle inc(dec(\sigma, \mathsf{e}_1), \mathsf{e}_2) \rangle$, where label ϵ denotes the movement of the token internally to the process. The next step corresponds to the execution of Task 1, which is as in the case of the *Send* pattern. Once the message m_1 has been sent, and assuming that there is a token in e_6 ($\sigma(\mathsf{e}_6) > 0$), the event-based gateway can evolve by applying the corresponding rule. This corresponds to the transition $\langle \mathsf{eventBased}(\mathsf{e}_6, (\mathsf{m}_1, \mathsf{e}_7), (\mathsf{m}_2, \mathsf{e}_8)), \sigma' \rangle \xrightarrow{?\mathsf{m}_1} inc(dec(\sigma', \mathsf{e}_6), \mathsf{e}_7)$. The rule moves the token from the incoming edge to the outgoing edge corresponding to the received message. The produced label enables the application of rule

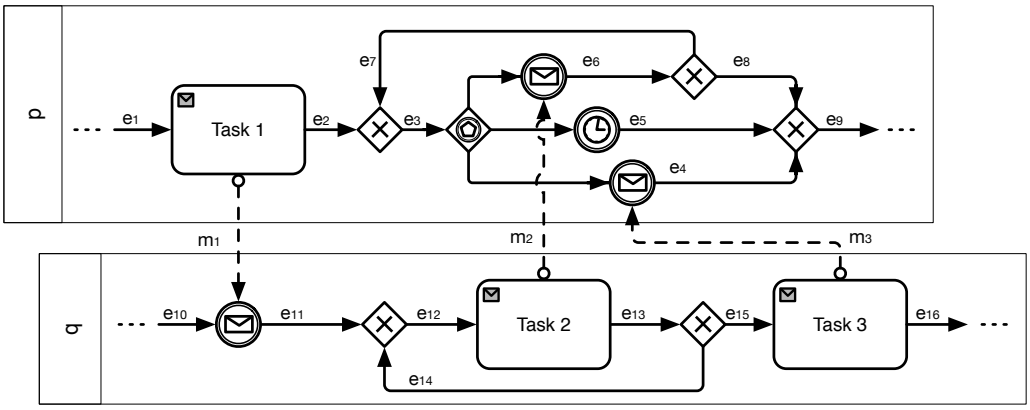


Figure 8.15: Multi-responses.

C-Receive at collaboration level, which consumes message m_1 in δ. The case where message m_2 is selected to be sent is similar.

In the scenario shown in Figure 8.14, even if the transitions produced by the collaboration have the same labels, the pattern semantics is quite different. In fact, in the previous case the organisation p internally decides which message will be sent and only one message will be delivered and consumed, while in this case the organisations p and r act independently from each other and it may occur that both m_1 and m_2 are sent to q. In such a case, one of the two messages will be consumed, depending on their arrival time, and the other message will be pending forever.

Multi-Responses Pattern.

Informal Description. A party p sends a request to another party q. Then, p receives any number of responses from q. The interaction may be stopped either by p, if a temporal condition is met, or by q, based on a message content.

This pattern can be rendered as the collaboration fragment in Figure 8.15.

Textual Specification. To simplify the formal treatment, a macro is used for the intermediate timer event. It is abstracted via a non-deterministic choice, by resorting to a race condition. In detail, the sending party p will send a message to a specific pool t and will get, via an event based gateway, at certain point a time-out message from it. In the textual notation the pattern is rendered as follows: $C = \mathsf{pool}(\mathsf{p}, P) \parallel \mathsf{pool}(\mathsf{q}, Q) \parallel \mathsf{pool}(\mathsf{t}, T)$, where:

$$
\begin{aligned}
P &= \mathsf{taskSnd}(\mathsf{e}_1, \mathsf{m}_1, \mathsf{e}_2) \parallel \mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_7\}, \mathsf{e}') \parallel \mathsf{taskSnd}(\mathsf{e}', \mathsf{m}_{startTimer}, \mathsf{e}_3) \parallel \\
&\quad \mathsf{eventBased}(\mathsf{e}_3, (\mathsf{m}_2, \mathsf{e}_6), (\mathsf{m}_{timeout}, \mathsf{e}_5), \mathsf{m}_3, \mathsf{e}_4)) \parallel \mathsf{xorSplit}(\mathsf{e}_6, \{\mathsf{e}_7, \mathsf{e}_8\}) \parallel \\
&\quad \mathsf{xorJoin}(\{\mathsf{e}_8, \mathsf{e}_5\}, \mathsf{e}_9) \parallel P' \\
Q &= \mathsf{interRcv}(\mathsf{e}_{10}, \mathsf{m}_1, \mathsf{e}_{11}), \parallel \mathsf{xorJoin}(\{\mathsf{e}_{11}, \mathsf{e}_{14}\}, \mathsf{e}_{12}) \parallel \mathsf{taskSnd}(\mathsf{e}_{12}, \mathsf{m}_2, \mathsf{e}_{13}) \parallel \\
&\quad \mathsf{xorSplit}(\mathsf{e}_{13}, \{\mathsf{e}_{14}, \mathsf{e}_{15}\}) \parallel \mathsf{taskSnd}(\mathsf{e}_{15}, \mathsf{m}_3, \mathsf{e}_{16}) \parallel Q' \\
T &= \mathsf{startRcv}(\mathsf{e}'', \mathsf{m}_{startTimer}, \mathsf{e}_{17}) \parallel \mathsf{taskSnd}(\mathsf{e}_{17}, \mathsf{m}_{timeout}, \mathsf{e}_{18}) \parallel \\
&\quad \mathsf{end}(\mathsf{e}_{18}, \mathsf{e}''')
\end{aligned}
$$

Formal Semantics. The first execution steps of this pattern are realised by combining the semantic rules for the *Send* and *Receive* patterns. In detail: supposing that there is a token in the incoming edge of Task 1 ($\sigma(\mathsf{e}_1) > 0$), by applying rule *P-TaskSnd* it follows $\langle \mathsf{taskSnd}(\mathsf{e}_1, \mathsf{m}_1, \mathsf{e}_2), \sigma \rangle \xrightarrow{!\mathsf{m}_1} inc(dec(\sigma, \mathsf{e}_1), \mathsf{e}_2)$. Then, P evolves by performing a sending action, by means of the interleaving rule *P-Int$_1$*, that is $\langle P, \sigma \rangle \xrightarrow{!\mathsf{m}} \sigma'$. At the collaboration layer, by applying rule *C-Deliver*, the message m_1 is delivered to q. Now, on the receiving party, assuming that there is a token on e_{10}, by applying rules *P-InterRcv* and *P-Int$_1$*, $\langle Q, \sigma_2 \rangle \xrightarrow{?\mathsf{m}_1} inc(dec(\sigma_2, \mathsf{e}_{10}), \mathsf{e}_{11})$. The observed label indicates the willingness to receive a message of type m_1. Thus, at collaboration level, rule *C-Receive* can be applied to allow process Q to actually consume the sent request message. Now, the XOR-Join gateway in q is enabled and, by applying rule *P-XorJoin* a token is moved from e_{11} to e_{12} enabling Task 2. Now, proceeding in a specular way, Q can send the first response message m_2 and P can consume it. The execution of Task 2 enables the activation of the XOR-Split gateway and starts the looping behaviour obtained by repeatedly applying the semantic rules of the XOR gateways and the send task. On the other side, the reception of messages is regulated by event based gateway. In detail, after executing Task 1 process P sends a message $\mathsf{m}_{startTimer}$ to process T, that is delivered as in the previous case. Now, process P can either receive messages from process Q or stop waiting for them as the timeout occurs, that is message $\mathsf{m}_{timeout}$ is delivered or receive a message m_3 from process Q saying that no further responses will arrive.

Request with Referral Pattern.

Informal Description. A party p sends a request to another party q indicating that any follow-up should be sent to another party r.

An example of a BPMN collaboration involving the request with referral pattern is shown in Figure 8.16.

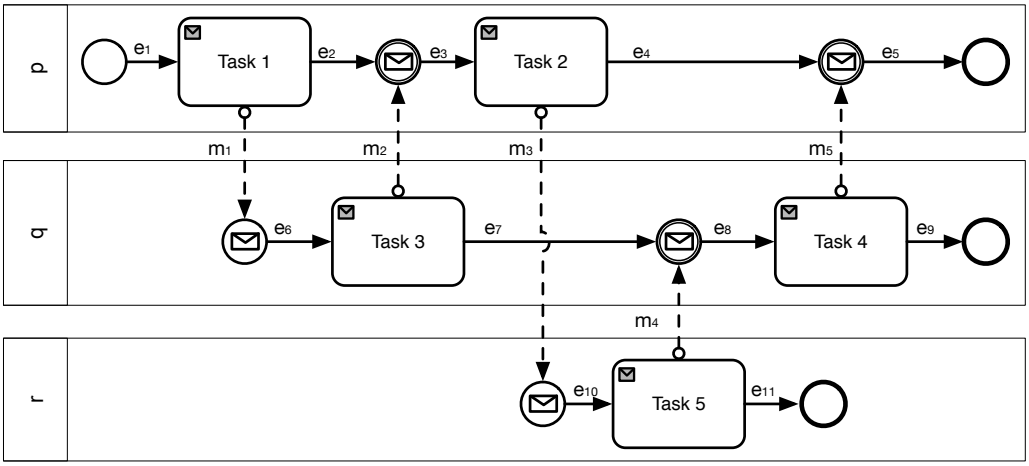


Figure 8.16: Request with Referral.

Textual Specification. In the textual specification $C = \mathsf{pool}(\mathsf{p}, P) \parallel \mathsf{pool}(\mathsf{q}, Q) \parallel \mathsf{pool}(\mathsf{r}, R)$, where:

$$P = \mathsf{start}(\mathsf{e}_0, \mathsf{e}_1) \parallel \mathsf{taskSnd}(\mathsf{e}_1, \mathsf{m}_1, \mathsf{e}_2) \parallel \mathsf{interRcv}(\mathsf{e}_2, \mathsf{m}_2, \mathsf{e}_3) \parallel \mathsf{taskSnd}(\mathsf{e}_3, \mathsf{m}_3, \mathsf{e}_4) \parallel \mathsf{interRcv}(\mathsf{e}_4, \mathsf{m}_4, \mathsf{e}_5) \parallel \mathsf{end}(\mathsf{e}_5, \mathsf{e}')$$

and Q and R are defined in a similar way.

Formal Semantics. The execution steps and their results are simply realised by applying the semantic rules for the different BPMN elements, as already shown for the previous patterns. It is up to the message sent by pool p to pool r to specify in its content the reference to pool q, whose process waits for the routed message.

Relayed Request Pattern.

Informal Description. A party p makes a request to party q, which delegates the request processing to another party r. This latter party interacts with party p while party q observes a view of the interactions.

This pattern can be rendered as the collaboration fragment in Figure 8.17.

Textual Specification. In the textual notation the pattern is rendered as follows: $C = \mathsf{pool}(\mathsf{p}, P) \parallel \mathsf{pool}(\mathsf{q}, Q) \parallel \mathsf{pool}(\mathsf{r}, R)$, where:

$$R = \mathsf{startRcv}(\mathsf{e}', \mathsf{m}_2, \mathsf{e}_7) \parallel \mathsf{andSplit}(\mathsf{e}_7, \{\mathsf{e}_8, \mathsf{e}_9\}) \parallel \mathsf{taskSnd}(\mathsf{e}_8, \mathsf{m}_4, \mathsf{e}_{10}) \parallel \mathsf{taskSnd}(\mathsf{e}_9, \mathsf{m}_3, \mathsf{e}_{11}) \parallel R'$$

while P and Q are defined in a similar way.

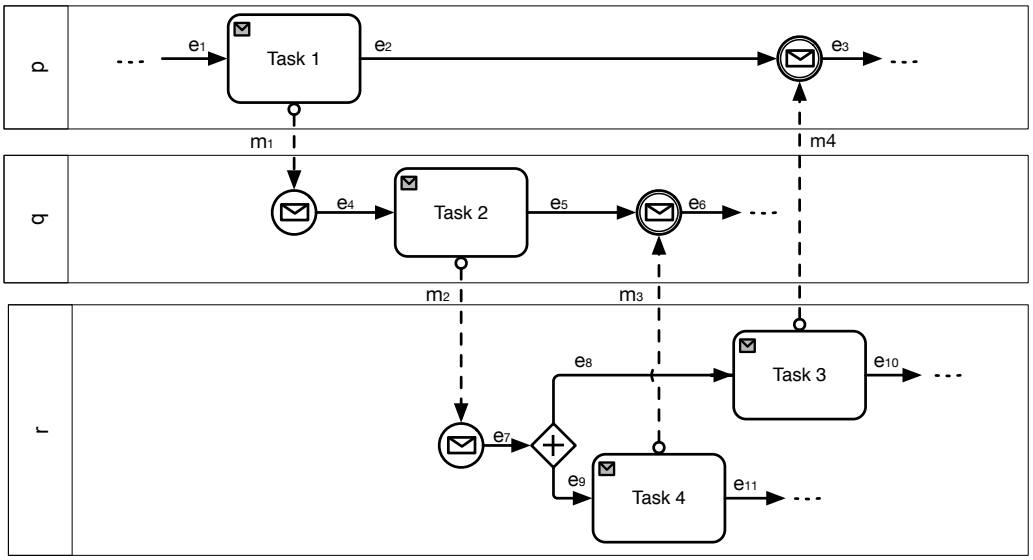


Figure 8.17: Example of Relayed Request.

Formal Semantics. Similar to the previous pattern, the formal semantics of this pattern is determined by the application of rules for sending and receiving message already described, except for the AND-Split gateway that simply consumes a token in e_7 and, simultaneously, produces one token in e_8 and one token in e_9.

8.2 BPMN Patterns with OR Gateways

This section shows how the extension of the semantics with the OR gateway allows to visualise and formalise a series of patterns which characterise more complex branching and merging concepts which arise in business processes.

Multiple Choice Pattern.

Informal Description. The ability to depict the divergence of a branch into two or more branches. When the incoming branch is enabled, the thread of control is passed to one or more of the outgoing branches.

This pattern can be captured in BPMN through the OR-Split gateway as shown in Figure 8.18.

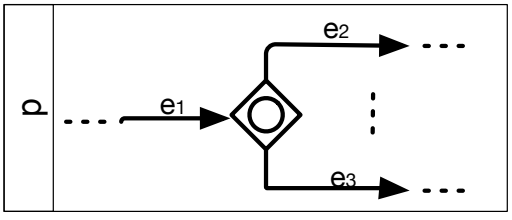


Figure 8.18: Multiple Choice.

Textual Specification. The process fragment in Figure 8.18 is represented in the textual notation as

follows:

$$P = \mathsf{orSplit}(\mathsf{e}_1, \{\mathsf{e}_2, \mathsf{e}_3\}) \parallel P'$$

Formal Semantics. The formal semantics of this pattern is determined by the application of rule for OR-Split gateway as described in Section 5.3. In particular, assuming that the OR-Split gateway is enabled by a token in e_1, then the process P can evolve by means of rule *G-OrSplit*. Assuming that only one branch is selected, for instance e_2, then the application of the rule produces the following transition $\langle \mathsf{orSplit}(\mathsf{e}_1, \{\mathsf{e}_2, \mathsf{e}_3\}), \sigma \rangle \rightarrow_G inc(dec(\sigma, \mathsf{e}_1), \mathsf{e}_2)$. Then, the process P evolves by means of the interleaving rule *G-Int*$_1$.

Synchronising Merge Pattern.

Informal Description. The ability to depict the convergence of two or more branches into a single subsequent branch where the thread of control is passed to the subsequent branch when each active incoming branch has been enabled.

This pattern can be captured in BPMN through the OR-Join gateway as shown in Figure 8.19.

Textual Specification. In the textual notation the pattern is as follows:

$$P = \mathsf{orJoin}(\{\mathsf{e}_1, \mathsf{e}_2\}, \mathsf{e}_3) \parallel P'$$

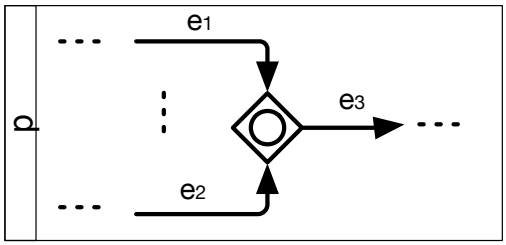


Figure 8.19: Synchronising Merge.

Formal Semantics. Similar to the previous case, the formal semantics of this pattern is determined by the application of rule for OR-Join gateway already described in Section 5.3. Morevoer, differently from what argue in [94], the provided semantics, compliant with the current BPMN 2.0 OR-Join specification, allows to fully support this pattern also when dealing with unstructured processes.

8.3 BPMN Patterns with Sub-Processes

This section shows how the BPMN definition of sub-process completion, fully supported by the semantics provided in Section 6.2, allows to characterise a particular type of termination.

Implicit Termination Pattern.

Informal Description. The notion that a given sub-process should be terminated when there are no remaining activities to be completed.

This pattern is rendered in BPMN by using a sub-process. An example is shown in Figure 8.20.

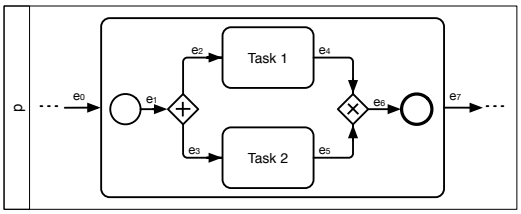


Figure 8.20: Implicit Termination.

Textual Specification. In the textual notation the process fragment in Figure 8.20 is as follows:

$$P = \mathsf{subProc}(\mathsf{e}_0, P_1, \mathsf{e}_7) \parallel P'$$

where P_1 is as follows:

$$\begin{aligned} P_1 \;=\; & \mathsf{start}(\mathsf{e}', \mathsf{e}_1), \mathsf{andSplit}(\mathsf{e}_1, \{\mathsf{e}_2, \mathsf{e}_3\}) \parallel \mathsf{task}(\mathsf{e}_2, \mathsf{e}_4) \parallel \mathsf{task}(\mathsf{e}_3, \mathsf{e}_5) \\ & \parallel \mathsf{xorJoin}(\{\mathsf{e}_4, \mathsf{e}_5\}, \mathsf{e}_6) \parallel \mathsf{end}(\mathsf{e}_6, \mathsf{e}'') \parallel P' \end{aligned}$$

Formal Semantics. This pattern is fully supported by the semantics provided for BPMN models including sub-processes (Section 6.2). In fact, according to the BPMN notion of sub-process completion, fully translated by the provided semantics, a sub-process instance completes when there are no more tokens in the sub-process and none of its activities is still active. In the example in Figure 8.20, assuming that there is a token in the incoming edge of the sub-process ($\sigma(\mathsf{e}_0) > 0$), rule *P-SubProcStart* can be applied, moving the token to the enabling edge of the start event in the sub-process body e'. Now, the sub-process behaves according to the behaviour of the elements it contains according to the rule *P-SubProcEvolution*. In the depicted case, rule *P-AndSplit* can be applied producing two tokens on the outgoing edges of the AND-Split gateway e_2, e_3. These tokens enable the interleaved execution of Task 1 and Task 2. After both the tasks are performed, the two tokens are merged by the XOR-Join gateway. Thus, in a certain state of the execution two tokens may occur on e_6. Only when both the tokens reach the end-event of the sub-process, the sub-process execution can complete, according to rule *P-SubProcEnd*.

8.4 BPMN Patterns with Multiple Instances

Among the workflow patterns, many of the service interaction patterns are supported by BPMN thanks to its capability to model multiple instances. However, the BPMN notation has some limitations and cannot completely

support all their features. For instance, while the informal and general description of these patterns leaves it open if in an interaction the counter-party is known at design-time or not, in BPMN it is expected to have a priori knowledge of the interacting partners, i.e. the target pool of a message edge cannot be dynamically selected. On the other hand, in case a message is directed to a multi-instance pool, BPMN supports a form of runtime binding of the message with the correct process instance by means of the correlation mechanism [68, Section 8.3.2]. Moreover, it is also possible to dynamically specify other model features, such as the number of involved participants and exchanged messages.

This section visualises and formalises those patterns modelled via BPMN multi-instance features.

One-To-Many Send Pattern.

Informal Description. A party sends messages to several parties. All messages have the same type (although their contents may be different). The number of parties to whom the message is sent may or may not be known at design time.

In BPMN, this pattern can be modelled as the collaboration fragment in Figure 8.21, where each party is represented as an instance of a multi-instance pool and a message is sent to each process instance via a sequential multi-instance send task. From now on, when a message is sent/received to/by several parties, these parties will be modelled as a multiple instance pool. This is the most interesting among various interpretations which are not considered in this work (e.g., representing multiple receiving parties as different single-instance pools). It can happen that the number of sent messages is either known at design time (by setting the LoopCardinality attribute of the send task) or it is read from a data object during the process execution.

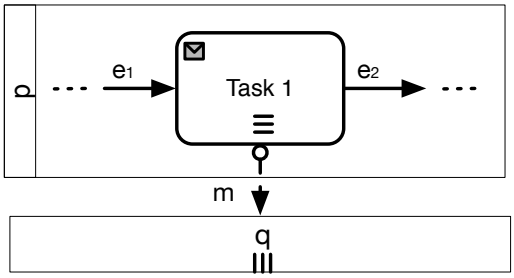


Figure 8.21: One-To-Many-Send.

Textual Specification. Here, to keep the pattern formalisation more manageable, the sequential multi-instance task is rendered as a macro. The macro encloses the task in a FOR-loop expressed by means of a pair of XOR join and split gateways, and an additional data object c_1 for the loop counter. In the textual notation we have $C = \mathsf{pool}(\mathsf{p}, P) \parallel \mathsf{miPool}(\mathsf{q}, Q)$, where process Q is left unspecified and in P the attribute LoopCardinality is set to n:

$$P = \mathsf{xorJoin}(\{\mathsf{e}_1, \mathsf{e}_1'''\}, \mathsf{e}_1') \parallel \mathsf{taskSnd}(\mathsf{e}_1', \mathsf{c}_1.\mathsf{c} \neq \mathsf{null}, \mathsf{c}_1.\mathsf{c} := \mathsf{c}_1.\mathsf{c} + 1, \mathsf{m} : \tilde{\mathsf{exp}}_1, \mathsf{e}_1'') \parallel \mathsf{xorSplit}(\mathsf{e}_1'', \{(\mathsf{e}_1''', \mathsf{c}_1.\mathsf{c} \leqslant \mathsf{n}), (\mathsf{e}_2, \mathsf{default})\}) \parallel P'$$

Formal semantics. The execution steps are realised as in the previous cases, by repeatedly applying the semantic rules of the XOR gateway and the send task. It is worth noticing that at each application of rule *P-TaskSnd* the field c of the data object c_1 is updated with the assignment $\mathsf{c}_1.\mathsf{c} := \mathsf{c}_1.\mathsf{c} + 1$. At the end of the pattern execution, the message list $\delta(\mathsf{m})$ contains n sent messages.

One-From-Many Receive Pattern.

Informal Description. A party receives several logically related messages arising from autonomous events occurring at different parties. The arrival of messages must be timely so that they can be correlated as a single logical request. The interaction may complete successfully or not depending on the messages gathered. In this pattern the receiver does not know the number of messages that will arrive, and stops waiting as soon as a certain number of messages have arrived or a timeout occurs.

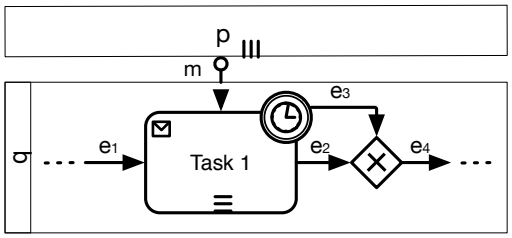


Figure 8.22:
One-From-Many-Receive.

This pattern can be modelled as the collaboration fragment shown in Figure 8.22.

Textual Specification. Also in this case, to simplify the formal treatment, a macro is used to render the multi-instance receive task with a timer. In particular, the multi-instance behaviour is represented by enclosing the receive task in a `FOR`-loop (as for the sequential multi-instance task). The timer attached to the receive task is instead abstracted via a non-deterministic choice, by resorting to a race condition. In detail, the receiving party q will get, via an event-based gateway, either a message from a sending party (i.e., an instance of p) or a time-out message from a specific pool t representing the timer. In the textual notation we have $C = \mathsf{miPool}(\mathsf{p}, P) \parallel \mathsf{pool}(\mathsf{q}, Q) \parallel \mathsf{pool}(\mathsf{t}, T)$, where

$$
\begin{aligned}
Q &= \mathsf{taskSnd}(\mathsf{e}_1, \mathsf{exp}_1, \epsilon, \mathsf{m}_{startTimer} : \tilde{\mathsf{exp}}_2, \mathsf{e}') \parallel \mathsf{xorJoin}(\{\mathsf{e}', \mathsf{e}^{\mathsf{v}}\}, \mathsf{e}'') \parallel \\
&\quad \mathsf{eventBased}(\mathsf{e}'', (\mathsf{m} : \tilde{\mathsf{t}}_1, \mathsf{e}'''), (\mathsf{m}_{timeout} : \tilde{\mathsf{t}}_2, \mathsf{e}_3)) \parallel \\
&\quad \mathsf{task}(\mathsf{e}''', \mathsf{c}_1.\mathsf{c} \neq \mathsf{null}, \mathsf{c}_1.\mathsf{c} := \mathsf{c}_1.\mathsf{c} + 1, \mathsf{e}^{\mathsf{iv}}) \parallel \\
&\quad \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{(\mathsf{e}^{\mathsf{v}}, \mathsf{c}_1.\mathsf{c} \leqslant \mathsf{n}), (\mathsf{e}_2, \mathsf{default})\}) \parallel \mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e3}\}, \mathsf{e}_4) \parallel Q' \\
T &= \mathsf{startRcv}(\mathsf{m}_{startTimer} : \tilde{\mathsf{t}_3}, \mathsf{e}_5) \parallel \mathsf{taskSnd}(\mathsf{e}_5, \mathsf{exp}_3, \epsilon, \mathsf{m}_{timeout} : \tilde{\mathsf{exp}}_4, \mathsf{e}_6) \parallel \\
&\quad \mathsf{end}(\mathsf{e}_6, \mathsf{e}_7)
\end{aligned}
$$

Formal Semantics. Once a token arrives at e_1 in the process Q, a $\mathsf{m}_{startTimer}$ message is sent to the pool t by means of the send task, in order to activate an instance of the timer process T. This instance will perform a send task, delivering a message $\mathsf{m}_{timeout}$, to signal that the timeout is expired, and then it terminates. As effect of the execution of the send task in Q, a token is moved in e', which enables the looping behaviour regulated by the XOR gateways. At each iteration, the event-based gateway consumes either a message m or $\mathsf{m}_{timeout}$; in the former case the non-communicating task increments the loop counter and the execution of another interaction is evaluated (by means of the XOR-Split conditions), while in the latter case the edge e_3 is followed and the pattern execution completes.

One-To-Many Send/Receive Pattern.

Informal Description. A party sends a request to several other parties. Responses are expected within a given time-frame. However, some responses may not arrive within the time-frame and some parties may even not respond at all.

This pattern can be rendered as the collaboration fragment in Figure 8.23.

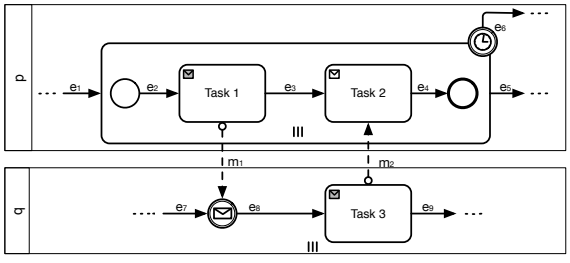


Figure 8.23: One-To-Many Send/Receive.

Textual Specification. This pattern relies on a multi-instance sub-process with a specific form, i.e. it is characterised by a sequence of a send task and a receive task, proceeded and followed by a start and an end event, respectively. As usual, to simplify the formal treatment a macro is used. In this case, it consists of a sequential send task followed by a multi-instance receive task with a timer. Thus, $C = \mathsf{pool}(\mathsf{p}, P) \parallel \mathsf{miPool}(\mathsf{q}, Q)$, where process P is rendered in terms of macros as already shown in the previous patterns (hence, for the sake of presentation, its specification is omitted),

while process Q is as follows:

$$Q \;=\; \mathsf{interRcv}(\mathsf{e}_7, \mathsf{m}_1 : \tilde{\mathsf{t}_1}, \mathsf{e}_8) \parallel \mathsf{taskSnd}(\mathsf{e}_8, \mathsf{exp}, A, \mathsf{m}_2 : \tilde{\mathsf{exp}}, \mathsf{e}_9) \parallel Q'$$

Formal Semantics. In this pattern process P sends out, by means of rule *P-TaskSnd*, several messages of type m1 that need to be properly correlated with the correct process instance of Q. The content of the messages themselves provides the correlation information. For example, let us assume that two messages of type m1 are sent to q, and that consist of three fields, say $\langle$"foo", $5, 1234\rangle$ and $\langle$"foo", $7, 9876\rangle$. Also, let us consider the case where there are two receiving instances, i.e. $\iota(\mathsf{q}) = \{\langle\sigma_1, \alpha_1\rangle, \langle\sigma_2, \alpha_2\rangle\}$, and that template $\tilde{\mathsf{t}_1}$ of the intermediate receiving event is defined as $\langle \mathsf{d.f}, \mathsf{d.id}, ?\mathsf{d.code}\rangle$, meaning that the fields f and id of the data object d identify correlation data while code is a formal field. Now, the correlation takes place according to the data states, which are assumed to be as follows: $\alpha_1(\mathsf{d.f}) = \alpha_2(\mathsf{d.f}) =$ "foo", $\alpha_1(\mathsf{d.id}) = 7$, and $\alpha_2(\mathsf{d.id}) = 5$. Therefore, the first message is delivered to the second instance, updating α_2 with the assignment d.code =9876, while the second message is delivered to the first instance, updating α_1 with the assignment d.code =1234.

Contingent Requests Pattern.

Informal Description. A party p makes a request to another party q. If p does not receive a response within a certain time-frame, it sends a request to another party r, and so on.

This pattern can be rendered as the collaboration fragment in Figure 8.24.

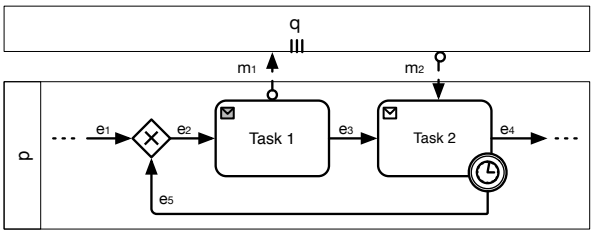


Figure 8.24: Contingent Requests.

Textual Specification. In the scenario in Figure 8.24, requests are made to a multi-instance pool q. To simplify the formal treatment, as usual, macro is used for the multi-instance receive task with a timer. In the textual notation the pattern is rendered as follows: $C = \mathsf{pool}(\mathsf{p}, P) \parallel \mathsf{miPool}(\mathsf{q}, Q) \parallel \mathsf{pool}(\mathsf{t}, T)$, where Q is left unspecified while P and T are as follow:

$$\begin{aligned} P \;=\;& \mathsf{xorJoin}(\{\mathsf{e}_1, \mathsf{e}_5\}, \mathsf{e}_2) \parallel \mathsf{taskSnd}(\mathsf{e}_2, \mathsf{exp}_1, \epsilon, \mathsf{m}_1 : \tilde{\mathsf{exp}}_2, \mathsf{e}') \parallel \\ & \mathsf{taskSnd}(\mathsf{e}', \mathsf{exp}_3, \epsilon, \mathsf{m}_{startTimer} : \tilde{\mathsf{exp}}_4, \mathsf{e}_3) \parallel \\ & \mathsf{eventBased}(\mathsf{e}_3, (\mathsf{m}_2 : \tilde{\mathsf{t}_1}, \mathsf{e}_4), (\mathsf{m}_{timeout} : \tilde{\mathsf{t}_3}, \mathsf{e}_5)) \parallel P' \\ T \;=\;& \mathsf{startRcv}(\mathsf{m}_{startTimer} : \tilde{\mathsf{t}_4}, \mathsf{e}_6) \parallel \mathsf{taskSnd}(\mathsf{e}_6, \mathsf{exp}_5, \epsilon, \mathsf{m}_{timeout} : \tilde{\mathsf{exp}}_6, \mathsf{e}_7) \parallel \\ & \mathsf{end}(\mathsf{e}_7, \mathsf{e}''') \end{aligned}$$

Formal Semantics. The execution steps of this pattern are realised by repeatedly applying the semantic rules of the *Send/Receive* pattern while taking into account the timer condition and the correlation mechanism. Each time a response does not arrive within the time condition a new request is sent to a new instance of pool `q`.

8.5 Semantics Validation

The workflow pattens provide an analysis framework for modelling languages. They have been also used to evaluate BPMN expressive power [94]. Investigating the possibility of expressing every BPMN pattern in a selected semantics builds up a comprehensive picture of the suitability of this semantics to cover the patterns expressed in BPMN, thus providing a validation of the formal semantics.

This section reports in Table 8.1 the results of the semantics evaluation. The evaluation is done with respect to the BPMN service interaction patterns as described in [15], and to the workflow patterns fully supported by BPMN, as specified in [94]. Even if these latter represent a subset of the whole workflow patterns, they result to be a reference framework as they consist of generic, recurring concepts and constructs relevant in the context of business processes.

The evaluation does not directly take into account the data perspective. In fact, although the data perspective is addressed in the semantics for multi-instance collaborations, data patterns are mostly specified through the attributes of the BPMN elements, that are part of the low-level XML representation. Moreover, since BPMN is not specifically tailored to the needs of service-oriented interaction patterns, the notation cannot completely support all the interaction patterns [15]. Thus, those patterns realised by means of different workarounds (e.g., atomic multicast notification) are not considered.

The table reports only the patterns supported by at least one of the provided semantics. For each of the considered pattern it is shown if each semantics is able to formalise it (+) or not (-).

Specifically, Table 8.1 shows that the core BPMN semantics covers the basic patterns. It supports the 60% of the BPMN service interaction patterns and workflow patterns [94]. Increasing the pattern complexity, new elements are required to model them and the semantics needs to be extended. However, the provided BPMN semantics including OR-Joins is still restricted to a limited number of elements, due to the intricate OR-Join behaviour, supporting only the 30,3% of the patterns. Instead, the semantics of BPMN models including sub-processes enlarges the number of covered patterns, by supporting the 62,8% of them. It is with the semantics of BPMN multi-instance collaborations that there is the highest number of supported patterns. This

Pattern	Core BPMN	BPMN with OR-Joins	BPMN with Sub-Processes	BPMN with Multiple Instances
Sequence	+	+	+	+
Parallel Split	+	+	+	+
Synchronisation	+	+	+	+
Exclusive Choice	+	+	+	+
Simple Merge	+	+	+	+
Multiple Choice	-	+	-	-
Synchronising Merge	-	+	-	-
Multiple Merge	+	+	+	+
Arbitrary Cycles	+	+	+	+
Implicit Termination	-	-	+	-
Deferred Choice	+	-	+	+
Cancel Case	+	-	+	+
Direct Allocation	+	-	+	+
Role-Based Allocation	+	-	+	+
Automatic Execution	+	+	+	+
Commencement on Creation	+	-	+	+
Chained Execution	+	-	+	+
Send	+	-	+	+
Receive	+	-	+	+
Send/Receive	+	-	+	+
Racing Incoming Messages	+	-	+	+
Multi-Responses	+	-	+	+
Contingent Requests	-	-	-	+
One-To-Many Send	-	-	-	+
One-From-Many Receive	-	-	-	+
One-To-Many Send/Receive	-	-	-	+
Request with Referral	+	-	+	+
Relay Request	+	-	+	+

Table 8.1: Semantics Validation Results.

semantics is able to formalise the 71,4% of the patterns.

Part IV

Conclusions & Future Work

Chapter 9

Conclusions and Future Work

The research presented in this thesis is centred around the topic of providing a uniform formal framework to characterise BPMN models, with the aim to classify them according to relevant properties of the business process domain. Chapter 1 reports an introduction to business process modelling languages, with a focus on BPMN, and on business process model properties. It concludes by showing the objectives and research questions of the thesis with a summary description of the thesis structure. Chapter 2 provides preliminary notions used along the thesis. Specifically, it introduces concepts of BPM, with an insight to the BPMN notation, and relevant properties of the business process domain. It concludes with an overview of formal methods techniques. Targeting the main research question of this thesis on the formalisation and classification of BPMN models, several sub-research questions are identified, defining the scope of the thesis. In particular, Chapter 3 defines a direct formal semantics in the SOS style for a core subset of BPMN elements. Being close to the BPMN standard the semantics enables to catch the language peculiarities such as the asynchronous communication models, and the completeness notion distinguishing the effect of end event and the terminate event. Towards this formalisation, it is possible to formally define some important correctness properties, namely well-structuredness, safeness, and soundness, both at the process and collaboration level and to demonstrate the relationships among them. Specifically, it is shown 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. It is proved that there are well-structured collaborations that are neither sound nor message-relaxed sound. Finally, it is demonstrated there are sound and message-relaxed sound collaborations that are not safe. These models are typically discarded by the modelling approaches in the literature, as they are over suspected of carrying bugs. However, as shown, this attitude significantly limits the use of concurrency in business process modelling, which

is an important feature in modern systems and organisations practice. To continue to effectively reason on the whole set of BPMN elements included in a collaboration, the formal semantics is extended by including other BPMN constructs. Specifically, Chapter 5 presents a global and local direct formalisation of BPMN process models compliant with the OR-Join semantics reported in the BPMN 2.0 standard. The two semantics are proven to be in formal correspondence. However, while the global semantics provides a reference point, the local semantics fosters a compositional, and hence more scalable, approach to practically enact business processes involving OR-Joins. The precise semantics allows to properly classify BPMN models including the OR-Join. Chapter 6 enriches the semantics with the sub-process element, thus adding another level of abstraction and considering the effects of the sub-process completion on the provided classification. Chapter 7 extends the semantics by taking into account the data perspective. This is strongly related to message exchanges and control features, especially in scenario involving multi-interacting participants. It is shown that in such complex models the classification results are still valid. Moreover, the formalisation enables to develop an animator tool, called MIDA, supporting model debugging. Finally, Chapter 8 validates the different proposed semantics as it is shown they are suitable to cover the Workflow Patterns expressed in BPMN.

This section concludes by discussing results of the thesis, and the assumptions and limitations of the given approach, also touching upon directions for future work.

Results. The thesis contributes to the definition of a direct BPMN operational semantics, providing a uniform formal framework to study BPMN models and their main correctness properties (well-structuredness, safeness and soundness), thus classifying BPMN models according to the properties they satisfy.

First only a subset of BPMN elements is considered. Indeed, such subset of BPMN elements has been selected by following a pragmatic approach and only by retaining the features mostly used in practice. Therefore, even if it focusses on a restricted number of elements, such design choice cannot be considered a major limitation. However, if necessary, the framework can be extended to cover further elements, thus enabling the classification of a major number of BPMN models. The considered BPMN features (OR-Join, sub-processes and multiple instances and data) are rather orthogonal; indeed it could be expected that the different variants can be integrated each other. This is quite evident when considering the semantics of BPMN with sub-processes, which has been obtained from the one of the BPMN core elements by simply adding a new construct to the formal framework. Differently, some challenges could arise when considering the OR-Join gateway, due to its quite

articulated behaviour. Indeed, the presented semantics with OR gateways considers only the process level. Adding other level of abstractions (collaborations and sub-processes) would add complexity to the formal treatment, as explained later in the section. Finally, the semantics for multi-instance collaborations naturally extends the one given for the core BPMN elements, although it adds new information in order to deal with data and instance correlations. However, also in this case, the sub-process level is not considered, as discussed later in the section.

The possibility to combine the different semantics is supported by the classification and compositionality results, that are preserved through the various frameworks. These results are summed up in Table 9.1. The table presents, for each of the considered framework, the results obtained in terms of classification of BPMN models (both at the process and collaboration level) and of compositionality. In particular, the terms *(Terminate)* and *(Terminate + Sub-Process)* indicate the impact of these elements on the relationship between properties, while a gray column indicates those aspects that have not been considered in a specific framework.

Assumptions and Limitations. The provided formal semantics focusses on the communication mechanisms of collaborative systems, where (multiple) participants cooperate and share information. Thus, this thesis has left out those features of BPMN whose formal treatment is orthogonal to the addressed problem, such as timed events and error handling. To keep the formalisations more manageable other simplifications are in place. They are discussed below.

Concerning the formalisation of models including OR-Join gateways, the thesis considers only the process level. Extending the formal framework to collaboration diagrams would add complexity to the formal treatment. In fact, in collaborative scenarios the conditions enabling the OR-Join should take into account also the possible arrival of message tokens.

Regarding the extension of the semantics with multiple instances and data, multi-instance parallel tasks, sub-processes and data stores are left out, despite they can be relevant for multi-instance collaborations. The impact of their addition to the presented work is as follows. Considering multi-instance tasks, the sequential instances case, as shown in the formalisation of the running example, can be simply dealt with as a macro; indeed, it corresponds to a task enclosed within a 'for' loop. The parallel case, instead, is more tricky. It is a common practice to consider it as a macro as well, which can be replaced by tasks between AND split and join gateways [29, 111], assuming to know at design time the number of instances to be generated. However, this replacement is no longer admissible when this kind of element is used within multi-instance pools [23], thus requiring a direct definition of the formal se-

<table>
<tr><th colspan="2"></th><th>Core
BPMN</th><th>BPMN
with OR-Joins</th><th>BPMN with
Sub-Processes</th><th>BPMN with
Multiple Instances</th></tr>
<tr><td rowspan="3">CLASSIFICATION RESULTS</td><td>Process</td><td colspan="4">WS ⇒ Safe
WS ⇒ Sound
Sound ⇏ Safe</td></tr>
<tr><td>Process</td><td>Unsafe ⇏ Unsound
(Terminate)</td><td rowspan="2"></td><td>Unsafe ⇏ Unsound
(Terminate + Sub-Process)</td><td>Unsafe ⇏ Unsound
(Terminate)</td></tr>
<tr><td>Collaboration</td><td>WS ⇒ Safe
WS ⇏ (MR)-Sound
(MR)-Sound ⇏ Safe
Unsafe ⇏ Unsound</td><td colspan="2">WS ⇒ Safe
WS ⇏ (MR)-Sound
(MR)-Sound ⇏ Safe
Unsafe ⇏ Unsound</td></tr>
<tr><td colspan="2">COMPOSITIONALITY</td><td>Safe Processes ⇒
Safe Collaboration

Unsound Processes ⇒
Unsound Collaboration</td><td></td><td>Unsafe Sub-Process ⇒
Unsafe Process

Unsound Sub-Process ⇏
Unsound Process</td><td>Safe Processes ⇒
Safe Collaboration

Unsound Processes ⇒
Unsound Collaboration</td></tr>
</table>

Table 9.1: Results

mantics of multi-instance parallel tasks. Similar reasoning can be done for sub-processes, which again are not mere macros. In fact, in general, simply flattening a process by replacing its sub-process elements by their expanded processes results in a model with different behaviour. This because a sub-process, for example, delimits the scope of the enclosed data objects and confines the effect of termination events. Therefore, it would be necessary to explicitly deal with the resulting multi-layer perspective, which adds complexity to the formal treatment. The formalisation would become even more complex when considering multi-instance sub-processes, which would require an extension of the correlation mechanism. Moreover, BPMN data stores are not considered. Providing a formalisation for data stores would require to extend collaboration configurations with a further state function, dedicated to data stores. In addition, the treatment of data assignments would become more intricate, as it would be necessary to distinguish updates of data objects from those of data stores, which affect different data state functions in the configuration. Finally, values of data objects can be somehow "constrained" by assignments. Indeed, assignment expressions can restrict the set of possible values that can be assigned to a data object field. Moreover, guard expressions of tasks or XOR-Split gateways can check if data object values respect given conditions. However, such constraints imposed on data object values are currently "hidden" in the expressions and, hence, in their evaluation. Assignments could be extended with an explicit definition of constraints in order to ease their specification and make more evident the effects of assignments on data values.

Future Work. As a future direction it is planned to continue the programme to effectively support modelling of BPMN collaborations, by overcoming the above limitations. More practically, the formal framework could be extended, in order to have a comprehensive classification of BPMN models. Moreover, it is planned to formally study other properties, especially related to the data perspective (e.g., missing data, lost data updates).

Beyond this, other aspects could be studied, such as other possibilities for validating the semantics and verification frameworks for BPMN.

Regarding the semantics validation, the formalisation of the Workflow Patterns expressed in BPMN, presented in Chapter 8, allows to validate the semantics, both in terms of the considered BPMN elements and of the expected semantic behaviour. However, ensuring that the provided formal semantics is exactly the semantics of BPMN, as described in the standard is another, recurrent issue. This could be faced in different and complementary ways:

- Comparing the provided semantics with respect to implemented semantics of BPMN. Even if the comparison is not exhaustive, it can be

used to verify that one BPMN diagram, whose execution is formally specified, is consistent with an implemented one;

- Comparing the semantics with well-established formalisations, such mapping into other formalisms, e.g. Petri Nets;
- Making clear the correspondence between the semantic choices done in the thesis and the BPMN standard, by quoting this latter in the description of each rule (as already done for the OR-Join rule in Section 5.3).

Concerning verification, the proposed formalisation provides a compositional approach based on LTS, which paves the way for the use of consolidated analysis techniques and related software tools. Indeed, it allows one to verify properties on the model using consolidated formal reasoning techniques based on LTS, as e.g. model checking of properties expressed as formulae of action-based temporal logic (as, for instance, has been done for the core BPMN in [22]). In this regard, a study on the complexity of verification problems could be done. This should be analysed according to the considered class of business processes and to the used verification techniques. Some results can be expected from studies already done in the field of WF-Nets. Specifically, in [104] it is argued that for simple models (independent of the language used) any form of soundness is decidable while for models using notions such as priorities, cancellation, etc. no form of soundness is decidable, while in [96] it is proved that in case of well-structured WF-Nets soundness can be verified very efficiently. To sum up, there is the intention to exploit an extended formal semantics, and its implementation, to enable the verification of properties using, e.g., model checking techniques.

Bibliography

[1] Flowable v.6.1.0, www.flowable.org.

[2] Process maker v.3.2, www.processmaker.com.

[3] Signavio v.11.2.0, www.signavio.com.

[4] Stadust v.4.1.0, www.eclipse.org/stardust.

[5] Sydle, www.sydle.com.

[6] Thomas Allweyer and Stefan Schweitzer. A Tool for animating BPMN token flow. In *International Workshop on Business Process Modeling Notation*, LNBIP, pages 98–106. Springer, 2012.

[7] Farhad Arbab, Natallia Kokash, and Sun Meng. Towards Using Reo for Compliance Aware Business Process Modeling. In *Leveraging Applications of Formal Methods, Verification and Validation*, volume 17 of *CCIS*, pages 108–123. Springer, 2008.

[8] Ahmed Awad, Gero Decker, and Niels Lohmann. Diagnosing and Repairing Data Anomalies in Process Models. In *Business Process Management Workshops*, volume 43 of *LNBIP*, pages 5–16. Springer, 2009.

[9] Alistair Barros and Egon Börger. A compositional framework for service interaction patterns and interaction flows. In *International Conference on Formal Engineering Methods*, volume 3785 of *LNCS*, pages 5–35. Springer, 2005.

[10] Alistair Barros, Marlon Dumas, and Arthur H.M. ter Hofstede. Service interaction patterns. In *Business Process Management*, pages 302–318. Springer, 2005.

[11] Kimon Batoulis and Mathias Weske. Soundness of decision-aware business processes. In *International Conference on Business Process Management*, volume 297 of *LNBIP*, pages 106–124. Springer, 2017.

[12] Jörg Becker, Martin Kugeler, and Michael Rosemann. *Process management: a guide for the design of business processes*. Springer Science & Business Media, 2013.

[13] Marco Brambilla, Piero Fraternali, and Carmen Vaca. BPMN and design patterns for engineering social BPM solutions. In *International Conference on Business Process Management*, volume 99 of *LNBIP*, pages 219–230. Springer, 2011.

[14] Egon Börger and Bernhard Thalheim. A Method for Verifiable and Validatable Business Process Modeling. In *Advances in Software Engineering*, volume 5316 of *LNCS*, pages 59–115. Springer, 2008.

[15] Dario Campagna, Carlos Kavka, and Luka Onesti. Enhancing BPMN 2.0 support for service interaction patterns. In *2014 9th International Conference on Software Engineering and Applications (ICSOFT-EA)*, 2014.

[16] Camunda services GmbH. Camunda 2017 v.7.7.0, www.camunda.com.

[17] Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. In *Foundations of Software Technology and Theoretical Computer Science*, pages 326–337. Springer, 1993.

[18] David Raymond Christiansen, Marco Carbone, and Thomas Hildebrandt. Formal Semantics and Implementation of BPMN 2.0 Inclusive Gateways. In *Web Services and Formal Methods*, volume 6551 of *LNCS*, pages 146–160. Springer, 2011.

[19] Flavio Corradini, Alessio Ferrari, Fabrizio Fornari, Stefania Gnesi, Andrea Polini, Barbara Re, and Giorgio Oronzo Spagnolo. A Guidelines framework for understandable BPMN models. *Data & Knowledge Engineering*, 113:129–154, 2018.

[20] Flavio Corradini, Fabrizio Fornari, Chiara Muzi, Andrea Polini, Barbara Re, and Francesco Tiezzi. On Avoiding Erroneous Synchronization in BPMN Processes. In *International Conference on Business Information Systems*, volume 288, pages 106–119. Springer, LNBIP, 2017.

[21] Flavio Corradini, Fabrizio Fornari, Andrea Polini, Barbara Re, and Francesco Tiezzi. A Formal Approach to Modelling and Verification of Business Process Collaborations. In *Science of Computer Programming*, volume 166, pages 35–70, 2018.

[22] Flavio Corradini, Fabrizio Fornari, Andrea Polini, Barbara Re, Francesco Tiezzi, and Andrea Vandin. BProVe: A formal verification framework for business process models. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, pages 217–228. IEEE Press.

[23] Corradini et al. Animating multiple instances in BPMN collaborations. Tech.Rep., University of Camerino, 2018. Available at: http://pros.unicam.it/mida/.

[24] Gero Decker, Remco Dijkman, Marlon Dumas, and Luciano García-Bañuelos. Transforming BPMN diagrams into YAWL nets. In *Business Process Management*, volume 5240 of *LNCS*, pages 386–389. Springer, 2008.

[25] Gero Decker, Frank Puhlmann, and Mathias Weske. Formalizing service interactions. In *International Conference on Business Process Management*, volume 4102 of *LNCS*, pages 414–419. Springer, 2006.

[26] Juliane Dehnert and Peter Rittgen. Relaxed Soundness of Business Processes. In *Advanced Information Systems Engineering*, volume 2068, pages 157–170. Springer, 2001.

[27] Juliane Dehnert and Armin Zimmermann. On the Suitability of Correctness Criteria for Business Process Models. In *Business Process Management*, volume 3649 of *LNCS*, pages 386–391. Springer, 2005.

[28] Jörg Desel. Teaching system modeling, simulation and validation. In *Winter simulation*, pages 1669–1675. International Society for Computer Simulation, 2000.

[29] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and analysis of business process models in BPMN. *Information and Software Technology*, 50(12):1281–1294, November 2008.

[30] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[31] Marlon Dumas, Alexander Grosskopf, Thomas Hettel, and Moe Wynn. Semantics of standard process models with OR-joins. In *On the Move to Meaningful Internet Systems*, volume 4803 of *LNCS*, pages 41–58. Springer, 2007.

[32] Marlon Dumas, Marcello La Rosa, Jan Mendling, Raul Mäesalu, Hajo A. Reijers, and Nataliia Semenenko. Understanding Business Process Models: The Costs and Benefits of Structuredness. In *Advanced Information Systems Engineering*, pages 31–46. Springer, 2012.

[33] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of business process management.* Springer, 2013.

[34] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management, Second Edition.* Springer, 2018.

[35] Nissreen El-Saber and Artur Boronat. BPMN Formalization and Verification Using Maude. In *Workshop on Behaviour Modelling-Foundations and Applications*, pages 1–12. ACM, 2014.

[36] Nissreen AS El-Saber. *CMMI-CM compliance checking of formal BPMN models using Maude.* PhD thesis, Department of Computer Science, 2015.

[37] Dirk Fahland and Hagen Völzer. Dynamic Skipping and Blocking and Dead Path Elimination for Cyclic Workflows. In *Business Process Management*, volume 9850 of *LNCS*, pages 234–251. Springer, 2016.

[38] Cédric Favre and Hagen Völzer. Symbolic execution of acyclic workflow graphs. *Business Process Management*, pages 260–275, 2010.

[39] Cédric Favre and Hagen Völzer. The difficulty of replacing an inclusive OR-join. In *International Conference on Business Process Management*, volume 7481 of *LNCS*, pages 156–171. Springer, 2012.

[40] Michael Hahn, Uwe Breitenbücher, Oliver Kopp, and Frank Leymann. Modeling and execution of data-aware choreographies: an overview. *Computer Science-Research and Development*, pages 1–12, 2017.

[41] Andreas Hermann, Hendrik Scholta, Sebastian Bräuer, and Jörg Becker. Collaborative Business Process Management - A Literature-based Analysis of Methods for Supporting Model Understandability. In *Towards Thought Leadership in Digital Transformation: 13. Internationale Tagung Wirtschaftsinformatik*, 2017.

[42] Wenjia Huai, Xudong Liu, and Hailong Sun. Towards Trustworthy Composite Service Through Business Process Model Verification. In *Ubiquitous Intelligence & Computing and Autonomic & Trusted Computing*, pages 422–427. IEEE, 2010.

[43] Ahmed Kheldoun, Kamel Barkaoui, and Malika Ioualalen. Formal verification of complex business processes based on high-level Petri nets. *Information Sciences*, 385-386:39–54, 2017.

[44] Bartek Kiepuszewski, Arthur H.M. ter Hofstede, and Christoph J. Bussler. On structured workflow modelling. In *International Conference on Advanced Information Systems Engineering*, volume 1789 of *LNCS*, pages 431–445. Springer, 2000.

[45] David Knuplesch, Rüdiger Pryss, and Manfred Reichert. Data-aware interaction in distributed and collaborative workflows: Modeling, semantics, correctness. In *Collaborative Computing: Networking, Applications and Worksharing 2012*, pages 223–232. IEEE, 2012.

[46] Ryszard Koniewski, Andrzej Dzielinski, and Krzysztof Amborski. Use of Petri Nets and Business Processes Management Notation in Modelling and Simulation of Multimodal Logistics Chains. In *European Conference on Modeling and Simulation*, pages 28–31, 2006.

[47] Felix Kossak, Christa Illibauer, Verena Geist, Jan Kubovy, Christine Natschläger, Thomas Ziebermayr, Theodorich Kopetzky, Bernhard Freudenthaler, and Klaus-Dieter Schewe. A Rigorous Semantics for BPMN 2.0 Process Diagrams. In *A Rigorous Semantics for BPMN 2.0 Process Diagrams*, pages 29–152. Springer, 2014.

[48] Matthias Kunze, Philipp Berger, and Mathias Weske. BPM Academic Initiative - Fostering Empirical Research. In *Business Process Management (Demo Track)*, volume 940, pages 1 – 5, 2012.

[49] Matthias Kunze, Alexander Luebbe, Matthias Weidlich, and Mathias Weske. Towards understanding process modeling – the case of the bpm academic initiative. In Remco Dijkman, Jörg Hofstetter, and Jana Koehler, editors, *Business Process Model and Notation*, pages 44–58, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[50] Matthias Kunze and Mathias Weske. *Behavioural Models, Testo originale: From Modelling Finite Automata to Analysing Business Processes*. Springer International Publishing, 2016.

[51] Ann Lindsay, Denise Downs, and Ken Lunn. Business processes—attempts to find a definition. *Information and Software Technology*, 45(15):1015–1019, 2003.

[52] María Teresa Gómez López et al. Guiding the creation of choreographed processes with multiple instances based on data models. In *BPMWorkshops*, volume 281 of *LNBIP*, pages 239–251, 2016.

[53] Thomas M. Prinz and Wolfram Amme. A Complete and the Most Liberal Semantics for Converging OR Gateways in Sound Processes. *Complex Systems Informatics and Modeling Quarterly*, (4):32–49, 2015.

[54] Jan Mendling. *Detection and prediction of errors in EPC business process models*. PhD thesis, Wirtschaftsuniversität Wien Vienna, 2007.

[55] Jan Mendling, Hajo A. Reijers, and Wil M.P. van der Aalst. Seven process modeling guidelines (7pmg). *Information and Software Technology*, 52(2):127–136, 2010.

[56] Jan Mendling, Laura Sanchez-Gonzalez, Felix Garcia, and Marcello La Rosa. Thresholds for error probability measures of business process models. *Journal of Systems and Software*, 85(5):1188–1197, 2012.

[57] Andreas Meyer. *Data perspective in business process management*. 2015.

[58] Andreas Meyer, Luise Pufahl, Kimon Batoulis, Dirk Fahland, and Mathias Weske. Automating data exchange in process choreographies. *Information Systems*, 53:296–329, 2015.

[59] Andreas Meyer, Luise Pufahl, Dirk Fahland, and Mathias Weske. *Modeling and enacting complex data dependencies in business processes*. Springer, 2013.

[60] Andreas Meyer et al. Data perspective in process choreographies: modeling and execution. In *Techn. Ber. BPM Center Report BPM-13-29. BPMcenter. org*, 2013.

[61] Mariusz Momotko and Bartosz Nowicki. Visualisation of (distributed) process execution based on extended BPMN. In *DEXA*, pages 280–284. IEEE, 2003.

[62] Marco Montali and Diego Calvanese. Soundness of data-aware, case-centric processes. *International Journal on Software Tools for Technology Transfer (STTT)*, 18(5):535–558, 2016.

[63] Isel Moreno-Montes de Oca and Monique Snoeck. Pragmatic guidelines for business process modeling. Technical Report 2592983, KU Leuven, Faculty of Economics and Business - Management Information Systems Group, 2014.

[64] Michael zur Muehlen and Jan Recker. How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In *Advanced Information Systems Engineering*, volume 5074, pages 465–479. Springer, 2008.

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

[66] Rocco De Nicola. A gentle introduction to process algebras ? 2013.

[67] OASIS WSBPEL TC. Web Services Business Process Execution Language Version 2.0. Technical report, OASIS, April 2007. Available at http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[68] OMG. Business Process Model and Notation (BPMN V 2.0), 2011.

[69] Oscar Pastor. Model-Driven Development in Practice: From Requirements to Code. In *SOFSEM 2017: Theory and Practice of Computer Science*, volume 10139 of *LNCS*, pages 405–410. Springer, 2017.

[70] Philipp Fromme and Sebastian Warnke and Patrick Dehn. bpmn-js Token Simulation, 2017. https://github.com/bpmn-io/bpmn-js-token-simulation.

[71] Gordon D. Plotkin. A structural approach to operational semantics. *The Journal of Logic and Algebraic Programming*, 60(61):17–139, 2004.

[72] Artem Polyvyanyy and Christoph Bussler. The structured phase of concurrency. In *Seminal Contributions to Information Systems Engineering*, pages 257–263. Springer, 2013.

[73] Artem Polyvyanyy, Luciano Garcia-Banuelos, Dirk Fahland, and Mathias Weske. Maximal Structuring of Acyclic Process Models. *The Computer Journal*, 57(1):12–35, 2014.

[74] Artem Polyvyanyy, Luciano García-Bañuelos, and Marlon Dumas. Structuring acyclic process models. *Information Systems*, 37(6):518–538, 2012.

[75] Davide Prandi, Paola Quaglia, and Nicola Zannone. Formal Analysis of BPMN Via a Translation into COWS. In *Coordination Models and Languages*, volume 5052 of *LNCS*, pages 249–263. Springer, 2008.

[76] Thomas M. Prinz. Fast soundness verification of workflow graphs. *Services and their Composition*, pages 31–38, 2013.

[77] Rosario Pugliese and Francesco Tiezzi. A calculus for orchestration of web services. *Journal of Applied Logic*, 10(1):2–31, 2012.

[78] Frank Puhlmann. Soundness Verification of Business Processes Specified in the Pi-Calculus. In *On the Move to Meaningful Internet Systems*, volume 4803 of *LNCS*, pages 6–23. Springer, 2007.

[79] Frank Puhlmann and Mathias Weske. Interaction soundness for service orchestrations. In *Service-Oriented Computing*, volume 4294 of *LNCS*, pages 302–313. Springer, 2006.

[80] Frank Puhlmann and Mathias Weske. Investigations on Soundness Regarding Lazy Activities. In *Business Process Management*, volume 4102 of *LNCS*, pages 145–160. Springer, 2006.

[81] Frank Puhlmann and Mathias Weske. Investigations on soundness regarding lazy activities. In *Business Process Management*, volume 4102 of *LNCS*, pages 145–160. Springer, 2006.

[82] Anass Rachdi, Abdeslam En-Nouaary, and Mohamed Dahchour. Dataflow analysis in bpmn models. In *ICEIS*, volume 2, pages 229–237. SciTePress, 2017.

[83] Mohamed Ramadan, Hicham G. Elmongui, and Riham Hassan. BPMN Formalisation using Coloured Petri Nets. In *GSTF International Conference on Software Engineering & Applications*, 2011.

[84] Red Hat. jbpm v.7.0.0, www.jBPM.org.

[85] Manfred Reichert and Barbara Weber. *Enabling flexibility in process-aware information systems: challenges, methods, technologies*. Springer Science & Business Media, 2012.

[86] Jorge Roa, Omar Chiotti, and Pablo Villarreal. A verification method for collaborative business processes. In *International Conference on Business Process Management*, pages 293–305. Springer, 2011.

[87] Grzegorz Rozenberg and Joost Engelfriet. Elementary net systems. In *Lectures on Petri Nets I: Basic Models*, pages 12–121. Springer, 1998.

[88] Nick Russell, Wil M.P. van der Aalst, Arthur H. M. ter Hofstede, and David Edmond. Workflow resource patterns: Identification, representation and tool support. In *Advanced Information Systems Engineering*, volume 3520 of *LNCS*, pages 216–232. Springer, 2005.

[89] Natalia Sidorova, Christian Stahl, and Nikola Trčka. Workflow soundness revisited: Checking correctness in the presence of data while staying conceptual. In *International Conference on Advanced Information Systems Engineering*, volume 6051 of *LNCS*, pages 530–544. Springer, 2010.

[90] Anna Suchenia, Tomasz Potempa, Antoni Ligęza, Krystian Jobczyk, and Krzysztof Kluza. Selected Approaches Towards Taxonomy of Business Process Anomalies. In *Advances in Business ICT: New Ideas from Ongoing Research*, pages 65–85. Springer, 2017.

[91] Jayme L. Szwarcfiter and Peter E. Lauer. A search strategy for the elementary cycles of a directed graph. *BIT Numerical Mathematics*, 16(2):192–204, 1976.

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

[93] Bernhard Thalheim, Ove Sorensen, and Egon Borger. On Defining the Behavior of OR-joins in Business Process Models. *Journal of Universal Computer Science*, 15(1):3 – 32, 2009.

[94] Petia Van Der Aalst, M P, Wil Dumas, Ter Hofstede, Arthur H M And, Petia Wohed, Wil M. P. Aalst, Marlon Dumas, Arthur Ter, and Nick Russell. Pattern-based analysis of the control-flow perspective of uml activity diagrams. *Nick*, pages 63–78, 01 2005.

[95] Wil M. P. Van Der Aalst, Arthur H. M. Ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.

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

[97] Wil M.P. van der Aalst. Verification of workflow nets. In *International Conference on Application and Theory of Petri Nets*, volume 1248 of *LNCS*, pages 407–426. Springer, 1997.

[98] Wil M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems and Computers*, 08(01):21–66, 1998.

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

[100] Wil M.P. van der Aalst. Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In *Business Process Management*, volume 1806 of *LNCS*, pages 161–183. Springer, 2000.

[101] Wil M.P. van der Aalst, Jörg Desel, and Ekkart Kindler. On the semantics of EPCs: A vicious circle. In *BUSINESS PROCESS MANAGEMENT USING EPCS*, pages 71–79, 2002.

[102] Wil M.P. van der Aalst, Arjan J. Mooij, Christian Stahl, and Karsten Wolf. Service interaction: Patterns, formalization, and analysis. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, volume 5569 of *LNCS*, pages 42–88. Springer, 2009.

[103] Wil M.P. Van Der Aalst and Arthur HM Ter Hofstede. YAWL: yet another workflow language. *Information systems*, 30(4):245–275, 2005.

[104] Wil M.P. van der Aalst, Kees M. van Hee, Arthur H.M. ter Hofstede, Natalia Sidorova, H.M.W. Verbeek, Marc Voorhoeve, and Moe Thandar Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011.

[105] Pieter Van Gorp and Remco Dijkman. A visual token-based formalization of BPMN 2.0 based on in-place transformations. *Information and Software Technology*, 55(2):365–394, 2013.

[106] Kees van Hee, Olivia Oanea, Alexander Serebrenik, Natalia Sidorova, and Marc Voorhoeve. History-based joins: Semantics, soundness and implementation. In *International Conference on Business Process Management*, pages 225–240. Springer, 2006.

[107] Hagen Völzer. A new semantics for the inclusive converging gateway in safe processes. In *Business Process Management*, volume 6336 of *LNCS*, pages 294–309. Springer, 2010.

[108] Hagen Völzer. Faster Or-Join Enactment for BPMN 2.0. In *Business Process Model and Notation*, volume 95, pages 31–43. Springer, 2011.

[109] Mathias Weske. *Business Process Management: concepts, languages, architectures*. Springer, 2012.

[110] Branimir. Wetzstein, Zhilei Ma, Agata Filipowska, Monika Kaczmarek, Sami Bhiri, Silvestre Losada, Jose-Manuel Lopez-Cobo, and Laurent Cicurel. Semantic business process management: A lifecycle based requirements analysis. In *Workshops on Semantic Business Process and Product Lifecycle Management*, volume 251, pages 1–11. CEUR-WS.org, 2007.

[111] Petia Wohed, Wil MP van der Aalst, Marlon Dumas, Arthur HM ter Hofstede, and Nick Russell. Pattern-based analysis of UML activity diagrams. *Beta, Research School for Operations Management and Logistics, Eindhoven*, 2004.

[112] Peter Y. H. Wong and Jeremy Gibbons. A Process Semantics for BPMN. In *Formal Methods and Software Engineering*, volume 5256 of *LNCS*, pages 355–374. Springer, 2008.

[113] Peter Y.H. Wong and Jeremy Gibbons. Formalisations and applications of BPMN. *Science of Computer Programming*, 76(8):633–650, 2011.

[114] Moe Thandar Wynn, H. M. W. Verbeek, Wil M.P. van der Aalst, Arthur H.M. ter Hofstede, and David Edmond. Business process verification-finally a reality! *Business Process Management Journal*, 15(1):74–92, 2009.

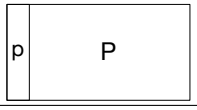
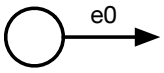
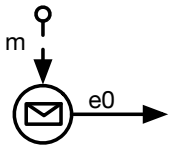
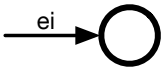
[115] Jianhong Ye and Wen Song. Transformation of BPMN Diagrams to YAWL Nets. *Journal of Software*, 5(4):396–404, 2010.

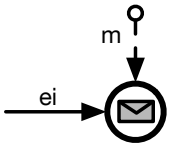
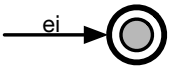
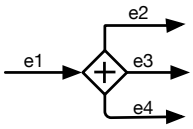
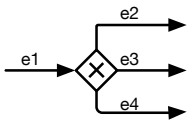
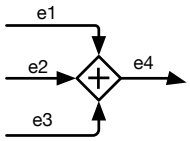
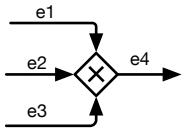
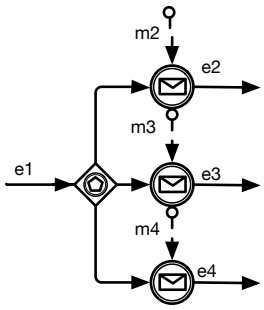
Appendix A

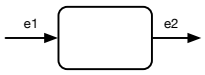
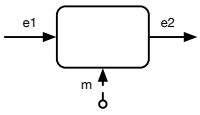
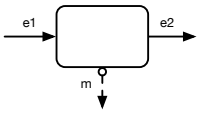
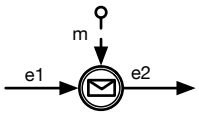
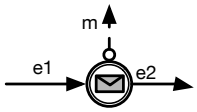
Appendix: Notation Correspondence

A.1 Core BPMN

This appendix reports the complete correspondence between the BPMN graphical notation and the syntax presented in Section 3.2. For the sake of presentation, join and split gateways include only three incoming/outgoing branching respectively.

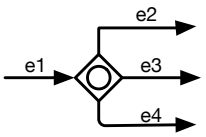
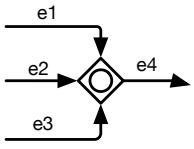
Graphical Notation	Textual Notation
	$\mathsf{pool}(\mathsf{p}, P)$
	$\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e0})$
	$\mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e0})$
	$\mathsf{end}(\mathsf{ei}, \mathsf{e}_{cmp})$

Graphical Notation	**Textual Notation**
	endSnd(ei, m, e_{cmp})
	terminate(ei)
	andSplit(e1, {e2, e3, e4})
	xorSplit(e1, {e2, (e3, e4})
	andJoin({e1, e2, e3}, e4)
	xorJoin({e1, e2, e3}, e4)
	eventBased(e1, (m2, e2), (m3, e3), (m4, e4))

Graphical Notation	Textual Notation
	task(e1, e2)
	taskRcv(e1, m, e2)
	taskSnd(e1, m, e2)
	interRcv(e1, m, e2)
	interSnd(e1, m, e2)

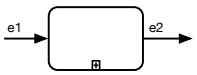
A.2 OR-Join

This appendix reports the correspondence between the BPMN graphical notation for the OR gateways and the syntax presented in Section 5.3.

Graphical Notation	Textual Notation
	orSplit(e1, {e2, (e3, e4})
	orJoin({e1, e2, e3}, e4)

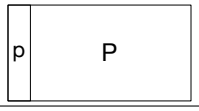
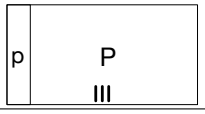
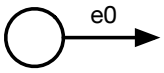
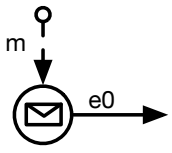
A.3 Sub-Processes

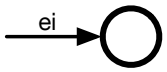
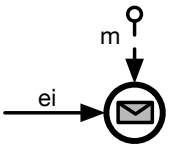
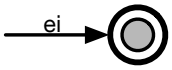
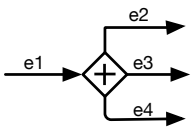
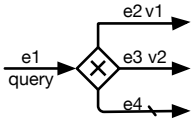
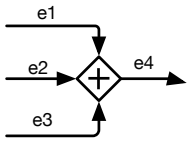
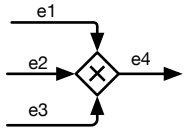
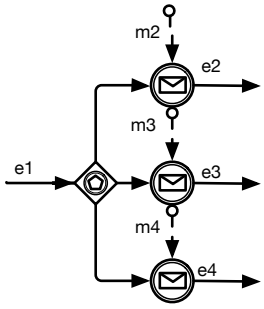
This appendix reports the correspondence between the BPMN graphical notation for the sub-process element and the syntax presented in Section 6.2.

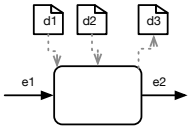
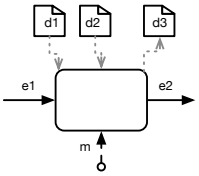
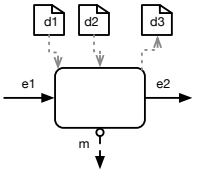
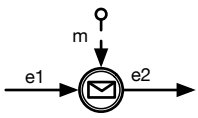
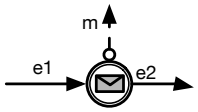
Graphical Notation	Textual Notation
	$\mathsf{subProc}(\mathsf{e1}, P, \mathsf{e2})$

A.4 Multiple Instances and Data

This appendix reports the complete correspondence between the BPMN graphical notation and the syntax presented in Section 7.2.

Graphical Notation	Textual Notation
	$\mathsf{pool}(\mathsf{p}, P)$
	$\mathsf{miPool}(\mathsf{p}, P)$
	$\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e0})$
	$\mathsf{startRcv}(\mathsf{m}:\tilde{\mathsf{t}}, \mathsf{e0})$

Graphical Notation	**Textual Notation**
	$\mathsf{end}(\mathsf{ei}, \mathsf{e}_{cmp})$
	$\mathsf{endSnd}(\mathsf{ei}, \mathsf{m} : \tilde{\mathsf{exp}}, \mathsf{e}_{cmp})$
	$\mathsf{terminate}(\mathsf{ei})$
	$\mathsf{andSplit}(\mathsf{e1}, \{\mathsf{e2}, \mathsf{e3}, \mathsf{e4}\})$
	$\mathsf{xorSplit}(\mathsf{e1}, \{(\mathsf{e2}, \mathsf{query} = \mathsf{v}_1), (\mathsf{e3}, \mathsf{query} = \mathsf{v}_2), (\mathsf{e4}, \mathsf{default})\})$
	$\mathsf{andJoin}(\{\mathsf{e1}, \mathsf{e2}, \mathsf{e3}\}, \mathsf{e4})$
	$\mathsf{xorJoin}(\{\mathsf{e1}, \mathsf{e2}, \mathsf{e3}\}, \mathsf{e4})$
	$\mathsf{eventBased}(\mathsf{e1}, (\mathsf{m2} : \tilde{\mathsf{t}}_2, \mathsf{e2}), (\mathsf{m3} : \tilde{\mathsf{t}}_3, \mathsf{e3}), (\mathsf{m4} : \tilde{\mathsf{t}}_4, \mathsf{e4}))$

Graphical Notation	**Textual Notation**
	$\mathsf{task}(\mathsf{e1}, \mathsf{exp}(\mathsf{d}_1, \mathsf{d}_2))(\mathsf{d}_3.\mathsf{f}_1 := \mathsf{exp}_1, \ldots, \mathsf{d}_3.\mathsf{f}_n := \mathsf{exp}_n), \mathsf{e2})$
	$\mathsf{taskRcv}(\mathsf{e1}, \mathsf{exp}(\mathsf{d}_1, \mathsf{d}_2), (\mathsf{d}_3.\mathsf{f}_1 := \mathsf{exp}_1, \ldots, \mathsf{d}_3.\mathsf{f}_n := \mathsf{exp}_n), \mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e2})$
	$\mathsf{taskSnd}(\mathsf{e1}, \mathsf{exp}(\mathsf{d}_1, \mathsf{d}_2), (\mathsf{d}_3.\mathsf{f}_1 := \mathsf{exp}_1, \ldots, \mathsf{d}_3.\mathsf{f}_n := \mathsf{exp}_n), \mathsf{m} : \tilde{\mathsf{exp}}, \mathsf{e2})$
	$\mathsf{interRcv}(\mathsf{e1}, \mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e2})$
	$\mathsf{interSnd}(\mathsf{e1}, \mathsf{m} : \tilde{\mathsf{exp}}, \mathsf{e2})$

Appendix B

Appendix: Definitions

This appendix reports the complete definitions of some auxiliary notions used in the thesis. The various notions are here defined for the core set of BPMN elements. However, all the definitions extend naturally to the other frameworks.

Function $edges(P)$ refers to the edges in the scope of P and $edgesEl(P)$ to indicate the edges in the scope of P without considering the spurious edges.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Functions $in(P)$ and $out(P)$ are inductively defined. They are used to 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{m},\mathsf{e}')) = \varnothing & out(\mathsf{startRcv}(\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))) \\
\quad = \{\mathsf{e}\} & \quad = \{\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{e}',\mathsf{m})) = \{\mathsf{e}\} & out(\mathsf{interRcv}(\mathsf{e},\mathsf{e}',\mathsf{m})) = \{\mathsf{e}'\} \\
in(\mathsf{interSnd}(\mathsf{e},\mathsf{e}',\mathsf{m})) = \{\mathsf{e}\} & out(\mathsf{interSnd}(\mathsf{e},\mathsf{e}',\mathsf{m})) = \{\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)) \\
\quad \setminus (out(P_1) \cup out(P_2)) & \quad \setminus (in(P_1) \cup in(P_2))
\end{array}
$$

Definition 38 (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:*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Definition 39 (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* $\sigma(\mathsf{e}') = 1$

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

$isCompleteEl(\mathsf{interSnd}(\mathsf{e}, \mathsf{e}', \mathsf{m}), \sigma)$ *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 $\forall \mathsf{e}'' \in \{\mathsf{e}_{o1}, \ldots, \mathsf{e}_{ok}\} \backslash \mathsf{e}' \;.\; \sigma(\mathsf{e}'') = 0$

$isCompleteEl(P_1 \| P_2, \sigma)$ *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*:*

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

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

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

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

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

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

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

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

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

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

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

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

Appendix C

Appendix: Proofs

C.1 Core BPMN

This appendix reports the proofs of the results presented in Chapter 4.

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 $isInit(P, \sigma)$, it follows 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, it follows that $\forall \mathsf{e} \in edgesEl(P)\ .\ \sigma(\mathsf{e}) \leqslant 1$, which allows to conclude. □

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

Proof. 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 $\langle P, \sigma \rangle \xrightarrow{\ell} \sigma'$ is *P-Task*. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$; hence $0 < \sigma(\mathsf{e}) \leqslant 1$, i.e. $\sigma(\mathsf{e}) = 1$. It can be exploited the fact that $\langle P, \sigma \rangle$ 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, it follows that $\forall \mathsf{e}''' \in edgesEl(P)\ .\ \sigma'(\mathsf{e}''') \leqslant 1$, which allows 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 $\langle P, \sigma \rangle \xrightarrow{\ell} \sigma'$ is *P-TaskRcv*. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$; hence $0 < \sigma(\mathsf{e}) \leqslant 1$,

i.e. $\sigma(\mathsf{e}) = 1$. It can be exploited the fact that $\langle P, \sigma\rangle$ 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, it follows that $\forall \mathsf{e}''' \in edgesEl(P)\ .\ \sigma'(\mathsf{e}''') \leqslant 1$, which allows 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 $\langle P, \sigma\rangle \xrightarrow{\ell} \sigma'$ is *P-TaskSnd*. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$; hence $0 < \sigma(\mathsf{e}) \leqslant 1$, i.e. $\sigma(\mathsf{e}) = 1$. It can be exploited the fact that $\langle P, \sigma\rangle$ 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, it follows that $\forall \mathsf{e}''' \in edgesEl(P)\ .\ \sigma'(\mathsf{e}''') \leqslant 1$, which allows to conclude.

- $P = \mathsf{interRcv}(\mathsf{e}, \mathsf{e}', \mathsf{m})$. By hypothesis $\langle P, \sigma\rangle$ is cs-safe, then $edgesEl(P) = edgesEl(\mathsf{interRcv}(\mathsf{e}, \mathsf{e}', \mathsf{m})) = \{\mathsf{e}, \mathsf{e}'\}$ 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 $\langle P, \sigma\rangle \xrightarrow{\ell} \sigma'$ is *P-InterRcv*. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$; hence $0 < \sigma(\mathsf{e}) \leqslant 1$, i.e. $\sigma(\mathsf{e}) = 1$. It can be exploited the fact that $\langle P, \sigma\rangle$ 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, it follows that $\forall \mathsf{e}''' \in edgesEl(P)\ .\ \sigma'(\mathsf{e}''') \leqslant 1$, which allows to conclude.

- $P = \mathsf{interSnd}(\mathsf{e}, \mathsf{e}', \mathsf{m})$. By hypothesis $\langle P, \sigma\rangle$ is cs-safe, then $edgesEl(P) = edgesEl(\mathsf{interSnd}(\mathsf{e}, \mathsf{e}', \mathsf{m})) = \{\mathsf{e}, \mathsf{e}'\}$ 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 $\langle P, \sigma\rangle \xrightarrow{\ell} \sigma'$ is *P-InterSnd*. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$; hence $0 < \sigma(\mathsf{e}) \leqslant 1$, i.e. $\sigma(\mathsf{e}) = 1$. It can be exploited the fact that $\langle P, \sigma\rangle$ 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, it follows that $\forall \mathsf{e}''' \in edgesEl(P)\ .\ \sigma'(\mathsf{e}''') \leqslant 1$, which allows 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 $\langle P, \sigma\rangle \xrightarrow{\ell} \sigma'$ is *P-Empty*. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$; hence $0 < \sigma(\mathsf{e}) \leqslant 1$, i.e. $\sigma(\mathsf{e}) = 1$. It can be exploited the fact that $\langle P, \sigma\rangle$ 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, it follows that $\forall \mathsf{e}''' \in edgesEl(P)\ .\ \sigma'(\mathsf{e}''') \leqslant 1$, which allows to conclude.

Inductive cases:

- $\langle \mathsf{andSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{andJoin}(E', \mathsf{e}'), \sigma \rangle$, 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-AndSplit*. It can be exploited 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 it easily follows by inductive hypothesis.
  - Node $P_1 \parallel \ldots \parallel P_n$ evolves and affects the split and/or join gateways. In this case one 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*. It can be exploited 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.
- $\langle \mathsf{xorSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E', \mathsf{e}''), \sigma \rangle$, 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*. It can be exploited 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 it easily follows by inductive hypothesis.

- Node $P_1 \parallel \ldots \parallel P_n$ evolves and affects the split and/or join gateways. In this case one 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*. It can be exploited 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 $\langle \mathsf{xorSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E', \mathsf{e}''), \sigma \rangle$ 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.

- $\mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j, \mathsf{e}'_j) | j \in [1..n]\}) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E, \mathsf{e}'')$, 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*. It can be exploited 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 $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}'_j)$. $\forall \mathsf{e}''' \in edgesEl(\mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j, \mathsf{e}'_j | j \in [1..n]\}))$. $\sigma(\mathsf{e}''') \leqslant 1$. 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 it easily follows by inductive hypothesis.
  - Node $P_1 \parallel \ldots \parallel P_n$ evolves and affects the split and/or join gateways. In this case one 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*. It can be exploited 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 $\langle \mathsf{xorSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E, \mathsf{e}''), \sigma \rangle$ 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.

- $\mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{\mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\})$ 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*. It can be exploited 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 $\sigma(\mathsf{e}''') = 1$; assuming that the marking is $\sigma(\mathsf{e}''') = 1$ (since the other case is similar), then $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}'), \sigma \rangle \xrightarrow{\epsilon} \sigma'$ 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 $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{\mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\}), \sigma \rangle$ 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 $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{\mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\}), \sigma' \rangle$ is cs-safe.

  - Node $P_1 \parallel P_2$ evolves without affecting the split and join gateways. In this case it easily follows by inductive hypothesis.

  - Node $P_1 \parallel P_2$ evolves and affects the xor join and xor split gateways. In this case one can reason like in the first case, by relying on inductive hypothesis.

  - $\langle \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{\mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\}), \sigma \rangle$ evolves by means of rule *P-XorSplit*. It can be exploited the fact that this is a core reachable well-structured configuration to prove that the term is marked as $\sigma(\mathsf{e}^{\mathsf{iv}}) = 1$. Hence, it evolves in a cs-safe term; in fact assuming that it evolves in this way $\langle \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{\mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\}), \sigma \rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}})$, hence, $edgesEl(\mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{\mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\})) = \{\mathsf{e}^{\mathsf{iv}}, \mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\}$ and $\sigma'(\mathsf{e}^{\mathsf{iv}}) = 0, \sigma'(\mathsf{e}^{\mathsf{v}}) = 1, \sigma'(\mathsf{e}^{\mathsf{vi}}) = 0$, that is $\forall \mathsf{e} \in edgesEl(\mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{\mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\})) \,.\, \sigma'(\mathsf{e}) \leqslant 1$. By hypothesis $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{\mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\}), \sigma \rangle$ is cs-safe, i.e. if there is a token on e^{v} in the state $\langle \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{\mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\}), \sigma' \rangle$ all the other edges do not have token. This means that cs-safeness is not affected. Therefore, the overall term $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{\mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\}), \sigma' \rangle$ is cs-safe.

- Be $\langle P, \sigma \rangle = \langle P_1 \parallel P_2, \sigma \rangle$. The relevant case for cs-safeness is when P evolves by applying *P-Int*$_1$. It results that $\langle P_1 \parallel P_2, \sigma \rangle \xrightarrow{\ell} \sigma'$ with $\langle P_1, \sigma \rangle \xrightarrow{\ell} \sigma'$. By definition of $edgesEl(\cdot)$ function it follows that $edgesEl(P) = edgesEl(P_1) \cup edgesEl(P_2)$. By inductive hypothesis it follows 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 to conclude.

□

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

Proof. According to Definition 4, P can have 6 different forms. The proof is given by case analysis on the parallel component of $\langle P, \sigma\rangle$ that causes the transition $\langle P, \sigma\rangle \xrightarrow{\ell} \sigma'$.

It will be considered now the case P= $\mathsf{start}(\mathsf{e}, \mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'', \mathsf{e}''')$.

- $\mathsf{start}(\mathsf{e}, \mathsf{e}')$ evolves by means of the rule *P-Start*. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$, hence, by cs-safeness, $\sigma(\mathsf{e}) = 1$. It can be exploited 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, $\langle P, \sigma'_1\rangle = \langle \mathsf{start}(\mathsf{e}, \mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'', \mathsf{e}'''), \sigma'_1\rangle$ 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 can be exploited 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, $\langle P, \sigma'_1\rangle$ can only evolve by applying *P-Int₁* 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 it can be concluded that $\langle P, \sigma'\rangle$ is cs-safe.

- P' moves, that is $\langle P', \sigma\rangle \xrightarrow{\ell} \sigma'$. 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, hence $edgesEl(\mathsf{start}(\mathsf{e}, \mathsf{e}')) = \{\mathsf{e}'\}$ is such that $\sigma'(\mathsf{e}') \leqslant 1$ and $edgesEl(\mathsf{end}(\mathsf{e}'', \mathsf{e}''')) = \{\mathsf{e}''\}$ is such that $\sigma'(\mathsf{e}'') \leqslant 1$. Thus, it can be concluded that $\langle P, \sigma'\rangle$ is safe.

It is considered now 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 it can be applied is *P-Terminate*. It can be exploited 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 I have $\langle \mathsf{terminate}(\mathsf{e}''), \sigma\rangle \xrightarrow{kill} \sigma'_1$ with $\sigma'_1 = dec(\sigma, \mathsf{e}'')$. Now, $\langle P, \sigma'_1\rangle$ can only evolve by applying *P-Kill₁* producing $\langle P, \sigma'\rangle$ where σ' is completed unmarked; therefore it is cs-safe.

- P' moves: similar to the previous case.

Now it is considered 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 that can be applied is *P-EndSnd*. It can be exploited 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 it follows that $\langle \mathsf{endSnd}(\mathsf{e}'', \mathsf{m}, \mathsf{e}'''), \sigma \rangle \xrightarrow{!\mathsf{m}} \sigma_1'$ with $\sigma_1' = inc(dec(\sigma, \mathsf{e}''), \mathsf{e}''')$ Now, $\langle P, \sigma_1' \rangle$ can only evolve by applying *P-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 it can be concluded that $\langle P, \sigma' \rangle$ is cs-safe.
- P' moves: similar to the previous cases.

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

- $\mathsf{startRcv}(\mathsf{m}, \mathsf{e}')$ 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, $\sigma(\mathsf{e}) = 1$. It can be exploited the fact that this is a reachable well-structured configuration to prove that $\sigma(\mathsf{e}') = 0$. The rule produces the following transition $\langle \mathsf{startRcv}(\mathsf{m}, \mathsf{e}'), \sigma \rangle \xrightarrow{?\mathsf{m}} \sigma_1'$ with $\sigma_1' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$ where $\sigma_1'(\mathsf{e}) = 0$ and $\sigma_1'(\mathsf{e}') = 1$. Now, $\langle P, \sigma_1' \rangle = \mathsf{startRcv}(\mathsf{m}, \mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'', \mathsf{e}'''), \sigma_1' \rangle$ can evolve only through the application of *P-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-End*. It follows as in the first case.
- P' moves, that is $\langle P', \sigma \rangle \xrightarrow{\ell} \sigma'$. 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, thus $edgesEl(\mathsf{startRcv}(\mathsf{m}, \mathsf{e}')) = \{\mathsf{e}'\}$ is such that $\sigma'(\mathsf{e}') \leqslant 1$ and $edgesEl(\mathsf{end}(\mathsf{e}'', \mathsf{e}''')) = \{\mathsf{e}''\}$ is such that $\sigma'(\mathsf{e}'') \leqslant 1$. It can be concluded that $\langle P, \sigma' \rangle$ is safe.

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

- $\mathsf{startRcv}(\mathsf{m}, \mathsf{e}')$ evolves by means of the rule *P-StartRcv*: like in the previous case.
- The end terminate event evolves: the only transition it can be applied 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 $\langle P', \sigma \rangle \xrightarrow{\ell} \sigma'$. 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 thus $edgesEl(\mathsf{startRcv}(\mathsf{m}, \mathsf{e}')) = \{\mathsf{e}'\}$ is such that $\sigma'(\mathsf{e}') \leqslant 1$ and $edgesEl(\mathsf{terminate}(\mathsf{e}'')) = \{\mathsf{e}''\}$ is such that $\sigma'(\mathsf{e}'') \leqslant 1$. It follows that $\langle P, \sigma' \rangle$ is safe.

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

- $\mathsf{startRcv}(\mathsf{m},\mathsf{e}')$ 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 $\langle P',\sigma\rangle \xrightarrow{\ell} \sigma'$. By Lemma 2 $\langle P',\sigma'\rangle$ is safe, thus $\forall \mathsf{e} \in \mathit{edgesEl}(P')\ .\ \sigma'(\mathsf{e}) \leqslant 1$. By hypothesis, P is cs-safe thus $\mathit{edgesEl}(\mathsf{startRcv}(\mathsf{m},\mathsf{e}')) = \{\mathsf{e}'\}$ is such that $\sigma'(\mathsf{e}') \leqslant 1$ and $\mathit{edgesEl}(\mathsf{endSnd}(\mathsf{e}'',\mathsf{m},\mathsf{e}''')) = \{\mathsf{e}''\}$ is such that $\sigma'(\mathsf{e}'') \leqslant 1$. It follows that $\langle P,\sigma'\rangle$ is safe.

□

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

Proof. It has to be shown that if $\langle P,\sigma\rangle \rightarrow^* \sigma'$ then $\langle P,\sigma'\rangle$ is cs-safe. It proceeds 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 $\mathit{isInit}(P,\sigma')$ is satisfied. By Lemma 1 it can be concluded $\langle P,\sigma'\rangle$ is cs-safe.
Inductive Case: In this case $\langle P,\sigma\rangle \rightarrow^* \langle P,\sigma''\rangle \xrightarrow{\ell} \langle P,\sigma'\rangle$ for some process $\langle P,\sigma''\rangle$. By induction, $\langle P,\sigma''\rangle$ is cs-safe. By applying Lemma 3 to $\langle P,\sigma''\rangle \xrightarrow{\ell} \langle P,\sigma'\rangle$, it can be concluded that $\langle P,\sigma'\rangle$ is cs-safe. □

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

Proof. By contradiction, let C be well-structured and C be unsafe. By Definition 8, given σ and δ such that $\mathit{isInit}(C,\sigma,\delta)$ there exists a collaboration configuration $\langle C,\sigma',\delta'\rangle$ such that $\langle C,\sigma,\delta\rangle \rightarrow^* \langle C,\sigma',\delta'\rangle$ and $\exists P \in \mathit{participant}(C)$, $\langle P,\sigma'\rangle$ not cs-safe. From hypothesis $\mathit{isInit}(C,\sigma,\delta)$, it follows $\mathit{isInit}(P,\sigma)$. Thus, also $\langle P,\sigma'\rangle$ is reachable. From hypothesis C is well-structured, it follows 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 $\mathit{isWSCore}(P)$ and let $\langle P,\sigma\rangle$ be core reachable, then there exists σ' such that $\langle P,\sigma\rangle \rightarrow^* \sigma'$ and $\mathit{isCompleteEl}(P,\sigma')$.*

Proof. It follows by induction on the structure of $\mathit{isWSCore}(P)$. Base cases: by definition of $\mathit{isWSCore}()$, P can only be either a task or an intermediate event.

- $P = \mathsf{task}(\mathsf{e},\mathsf{e}')$. The only rule it can be applied is *P-Task*. 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 it follows that $\sigma(\mathsf{e}') = 0$. The application of the rule produces $\langle \mathsf{task}(\mathsf{e},\mathsf{e}'),\sigma\rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = \mathit{inc}(\mathit{dec}(\sigma,\mathsf{e}),\mathsf{e}')$. Thus, it follows $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$, which permits to conclude.

- $P = \mathsf{taskRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$. The only rule that can be applied is *P-TaskRcv*. In order to apply it 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 it follows $\sigma(\mathsf{e}') = 0$. 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, it follows $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$, which permits to conclude.

- $P = \mathsf{taskSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$. The only rule that can be applied is *P-TaskSnd*. 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 it follows $\sigma(\mathsf{e}') = 0$. The application of the rule produces $\langle \mathsf{taskSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}'), \sigma \rangle \xrightarrow{!\mathsf{m}} \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$. Thus, it follows $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$, which permits to conclude.

- $P = \mathsf{interRcv}(\mathsf{e}, \mathsf{e}', \mathsf{m})$. The only rule that can be applied is *P-InterRcv*. In order to apply it 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 it follows $\sigma(\mathsf{e}') = 0$. The application of the rule produces $\langle \mathsf{interRcv}(\mathsf{e}, \mathsf{e}', \mathsf{m}), \sigma \rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$. Thus, it follows $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$, which permits to conclude.

- $P = \mathsf{interSnd}(\mathsf{e}, \mathsf{e}', \mathsf{m})$,. The only rule that can be applied is *P-InterSnd*. In order to apply it 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 it follows $\sigma(\mathsf{e}') = 0$. The application of the rule produces $\langle \mathsf{interSnd}(\mathsf{e}, \mathsf{e}', \mathsf{m}), \sigma \rangle \xrightarrow{!\mathsf{m}} \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$. Thus, it follows $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$, which permits to conclude.

- $P = \mathsf{empty}(\mathsf{e}, \mathsf{e}')$,. The only rule that can be applied is *P-Empty*. 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 it follows $\sigma(\mathsf{e}') = 0$. The application of the rule produces $\langle \mathsf{empty}(\mathsf{e}, \mathsf{e}'), \sigma \rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$. Thus, it follows $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$, which permits to conclude.

Inductive cases:

- $P = \langle \mathsf{andSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{andJoin}(E', \mathsf{e}''), \sigma \rangle$. There are the following possibilities:

  - $\langle \mathsf{andSplit}(\mathsf{e}, E), \sigma \rangle$ evolves by means of rule *P-AndSplit*. It can be exploited 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 $\sigma'_1 = inc(dec(\sigma, \mathsf{e}), E)$. Now, P can evolve only through the application of *P-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 $isCompleteEl(P_1 \parallel \ldots \parallel P_n, \sigma'_3)$. Now,

P can only evolve by applying rule *P-Int1*, 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-AndJoin*. The application of the rule produces $\langle \mathsf{andJoin}(E', \mathsf{e}''), \sigma_4' \rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma_4', E'), \mathsf{e}'')$, i.e. $\sigma'(\mathsf{e}'') = 1$ and $\forall \mathsf{e}''' \in E' \,.\, \sigma'(\mathsf{e}''') = 0$. This permits to conclude.

- $P_1 \parallel \ldots \parallel P_n$ evolves without affecting the split and join gateways. In this case it easily follows by inductive hypothesis.
- $P_1 \parallel \ldots \parallel P_n$ evolves and affects the split and/or join gateways. In this case one can reason like in the first case.

- $P = \langle \mathsf{xorSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E', \mathsf{e}''), \sigma \rangle$. There are the following possibilities:
  - $\langle \mathsf{xorSplit}(\mathsf{e}, E), \sigma \rangle$ evolves by means of rule *P-XorSplit*. It can be exploited 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-Int1* 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 $isCompleteEl(P_1 \parallel \ldots \parallel P_n, \sigma_3')$. Now, P can only evolve by applying rule *P-Int1*, 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-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 to conclude.
  - $P_1 \parallel \ldots \parallel P_n$ evolves without affecting the split and join gateways. In this case if easily follows by inductive hypothesis.
  - $P_1 \parallel \ldots \parallel P_n$ evolves and affects the split and/or join gateways. In this case one can reason like in the first case.

- $P = \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j, \mathsf{e}_j') | j \in [1..n]\}) \parallel P_1 \parallel \ldots \mid P_n \parallel \mathsf{xorJoin}(E, \mathsf{e}'')$. 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-EventG*. It can be exploited 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-Int1* 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 $isCompleteEl(P_1 \parallel \ldots \parallel P_n, \sigma_3')$. Now, P can only evolve by applying rule *P-Int1*, 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-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 to conclude.

  - $P_1 \parallel \ldots \parallel P_n$ evolves without affecting the split and join gateways. In this case it easily follows by inductive hypothesis.
  - $P_1 \parallel \ldots \parallel P_n$ evolves and affects the split and/or join gateways. In this case one can reason like in the first case.

- $\mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{\mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\})$ 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*. It can be exploited 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 $\sigma(\mathsf{e}''') = 1$; assuming that the marking is $\sigma(\mathsf{e}''') = 1$ (since the other case is similar), then $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}'), \sigma \rangle \xrightarrow{\epsilon} \sigma_1'$ 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 to conclude.
  - $P_1 \parallel P_2$ evolves without affecting the split and join gateways. In this case it easily follows by inductive hypothesis.
  - $P_1 \parallel P_2$ evolves and affects the split and/or join gateways. In this case onecan reason like in the first case.

- Be $\langle P, \sigma \rangle = \langle P_1 \parallel P_2, \sigma \rangle$, 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*. It follows that $\langle P_1 \parallel P_2, \sigma \rangle \xrightarrow{\ell} \sigma_1'$ with $\langle P_1, \sigma \rangle \xrightarrow{\ell} \sigma_1'$. By inductive hypothesis it follows that there exists σ' such that $isCompleteEl(P_1, \sigma')$. By hypothesis $out(P_1) = in(P_2)$ thus, $isCompleteEl(\mathsf{getOutEl}(\mathsf{e}, P_1 \parallel P_2), \sigma')$ $= isCompleteEl(\mathsf{getOutEl}(\mathsf{e}, P_1), \sigma')$, that holds by inductive hypothesis. By hypothesis P_2 is well structured and core reachable, then it follows that $edges(P_2) \backslash out(P_2)) : \sigma'(\mathsf{e}) = 0$ By definition of $isCompleteEl(P_1, \parallel P_2, \sigma')$ one can conclude.

□

Theorem 3. *Let isWS(P), then P is sound.*

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

Assuming that $isInit(P, \sigma)$, then $\sigma(start(P)) = 1$, 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 it can be applied 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 it follows that $\langle \mathsf{start}(\mathsf{e}, \mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'', \mathsf{e}'''), \sigma'_3 \rangle$, where $\sigma'_3(\mathsf{e}'') = 1$ and $\forall \mathsf{e}^{\mathsf{v}} \in edges(P') \; . \; \sigma'_3(\mathsf{e}^{\mathsf{v}}) = 0$. Now it can be applied the rule *P-End* that decrements the token in e'' and produces a token in e''', which permits to conclude. □

Theorem 4. *Let C be a collaboration, $isWS(C)$ does not imply C is sound.*

Proof. Let C be a WS collaboration. Supposing that C is sound, then, it is sufficient to show a counter example, i.e. a WS collaboration that is not sound. It can be considered, for instance, the collaboration in Figure C.1. 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. □

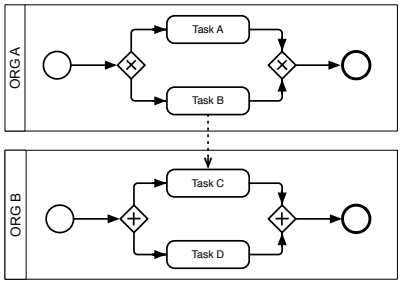


Figure C.1: Example of Unsound Collaboration with Sound WS Processes.

Theorem 5. *Let C be a collaboration, $isWS(C)$ does not imply C is message-relaxed sound.*

Proof. Let C be a WS collaboration. Supposing 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. One can consider again the collaboration in Figure C.1. 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 to conclude. □

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

Proof. Let P be a unsafe collaboration. Supposing that P is unsound, then, it is sufficient to show a counter example, i.e. a unsafe collaboration that is sound. One can consider the process in Figure C.2. It is unsafe since the AND-Split gateway creates two tokens that are then merged by the XOR-Join gateway producing two tokens on the outgoing edge of the XOR-Join. However, after *Task C* is executed and one token enables the terminate end event, the *kill* label is produced and the second token in the sequence flow is removed (rule *P-Terminate*), rendering the process sound. □

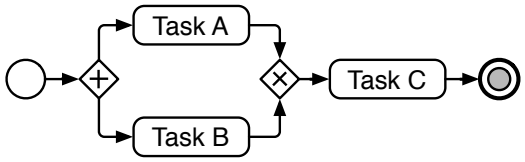


Figure C.2: Example of Unsafe but Sound Process.

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

Proof. Let C be a unsafe collaboration. Supposing that C is unsound, then, it is sufficient to show a counter example, i.e. a unsafe collaboration that is sound. One can consider the collaboration in Figure C.3. 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. □

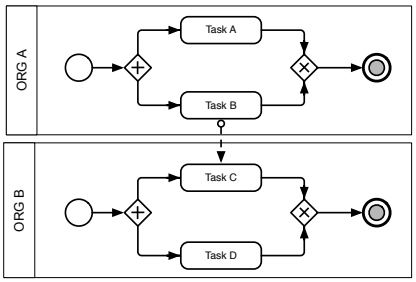


Figure C.3: Example of Unsafe but Sound Collaboration.

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 $P \in participant(C)$ 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, it follows that $\langle P, \sigma' \rangle$ is cs-safe, which is a contradiction. □

Theorem 9. *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 $\langle C, \sigma, \delta \rangle \rightarrow^* \langle \sigma', \delta' \rangle$ it follows that there exist σ'' and δ'' such that $\langle C, \sigma', \delta' \rangle \rightarrow^* \langle \sigma'', \delta'' \rangle$, and $\forall\ P \in participant(C)$ it follows that $\langle P, \sigma'' \rangle$ is cs-sound and $\forall\, \mathsf{m} \in \mathbb{M}\ .\ \delta''(\mathsf{m}) = 0$. Since P_1 is unsound, it follows that, given σ'_1, such that $isInit(P_1, \sigma'_1)$, for all σ'_2 such that $\langle P_1, \sigma \rangle \rightarrow^* \sigma'_2$ it follows that does not exist σ'_3 such that $\langle P_1, \sigma'_2 \rangle \rightarrow^* \sigma'_3$, and $\langle P_1, \sigma'_3 \rangle$ is cs-sound. Choosing $\langle C, \sigma', \delta' \rangle$ such that $P_1 \in participant(C)$, by unsoundness of P_1 it follows that there exists a process in C that is not cs-sound, which is a contradiction. □

C.2 OR-Join

This appendix reports the proofs of the results presented in Chapter 5.

Lemma 5. *Given a marked process M, if $M \equiv P \cdot \sigma \xrightarrow{\ell}_L M' \equiv P \cdot \sigma'$ then, given $M'' \equiv P' \cdot \sigma$ for some P, we have that $M'' \star \ell = P' \cdot \sigma'$.*

Proof. It follows by structural induction of P. Since the Lemma deals with marked process, the safeness assumption is made. Base cases (only the most interesting cases are shown, since the other are similar):

- *Start*: $M = \mathsf{start(e.nc.w)} \equiv \mathsf{start(e)} \cdot \sigma$, $\ell = \mathsf{l} : \mathsf{e}$. $M' = \mathsf{start(e.nc.l)} = \mathsf{start(e)} \cdot \sigma'$ where $\sigma(\mathsf{e}) = 0$, $\sigma'(\mathsf{e}) = 1$. Given $M'' \equiv P' \cdot \sigma$, $M'' \star \mathsf{l} : \mathsf{e} = P' \cdot \sigma''$, since $\mathsf{l} : \mathsf{e}$ affects only the value of e, σ'' is identically to σ except for the value of e, in particular $\sigma''(\mathsf{e}) = 1$. Thus, $\sigma'' = \sigma'$.

- *End*: $M = \mathsf{end(e.nc.l, e'.nc.w)} \equiv \mathsf{end(e, e')} \cdot \sigma$, $\ell = (\mathsf{d} : \mathsf{e}, \mathsf{l} : \mathsf{e}')$. $M' = \mathsf{end(e.nc.d, e'.nc.l)} \equiv \mathsf{end(e, e')} \cdot \sigma'$, where $\sigma(\mathsf{e}) = 1$, $\sigma(\mathsf{e}') = 0$, $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$. Given $M'' = P' \cdot \sigma$, $M'' \star (\mathsf{d} : \mathsf{e}, \mathsf{l} : \mathsf{e}') \equiv P' \cdot \sigma''$. Since σ'' is identically to σ except for the value of edge e and e', in particular $\sigma''(\mathsf{e}) = 0$ and $\sigma''(\mathsf{e}') = 1$, thus $\sigma'' = \sigma'$.

- *OrSplit*: only one case is considered, since the other are similar, that is $M = \mathsf{orSplit(e.nc.}\Sigma, E_1 \sqcup E_2) \equiv \mathsf{orSplit(e}, E_1 \sqcup E_2) \cdot \sigma$, with $\ell = (\mathsf{d} : \{\mathsf{e}\} \sqcup E_2, \mathsf{l} : E_1)$. $M' = \mathsf{orSplit(e.nc.d}, setLive(E_1) \sqcup setDead(E_2)) = \mathsf{orSplit(e}, E_1 \sqcup E_2) \cdot \sigma'$ where $\sigma'(\mathsf{e}) = 0$, $\forall \mathsf{e}' \in E_2 . \sigma'(\mathsf{e}') = 0$ and $\forall \mathsf{e}' \in E_1 . \sigma'(\mathsf{e}') = 1$. Given $M'' \equiv P' \cdot \sigma$, $M'' \star (\mathsf{d} : \{\mathsf{e}\} \sqcup E_2, \mathsf{l} : E_1) \equiv P' \cdot \sigma''$, it follows that σ'' is identically equivalent to σ except for the value of e and $\mathsf{e}'\, \forall \mathsf{e}' \in E_1 \sqcup E_2$. However, by syntax correspondence $\sigma''(\mathsf{e}) = 0$, $\forall \mathsf{e}' \in E_2 . \sigma''(\mathsf{e}) = 0$ and $\forall \mathsf{e}' \in E_1 . \sigma''(\mathsf{e}) = 1$. Thus, $\sigma'' = \sigma'$.

- *OrJoin*: only one case is considered, that is $M = \mathsf{orJoin}(E_1 \sqcup E_2, \mathsf{e.nc}.\Sigma) = \mathsf{orJoin}(E_1 \sqcup E_2, \mathsf{e}) \cdot \sigma$, $\ell = (\mathsf{d} : E_1 \sqcup E_2, \mathsf{l} : \mathsf{e})$. $M' = \mathsf{orJoin}(setDead(E_1 \sqcup E_2), \mathsf{e.nc.l}) = \mathsf{orJoin}(E_1 \sqcup E_2, \mathsf{e}) \cdot \sigma'$ where $\sigma'(\mathsf{e}) = 1$, $\forall \mathsf{e}' \in E_1 \sqcup E_2 . \sigma(\mathsf{e}') = 0$. Given $M'' \equiv P' \cdot \sigma$, $M'' \star (\mathsf{d} : E_1 \sqcup E_2, \mathsf{l} : \mathsf{e}) \equiv P' \cdot \sigma''$. σ'' is identically equivalent to σ except for the value of e and $\mathsf{e}' \forall \mathsf{e}' \in E_1 \sqcup E_2$. However, by syntax correspondence $\forall \mathsf{e}' \in E_1 \sqcup E_2 . \sigma''(\mathsf{e}) = 0$ and $\sigma''(\mathsf{e}) = 1$. Thus, $\sigma'' = \sigma'$.

Inductive case:

- Given $M = M_1 \parallel M_2 \equiv P_1 \parallel P_2 \cdot \sigma = P \cdot \sigma \parallel P_2 \cdot \sigma$, $M' = M_1' \parallel M_2' \equiv P_1 \parallel P_2 \cdot \sigma' = P_1 \cdot \sigma' \parallel P_2 \cdot \sigma'$. Given $M'' = P' \cdot \sigma = M_1'' \parallel M_2'' \equiv P_1' \parallel P_2' \cdot \sigma$. $M'' \star \ell = M_1 \star \ell \parallel M_2 \star \ell \equiv P_1' \cdot \sigma'' \parallel P_2' \cdot \sigma'' = P_1' \parallel P_2' \cdot \sigma''$. By inductive hypothesis $P_1' \cdot \sigma''$ and $P_2' \cdot \sigma''$ are such that $\sigma'' = \sigma'$.

□

Theorem 10. *Let $\langle P, \sigma \rangle$ be a reachable process configuration, if $\langle P, \sigma \rangle \rightarrow_G \sigma'$ then for all reachable marked process M such that $M \equiv P \cdot \sigma$, $M \xrightarrow{\ell}_{L^+} M'$ and $M' \equiv P \cdot \sigma'$.*

Proof. By induction on the derivation of $\langle P, \sigma \rangle \rightarrow_G \sigma'$. Base cases:

- (*G-Start*): By hypothesis $\langle P, \sigma \rangle$ is a reachable process configuration, thus it follows that $\langle P, \sigma \rangle = \langle \mathsf{start}(\mathsf{e}), \sigma_0 \rangle$, $\sigma' = inc(\sigma_0, \mathsf{e})$. By definition of σ_0, it results $\sigma_0(\mathsf{e}) = 0$ and, by definition of *inc* function, $\sigma'(\mathsf{e}) = 1$. By definition of syntax correspondence $P \cdot \sigma = \mathsf{start}(\mathsf{e}) \cdot \sigma_0 = \mathsf{start}(\mathsf{e.t.nl})$ and $P \cdot \sigma' = \mathsf{start}(\mathsf{e.t.l})$. According to the process structure, it follows that $\mathsf{t} = \mathsf{nc}$. For any $M \equiv \mathsf{start}(\mathsf{e.nc.nl})$, by definition of initial status of a marked process, the only possible value of nl is w. Thus, it follows that there is only one M such that $M \equiv P \cdot \sigma$, that is $M = \mathsf{start}(\mathsf{e.nc.w})$. By applying rule *L-Start-NC*, $M \xrightarrow{\mathsf{l:e}}_L \mathsf{start}(\mathsf{e.nc.l})$. Thus, it results $\mathsf{start}(\mathsf{e.nc.l}) = M'$. This permits to conclude, as $M' \equiv P \cdot \sigma'$.

- (*G-End*): By hypothesis $\langle P, \sigma \rangle$ is a reachable process configuration, thus it follows that $\langle P, \sigma \rangle = \langle \mathsf{end}(\mathsf{e}, \mathsf{e}'), \sigma \rangle$, $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$, with $\sigma(\mathsf{e}) > 0$ (that under the safeness assumption means $\sigma(\mathsf{e}) = 1$) . By definition of functions *dec* and *inc* $\sigma'(\mathsf{e}) = 0$, $\sigma'(\mathsf{e}') = 1$. By definition of syntax correspondence $P \cdot \sigma = \mathsf{end}(\mathsf{e}, \mathsf{e}') \cdot \sigma = \mathsf{end}(\mathsf{e.t.l}, \mathsf{e}'.\mathsf{t.w})$ and $P \cdot \sigma' = \mathsf{end}(\mathsf{e.t.nl}, \mathsf{e}'.\mathsf{t.l})$. According to the process structure, it can only be $\mathsf{t} = \mathsf{nc}$. Thus, by the syntax correspondence, it follows that there exist only one M such that $M \equiv P \cdot \sigma$, that is $M = \mathsf{end}(\mathsf{e.nc.l}, \mathsf{e}'.\mathsf{nc.w})$. By applying rule *L-End-NC*, $M \xrightarrow{(\mathsf{d:e,l:e}')}_L \mathsf{end}(\mathsf{e.nc.d}, \mathsf{e}'.\mathsf{nc.l})$. So it follows that $\mathsf{end}(\mathsf{e.nc.d}, \mathsf{e}'.\mathsf{nc.l}) = M'$. It can be concluded as $M' \equiv P \cdot \sigma'$ by instantiating in $\mathsf{end}(\mathsf{e.t.nl}, \mathsf{e}'.\mathsf{nc.l})$ nl with d.

- (*G-AndSplit*): By hypothesis $\langle P, \sigma \rangle$ is a reachable process configuration, thus it follows that $\langle P, \sigma \rangle = \langle \mathsf{andSplit}(\mathsf{e}, E), \sigma \rangle, \sigma' = inc(dec(\sigma, \mathsf{e}), E)$, with

$\sigma(\mathsf{e}) > 0$ (that by safeness assumption means $\sigma(\mathsf{e}) = 1$). By definition of functions *inc* and *dec* it follows that $\sigma'(\mathsf{e}) = 0$ and $\forall \mathsf{e}' \in E.\sigma'(\mathsf{e}') = 1$. By definition of syntax correspondence $P \cdot \sigma = \mathsf{andSplit}(\mathsf{e}, E) \cdot \sigma = \mathsf{andSplit}(\mathsf{e.t.l}, (E \cdot \sigma))$ and $P \cdot \sigma' = \mathsf{andSplit}(\mathsf{e.t.nl}, (E \cdot \sigma'))$, where $isLive(E \cdot \sigma')$. Since M is a reachable marked process it follows $isWait(E \cdot \sigma)$. Indeed, in order to have an edge in $E \cdot \sigma$ annotated by d, one would have applied rule (*D-AndSplit*), which however cannot be applied because it requires edge e to be annotated by d, that is a contradiction as it is annotated by l. Thus, it follows $M \equiv P \cdot \sigma$ and we have two cases, according to instantiation of t: $M_1 = \mathsf{andSplit}(\mathsf{e.nc.l}, E)$ and $M_2 = \mathsf{andSplit}(\mathsf{e.c.l}, E)$. It follows by case analysis:

- Given M_1, by applying rule *L-AndSplit-NC*, $M_1 \xrightarrow{(\mathsf{d}:\mathsf{e},\mathsf{l}:E)}_L \mathsf{andSplit}(\mathsf{e.nc.d}, setLive(E))$. Thus, it follows $\mathsf{andSplit}(\mathsf{e.nc.d}, setLive(E)) = M_1'$. One can conclude as $M_1' \equiv P \cdot \sigma'$ up to the instantiation of nl occurrences in $\mathsf{andSplit}(\mathsf{e.t.nl}, (E \cdot \sigma'))$ with d.
- Given M_2, by applying rule *L-AndSplit-C*, $M_2 \xrightarrow{(\mathsf{w}:\mathsf{e},\mathsf{l}:E)}_L \mathsf{andSplit}(\mathsf{e.c.w}, setLive(E))$. Thus, it follows $\mathsf{andSplit}(\mathsf{e.c.w}, setLive(E)) = M_2'$. It can be concluded as $M_1' \equiv P \cdot \sigma'$ by instantiating nl occurrences in $\mathsf{andSplit}(\mathsf{e.t.nl}, (E \cdot \sigma'))$ with w.

- (*G-AndJoin*): By hypothesis $\langle P, \sigma \rangle$ is a reachable process configuration, thus it follows that $\langle P, \sigma \rangle = \langle \mathsf{andJoin}(E, \mathsf{e}), \sigma \rangle$, $\sigma' = inc(dec(\sigma, E), \mathsf{e})$, with $\forall \mathsf{e}' \in E.\sigma(\mathsf{e}') > 0$, that by safeness assumption means $\sigma(\mathsf{e}') = 1$. By definition of functions *inc* and *dec*, it follows $\sigma'(\mathsf{e}) = 1$ and $\forall \mathsf{e}' \in E.\sigma'(\mathsf{e}') = 0$. By definition of syntax correspondence $P \cdot \sigma = \mathsf{andJoin}(E, \mathsf{e}) \cdot \sigma = \mathsf{andJoin}((E \cdot \sigma), \mathsf{e.t.nl})$ where $E \cdot \sigma$ is such that $isLive(E \cdot \sigma)$ and $P \cdot \sigma' = \mathsf{andJoin}((E \cdot \sigma'), \mathsf{e.t.l})$, where $E \cdot \sigma'$ is such that either $isWait(E \cdot \sigma')$ or $isDead(E \cdot \sigma')$. Indeed, in order to have an edge e in $P \cdot \sigma$ annotated by d, one would have applied rule (*D-AndJoin*), which however cannot be applied because it requires $isDead(E \cdot \sigma)$, that is a contradiction as $isLive(E \cdot \sigma)$. Thus, there are only two cases to consider, according to the process structure, $M_1 = \mathsf{andJoin}(E, \mathsf{e.nc.}\Sigma)$ and $M_2 = \mathsf{andJoin}(E_1 \sqcup E_2, \mathsf{e.c.}\Sigma)$, where $E = E_1 \sqcup E_2$. By case analysis:

- Given M_1, by applying rule (*L-AndJoin-NC*), $M_1 \xrightarrow{(\mathsf{d}:E,\mathsf{l}:\mathsf{e})}_L \mathsf{andJoin}(setDead(E), \mathsf{e.nc.l})$. It follows $\mathsf{andJoin}(setDead(E), \mathsf{e.nc.l}) = M_1'$. One can conclude as $M_1' \equiv P \cdot \sigma'$ by instantiating nl occurrences in $\mathsf{andJoin}((E \cdot \sigma'), \mathsf{e.t.l})$ with d.
- Given M_2, by applying rule (*L-AndJoin-C*), $M_2 \xrightarrow{(\mathsf{w}:E_1,\mathsf{d}:E_2,\mathsf{l}:\mathsf{e})}_L \mathsf{andJoin}(setWait(E_1) \sqcup setDead(E_2), \mathsf{e.c.l})$. Thus, It follows $\mathsf{andJoin}(setWait(E_1) \sqcup setDead(E_2), \mathsf{e.c.l}) = M_2'$. One can conclude as $M_2' \equiv P \cdot \sigma'$ by instantiating nl occurrences in $\mathsf{andJoin}((E \cdot \sigma'), \mathsf{e.t.l})$ such that $isWait(E_1 \cdot \sigma')$, $isDead(E_2 \cdot \sigma')$.

- (*G-XorSplit*): By hypothesis $\langle P, \sigma\rangle$ is a reachable process configuration, thus it follows that $\langle P, \sigma\rangle = \langle \mathsf{xorSplit}(\mathsf{e}, \{\mathsf{e}'\} \cup E), \sigma\rangle$, $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$, with $\sigma(\mathsf{e}) = 1$ and $\forall \mathsf{e}'' \neq \mathsf{e}' \in E.\sigma'(\mathsf{e}'') = 0$ (by definition of reachable process and safeness assumption). By definition of functions *inc* and *dec*, $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$. By definition of syntax correspondence $P \cdot \sigma = \mathsf{xorSplit}(\mathsf{e}, \{\mathsf{e}'\} \cup E) \cdot \sigma = \mathsf{xorSplit}(\mathsf{e.t.l}, \{\mathsf{e}'.\mathsf{t.nl}\} \sqcup (E \cdot \sigma))$, where $E \cdot \sigma$ is such that either $isWait(E \cdot \sigma)$ or $isDead(E \cdot \sigma)$. Indeed, in order to have an edge in $E \cdot \sigma$ annotated by d, one would have applied rule (*D-XorSplit*), which however cannot be applied because it requires edge e to be annotated by d, that is a contradiction as it is annotated by l. Moreover, $P \cdot \sigma' = \mathsf{xorSplit}(\mathsf{e}, \{\mathsf{e}'\} \cup E) \cdot \sigma' = \mathsf{xorSplit}(\mathsf{e.t.nl}, \{\mathsf{e}'.\mathsf{t.l}\} \sqcup (E \cdot \sigma'))$, where $E \cdot \sigma'$ is such that either $isWait(E \cdot \sigma')$ or $isDead(E \cdot \sigma')$. Thus, it follows $M \equiv P \cdot \sigma$ and there three cases, according to the instantiation of t based on the process structure: $M_1 = \mathsf{xorSplit}(\mathsf{e.nc.l}, \{\mathsf{e}'.\mathsf{nc}.\Sigma'\} \sqcup E)$, $M_2 = \mathsf{xorSplit}(\mathsf{e.c.l}, \{\mathsf{e}'.\mathsf{c}.\Sigma'\} \sqcup E_1 \sqcup E_2)$, $M_3 = \mathsf{xorSplit}(\mathsf{e.c.l}, \{\mathsf{e}'.\mathsf{nc}.\Sigma'\} \sqcup E_1 \sqcup E_2)$.

  By case analysis:

  - Given M_1, by applying rule (*L-XorSplit-NC*) it follows $M_1 \xrightarrow{(\mathsf{w:e,l:e}')}_L \mathsf{xorSplit}(\mathsf{e.nc.d}, \{\mathsf{e}'.\mathsf{nc.l}\} \sqcup setDead(E))$. Thus, $\mathsf{xorSplit}(\mathsf{e.nc.d}, \{\mathsf{e}'.\mathsf{nc.l}\} \sqcup setDead(E)) = M_1'$. One can conclude as $M_1' \equiv P \cdot \sigma'$ by instantiating nl occurrences in $\mathsf{xorSplit}(\mathsf{e.t.nl}, \{\mathsf{e}'.\mathsf{t.l}\} \sqcup (E \cdot \sigma'))$ with d.
  - Given M_2, by applying rule (L_1-*XorSplit-C*) then $M_2 \xrightarrow{(\mathsf{w:e,l:e}')}_L \mathsf{xorSplit}(\mathsf{e.c.w}, \{\mathsf{e}'.\mathsf{c.l}\} \sqcup E_1 \sqcup E_2)$. Thus, it follows $\mathsf{xorSplit}(\mathsf{e.c.w}, \{\mathsf{e}'.\mathsf{c.l}\} \sqcup E_1 \sqcup E_2) = M_2'$. One can conclude as $M_2' \equiv P \cdot \sigma'$ by instantiating nl occurrences in $\mathsf{xorSplit}(\mathsf{e.t.nl}, \{\mathsf{e}'.\mathsf{t.l}\} \sqcup (E \cdot \sigma'))$ with w.
  - Given M_3, by applying rule (L_2-*XorSplit-C*) it results $M_3 \xrightarrow{(\mathsf{d}:\{\mathsf{e}\} \sqcup E_1 \sqcup E_2, \mathsf{l}:\mathsf{e}')}_L \mathsf{xorSplit}(\mathsf{e.c.d}, \{\mathsf{e}'.\mathsf{nc.l}\} \sqcup setDead(E_1) \sqcup setDead(E_2))$. Thus, $\mathsf{xorSplit}(\mathsf{e.c.d}, \{\mathsf{e}'.\mathsf{nc.l}\} \sqcup setDead(E_1) \sqcup setDead(E_2)) = M_3'$. One can conclude as $M_3' \equiv P \cdot \sigma'$ by instantiating nl occurrences in $\mathsf{xorSplit}(\mathsf{e.t.nl}, \{\mathsf{e}'.\mathsf{t.l}\} \sqcup (E \cdot \sigma'))$ with d

- (*G-XorJoin*): By hypothesis $\langle P, \sigma\rangle$ is a reachable process configuration, thus it follows that $\langle P, \sigma\rangle = \langle \mathsf{xorJoin}(\{\mathsf{e}'\} \sqcup E, \mathsf{e}), \sigma\rangle$, $\sigma' = inc(dec(\sigma, \mathsf{e}'), \mathsf{e})$, with $\sigma(\mathsf{e}') = 1$ (under safeness assumption). By definition of functions *inc* and *dec*, $\sigma'(\mathsf{e}) = 1$ and $\sigma'(\mathsf{e}') = 0$. By definition of syntax correspondence $P \cdot \sigma = \mathsf{xorJoin}(\{\mathsf{e}'\} \cup E, \mathsf{e}) \cdot \sigma = \mathsf{xorJoin}(\{\mathsf{e}'.\mathsf{t.l}\} \sqcup (E \cdot \sigma), \mathsf{e.t.nl})$, where $E \cdot \sigma$ is such that either $isWait(E \cdot \sigma)$ or $isDead(E \cdot \sigma)$. Indeed, $\langle P, \sigma\rangle$ is a safe reachable process configuration, therefore it should be $\forall \mathsf{e}'' \neq \mathsf{e}' \in E.\sigma'(\mathsf{e}'') = 0$. $P \cdot \sigma' = \mathsf{xorJoin}(\{\mathsf{e}'\} \cup E, \mathsf{e}) \cdot \sigma' = \mathsf{xorJoin}(\{\mathsf{e}'.\mathsf{t.nl}\} \sqcup (E \cdot \sigma'), \mathsf{e.t.l})$ where $E \cdot \sigma'$ is such that either $isWait(E \cdot \sigma')$ or $isDead(E \cdot \sigma')$. Thus, it follows $M \equiv P \cdot \sigma$ such as $M = \mathsf{xorJoin}(\{\mathsf{e}'.\mathsf{t.l}\} \sqcup E, \mathsf{e.t.}\Sigma)$ and there are three cases according to the type

t given by the preprocessing step: $M_1 = \mathsf{xorJoin}(\{\mathsf{e'.nc.l}\} \sqcup E, \mathsf{e.nc.}\Sigma)$, $M_2 = \mathsf{xorJoin}(\{\mathsf{e'.c.l}\} \sqcup E_1 \sqcup E_2, \mathsf{e.c.}\Sigma)$, $M_3 = \mathsf{xorJoin}(\{\mathsf{e'.nc.l}\} \sqcup E_1 \sqcup E_2, \mathsf{e.c.}\Sigma)$.

By case analysis:

- Given M_1, by applying rule (*L-XorJoin-NC*) it follows $M_1 \xrightarrow{(\mathsf{d}:\{\mathsf{e'}\}\sqcup E, \mathsf{l}:\mathsf{e})}_L \mathsf{xorJoin}(\{\mathsf{e'.nc.d}\} \sqcup \mathit{setDead}(E), \mathsf{e.nc.l})$. Thus, $\mathsf{xorJoin}(\{\mathsf{e'.nc.d}\} \sqcup \mathit{setDead}(E), \mathsf{e.nc.l}) = M_1'$. One can conclude as $M_1' \equiv P \cdot \sigma'$ by instantiating nl occurrences in $\mathsf{xorJoin}(\{\mathsf{e'.t.nl}\} \sqcup (E \cdot \sigma'), \mathsf{e.t.l})$ with d.
- Given M_2, by applying rule (L_1-*XorJoin-C*) it follows $M_2 \xrightarrow{(\mathsf{w}:\mathsf{e'},\mathsf{d}:E_2,\mathsf{l}:\mathsf{e})}_L \mathsf{xorJoin}(\{\mathsf{e'.c.w}\} \sqcup E_1 \sqcup \mathit{setDead}(E_2), \mathsf{e.c.l})$. Thus, $\mathsf{xorJoin}(\{\mathsf{e'.c.w}\} \sqcup E_1 \sqcup \mathit{setDead}(E_2), \mathsf{e.c.l}) = M_2'$. One can conclude as $M_2' \equiv P \cdot \sigma'$ up to the instantiation of nl occurrences in $\mathsf{xorJoin}(\{\mathsf{e'.t.nl}\} \sqcup (E \cdot \sigma'), \mathsf{e.t.l})$ such that $\mathsf{e'.t.w}$, $\mathit{isWait}(E_1 \cdot \sigma')$ and $\mathit{isDead}(E_2 \cdot \sigma')$.
- Given M_3, by applying rule (L_2-*XorJoin-C*) it follows $M_3 \xrightarrow{(\mathsf{d}:\{\mathsf{e'}\}\sqcup E_2,\mathsf{l}:\mathsf{e})}_L \mathsf{xorJoin}(\{\mathsf{e'.nc.d}\} \sqcup E_1 \sqcup \mathit{setDead}(E_2), \mathsf{e.c.l})$. Thus, $\mathsf{xorJoin}(\{\mathsf{e'.nc.d}\} \sqcup E_1 \sqcup \mathit{setDead}(E_2), \mathsf{e.c.l}) = M_3'$. One can conclude as $M_2' \equiv P \cdot \sigma'$ up to instantiation of nl occurrences in $\mathsf{xorJoin}(\{\mathsf{e'.t.nl}\} \sqcup (E \cdot \sigma'), \mathsf{e.t.l})$ such that $\mathsf{e'}$ is annotated by d, $\mathit{isWait}(E_1 \cdot \sigma')$, $\mathit{isDead}(E_2 \cdot \sigma')$.

- (*G-OrSplit*): By hypothesis $\langle P, \sigma\rangle$ is a reachable process configuration, thus it follows $\langle P, \sigma\rangle = \langle \mathsf{orSplit}(\mathsf{e}, E_1 \sqcup E_2), \sigma\rangle$, $\sigma' = \mathit{inc}(\mathit{dec}(\sigma, \mathsf{e}), E_1)$, with $\sigma(\mathsf{e}) = 1$ (by safeness assumption) and $E_1 \neq \varnothing$. Moreover, $\forall \mathsf{e'} \in E.\sigma(\mathsf{e'}) = 0$. Indeed, if a token marks an edge in E_1 after the transition more than one token would mark this edge violating the safeness assumption. Moreover, since $\langle P, \sigma\rangle$ is reachable also edges of E_2 are unmarked. By definition of functions *inc* and *dec* $\sigma'(\mathsf{e}) = 0$ and $\forall \mathsf{e'} \in E_1.\sigma'(\mathsf{e'}) = 1$. Since the transition does not affect E_2 it follows $\forall \mathsf{e''} \in E_2.\sigma'(\mathsf{e''}) = 0$. By definition of syntax correspondence $P \cdot \sigma = \mathsf{orSplit}(\mathsf{e}, E_1 \sqcup E_2) \cdot \sigma = \mathsf{orSplit}(\mathsf{e.t.l}, (E_1 \cdot \sigma) \sqcup (E_2 \cdot \sigma))$ where $E_1 \cdot \sigma$ (equivalently $E_2 \cdot \sigma$) is such that either $\mathit{isWait}(E_1 \cdot \sigma)$ or $\mathit{isDead}(E_1 \cdot \sigma)$ (equivalently $\mathit{isWait}(E_2 \cdot \sigma)$ or $\mathit{isDead}(E_2 \cdot \sigma)$). In particular, reasoning as in the previous cases one can prove that it can only be $\mathit{isWait}(E_1 \cdot \sigma)$ and $\mathit{isWait}(E_2 \cdot \sigma)$. $P \cdot \sigma' = \mathsf{orSplit}(\mathsf{e.t.nl}, (E_1 \cdot \sigma') \sqcup (E_2 \cdot \sigma'))$ where $E_1 \cdot \sigma'$ is such that $\mathit{isLive}(E_1 \cdot \sigma')$ while $E_2 \cdot \sigma'$ is such that either $\mathit{isWait}(E_2 \cdot \sigma')$ or $\mathit{isDead}(E_2 \cdot \sigma')$. Thus, for all reachable marked process M, $M \equiv P \cdot \sigma$ is such that $M = \mathsf{orSplit}(\mathsf{e.t.l}, E_1 \sqcup E_2)$ and there are two cases according to the instantiation of t: $M_1 = \mathsf{orSplit}(\mathsf{e.nc.l}, E_1 \sqcup E_2)$ and $M_2 = \mathsf{orSplit}(\mathsf{e.c.l}, E_1 \sqcup E_2)$.

  By case analysis:

  - Given M_1, by applying rule *L-OrSplit-NC*, $M_1 \xrightarrow{(\mathsf{d}:\{\mathsf{e}\}\sqcup E_2,\mathsf{l}:E_1)}_L \mathsf{orSplit}(\mathsf{e.nc.d}, \mathit{setLive}(E_1) \sqcup \mathit{setDead}(E_2))$. Thus, $\mathsf{orSplit}(\mathsf{e.nc.d}, \mathit{setLive}(E_1) \sqcup \mathit{setDead}(E_2)) = M_1'$. One can

conclude as $M_1' \equiv P \cdot \sigma'$ by instantiating nl occurrences in $\mathsf{orSplit}(\mathsf{e.t.nl}, (E_1 \cdot \sigma') \sqcup (E_2 \cdot \sigma'))$, with d.

– Given M_2 two rules can be applied L_1-*OrSplit-C* or L_2-*OrSplit-C*.

* L_1-*OrSplit-C*: it follows $M_2 \xrightarrow{(\mathsf{w:e,l:}E_1)}_L \mathsf{orSplit}(\mathsf{e.c.w}, \mathit{setLive}(E_1) \sqcup E_2)$. It follows $\mathsf{orSplit}(\mathsf{e.c.w}, \mathit{setLive}(E_1) \sqcup E_2) = M_2'$. One can conclude as $M_2' \equiv P \cdot \sigma'$ up to the instantiation of nl occurrences in $\mathsf{orSplit}(\mathsf{e.t.nl}, (E_1 \cdot \sigma') \sqcup (E_2 \cdot \sigma'))$ of with w.
* L_2-*OrSplit-C*: it follows $M_2 \xrightarrow{(\mathsf{d:}\{\mathsf{e}\} \sqcup E_2, \mathsf{l:}E_1)}_L \mathsf{orSplit}(\mathsf{e.c.d}, \mathit{setLive}(E_1) \sqcup \mathit{setDead}(E_2))$.
Thus, $\mathsf{orSplit}(\mathsf{e.c.d}, \mathit{setLive}(E_1) \sqcup \mathit{setDead}(E_2)) = M_2'$. One can conclude as $M_2' \equiv P \cdot \sigma'$ up to the instantiation of nl occurrences in $\mathsf{orSplit}(\mathsf{e.t.nl}, (E_1 \cdot \sigma') \sqcup (E_2 \cdot \sigma'))$ of with d.

- (G-*OrJoin*): By hypothesis $\langle P, \sigma \rangle$ is a reachable process configuration, thus it follows $\langle P, \sigma \rangle = \langle \mathsf{orJoin}(E_1 \sqcup E_2, \mathsf{e}), \sigma \rangle$, $\sigma' = \mathit{inc}(\mathit{dec}(\sigma, E_1), \mathsf{e})$, with $E_1 \neq \varnothing$, $\forall \mathsf{e}' \in E_1 . \sigma(\mathsf{e}') > 0$. By the safeness of the collaboration $\sigma(\mathsf{e}) = 0$ (otherwise edge e would be marked by more than one token after the transition, violating the safeness assumption); similarly $\forall \mathsf{e}' \in E_1 . \sigma(\mathsf{e}') = 1$. By definition of functions *inc* and *dec* $\sigma'(\mathsf{e}) = 1$ and $\forall \mathsf{e}' \in E_1 . \sigma'(\mathsf{e}') = 0$. Since the transition does not affect E_2 it follows $\forall \mathsf{e}'' \in E_2 . \sigma'(\mathsf{e}'') = 0$. By definition of syntax correspondence $P \cdot \sigma = \mathsf{orJoin}(E_1 \sqcup E_2, \mathsf{e}) \cdot \sigma = \mathsf{orJoin}((E_1 \cdot \sigma) \sqcup (E_2 \cdot \sigma), \mathsf{e.t.nl})$ where edge e is annotated by w and $E_1 \cdot \sigma$ is such that $\mathit{isLive}(E_1 \cdot \sigma)$ while $E_2 \cdot \sigma$ is such that either $\mathit{isWait}(E_2 \cdot \sigma)$ or $\mathit{isDead}(E_2 \cdot \sigma)$. Indeed in order to have edge e annotated by d, one would have applied rule (D-*OrJoin*), which however cannot be applied because it requires all edges of E to be annotated by d, that is a contradiction as $\mathit{isLive}(E_1 \cdot \sigma)$. Moreover, if $\mathit{isWait}(E_2 \cdot \sigma)$, by repetitively applying rules for dead status propagation it results that there exists M' such that $M \xrightarrow{\ell}_L^+ M'$, with $M' = \mathsf{orJoin}(E_1 \sqcup E_2, \mathsf{e.nc.}\Sigma)$ and $\mathit{isDead}(E_2)$. In fact, by hypothesis it follows $\forall p_1 \in \Pi . \exists\, p_2 \in \Pi_{p_1}$. By safeness assumption this condition holds only if $\nexists p_1 \in \Pi$. Otherwise there would be more than one token arriving in a E_1, thus after the transition more than one token would mark the same sequence flow, violating the safeness assumption. This means that $\nexists p \in \mathcal{P}(\mathsf{e}) \mid \mathit{isLive}(\mathsf{first}(p) \cdot \sigma) \wedge \mathsf{last}(p) \cdot \sigma \in E_2$. Thus, either $\mathit{isDead}(\mathsf{first}(p) \cdot \sigma)$ or $\mathit{isWait}(\mathsf{first}(p) \cdot \sigma)$. If $\mathit{isDead}(\mathsf{first}(p) \cdot \sigma)$, by applying rules for the propagation of the dead status it follows that $\mathit{isDead}(\mathsf{last}(p) \cdot \sigma) \in E_2$. If $\mathit{isWait}(\mathsf{first}(p) \cdot \sigma)$ a token may arrive on $\mathsf{first}(p) \cdot \sigma$ violating the condition for the application of (G-*OrJoin*), that is a contradiction. If $\mathit{isDead}(E_2 \cdot \sigma)$ It can be applied (L-*OrJoin-NC*) or (L_1-*OrJoin-C*) or (L_2-*OrJoin-C*). Moreover, $P \cdot \sigma' = \mathsf{orJoin}((E_1 \cdot \sigma) \sqcup (E_2 \cdot \sigma), \mathsf{e.t.l})$ where $E_1 \cdot \sigma'$ ($E_2 \cdot \sigma'$ resp.) is such that either $\mathit{isWait}(E_1 \cdot \sigma')$ or $\mathit{isDead}(E_1 \cdot \sigma')$ ($\mathit{isWait}(E_2 \cdot \sigma')$ or $\mathit{isDead}(E_2 \cdot \sigma')$ resp.). Thus, it follows that there exists only a reachable marked process M such that $M \equiv P \cdot \sigma$ and there are two options, according to the instantiation of t, $M_1 = \mathsf{orJoin}(E_1 \sqcup E_2, \mathsf{e.nc.}\Sigma)$ and $M_2 = \mathsf{orJoin}(E_1 \sqcup E_2, \mathsf{e.c.}\Sigma)$.

By case analysis:

- Given M_1 by applying rule (*L-OrJoin-NC*), $M_1 \xrightarrow{(\mathsf{d}:E_1 \sqcup E_2, \mathsf{l}:\mathsf{e})}_L \mathsf{orJoin}(setDead(E_1 \sqcup E_2), \mathsf{e.nc.l})$, I have $\mathsf{orJoin}(setDead(E_1 \sqcup E_2), \mathsf{e.nc.l}) = M_1'$. One can conclude as $M_1' \equiv P \cdot \sigma'$ by instantiating nl occurrences in $\mathsf{orJoin}((E_1 \cdot \sigma) \sqcup (E_2 \cdot \sigma), \mathsf{e.t.l})$ with d.
- Given M_2 two rules can be applied, (L_1-*OrJoin-C*) or (L_2-*OrJoin-C*).
  - (L_1-*OrJoin-C*): it follows $M_2 \xrightarrow{(\mathsf{w}:E_1, \mathsf{d}:E_2, \mathsf{l}:\mathsf{e})}_L \mathsf{orJoin}(setWait(E_1) \sqcup setDead(E_2), \mathsf{e.c.l})$. Thus, $\mathsf{orJoin}(setWait(E_1) \sqcup setDead(E_2), \mathsf{e.c.l}) = M_2'$. One can conclude as $M_2' \equiv P \cdot \sigma'$ up to the instantiation of nl occurrences in $\mathsf{orJoin}((E_1 \cdot \sigma) \sqcup (E_2 \cdot \sigma), \mathsf{e.t.l})$ such that $isWait(E_1 \cdot \sigma')$ and $isDead(E_2 \cdot \sigma')$.
  - L_2-*OrJoin-C*: it follows $M_2 \xrightarrow{(\mathsf{w}:E_2, \mathsf{d}:E_1, \mathsf{l}:\mathsf{e})}_L \mathsf{orJoin}(setDead(E_1) \sqcup setWait(E_2), \mathsf{e.c.l})$. Thus, $\mathsf{orJoin}(setDead(E_1) \sqcup setWait(E_2), \mathsf{e.c.l}) = M_2'$. One can conclude as $M_2' \equiv P \cdot \sigma'$ up to the instantiation of nl occurrences in $\mathsf{orJoin}((E_1 \cdot \sigma) \sqcup (E_2 \cdot \sigma), \mathsf{e.t.l})$ such that $isDead(E_1 \cdot \sigma')$ and $isWait(E_2 \cdot \sigma')$.

Inductive case:

- (*G-Int$_1$*): it results $P = P_1 \parallel P_2$. By inductive hypothesis there exists M_1 such that $M_1 \equiv P_1 \cdot \sigma$, $M_1 \xrightarrow{\ell}_L{}^{+} M_1'$ and $M_1' \equiv P_1 \cdot \sigma'$. Now, by definition of syntax correspondence $(P_1 \parallel P_2) \cdot \sigma = P_1 \cdot \sigma \parallel P_2 \cdot \sigma$. Considering $M = M_1 \parallel M_2 \equiv P_1 \cdot \sigma \parallel P_2 \cdot \sigma.$, then, by definition of the transitive closure of the labelled transition relation, $M_1 \xrightarrow{\ell}_L{}^{+} M_1'$ means $M_1 \xrightarrow{\ell_1}_L M^1 \xrightarrow{\ell_2}_L M^2 \ldots \xrightarrow{\ell_n}_L M^n = M_1'$, for some ℓ_i and M^i. By repetitively applying rule *M-StatusUpd$_1$*, $M_1 \parallel M_2 \xrightarrow{\ell_1}_L M^1 \parallel M_2 \star \ell_1 \xrightarrow{\ell_2}_L M^2 \parallel M_2 \star \ell_1 \star \ell_2 \ldots \xrightarrow{\ell_n}_L M^n \parallel M_2 \star \ell_1 \star \ell_2 \ldots \star \ell_n = M'$. To conclude it must be shown that $M' \equiv P \cdot \sigma' = P_1 \cdot \sigma' \parallel P_2 \cdot \sigma'$. Since by inductive hypothesis it results that $M_1' \equiv P_1 \cdot \sigma'$, it must just be shown that $M_2 \star \ell_1 \star \ell_2 \ldots \star \ell_n \equiv P_2 \cdot \sigma'$. This holds by applying Theorem 5.

□

Theorem 11. *Let M be a reachable marked process, with $M \equiv P \cdot \sigma$, if $M \overset{s}{\Rightarrow}_L M'$, where $s = \ell_1, \ldots, \ell_n$ is such that $\forall i = 1, \ldots, n-1$, ℓ_i is of the form $(\mathsf{d} : E_2)$ and ℓ_n is of the form $\ell_n = (\mathsf{w} : E_1, \mathsf{d} : E_2, \mathsf{l} : E_3)$ with $E_3 \neq \varnothing$ then $\langle P, \sigma \rangle \rightarrow_G \sigma'$ and $M' \equiv P \cdot \sigma'$.*

Proof. By induction on the derivation of $M \overset{s}{\Rightarrow}_L M'$. In particular, it is sufficient to consider only the last action that has produced the transition. In fact when the local semantics produces a propagation of only dead status no transition is produced by the global semantics. Base cases (only the most interesting cases are shown here, since the others are similar):

- (*L-Start-NC*): By hypothesis M is a reachable marked process, thus it follows that $M = \mathsf{start}(\mathsf{e.nc.w})$ and $M' = \mathsf{start}(\mathsf{e.nc.l})$, with $s = \ell_n = (\mathsf{l} : \mathsf{e})$. By the syntax correspondence we get $M \equiv P \cdot \sigma = \mathsf{start}(\mathsf{e.t.nl}) = \mathsf{start}(\mathsf{e}) \cdot \sigma$ with $\sigma(\mathsf{e}) = 0$ and $M' \equiv P \cdot \sigma'' = \mathsf{start}(\mathsf{e.t.l}) = \mathsf{start}(\mathsf{e}) \cdot \sigma''$, with $\sigma''(\mathsf{e}) = 1$. We consider only the case $\langle P, \sigma \rangle = \langle \mathsf{start}(\mathsf{e}), \sigma_0 \rangle$. By definition of σ_0, $\sigma_0(\mathsf{e}) = 0$. It can be applied rule *G-Start* producing $\langle P, \sigma \rangle \rightarrow_G \sigma'$ with $\sigma' = inc(\sigma_0, \mathsf{e})$. It must be shown that $P \cdot \sigma'' = P \cdot \sigma'$. One can conclude as, by definition of function *inc*, $\sigma'(\mathsf{e}) = 1$.

- (*L-End-NC*): By hypothesis M is a reachable marked process, thus it follows that $M = \mathsf{end}(\mathsf{e.nc.l}, \mathsf{e'.nc.w})$, $M' = \mathsf{end}(\mathsf{e.nc.d}, \mathsf{e'.nc.l})$, with $s = \ell_n = ((\mathsf{d} : \mathsf{e}, \mathsf{l} : \mathsf{e}'))$. By the syntax correspondence $M \equiv P \cdot \sigma = \mathsf{end}(\mathsf{e.t.l}, \mathsf{e'.t.nl}) = \mathsf{end}(\mathsf{e}, \mathsf{e}') \cdot \sigma$ with $\sigma(\mathsf{e}) = 1$, $\sigma(\mathsf{e}') = 0$ and $M' \equiv P \cdot \sigma'' = \mathsf{end}(\mathsf{e.t.nl}, \mathsf{e'.t.l}) = \mathsf{end}(\mathsf{e}, \mathsf{e}') \cdot \sigma''$, with $\sigma''(\mathsf{e}) = 0$, $\sigma''(\mathsf{e}') = 1$. Thus, considering $\langle P, \sigma \rangle = \langle \mathsf{end}(\mathsf{e}, \mathsf{e}'), \sigma \rangle$, with $\sigma(\mathsf{e}) = 1$, by applying rule *G-End* it results $\langle P, \sigma \rangle \rightarrow_G \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$. It must be shown that $P \cdot \sigma'' = P \cdot \sigma'$. One can conclude as, by definition the functions *dec* and *inc* it follows that $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$.

- there are two live rules for the AND-Split gateway: *L-AndSplit-NC* and *L-AndSplit-C*.

  - (*L-AndSplit-NC*): By hypothesis M is a reachable marked process, thus it follows that $M = \mathsf{andSplit}(\mathsf{e.nc.l}, E)$, $M' = \mathsf{andSplit}(\mathsf{e.nc.d}, setLive(E))$, with $(\mathsf{d} : \mathsf{e}, \mathsf{l} : E)$. By definition of syntax correspondence $M \equiv P \cdot \sigma = \mathsf{andSplit}(\mathsf{e.t.l}, (E \cdot \sigma)) = \mathsf{andSplit}(\mathsf{e}, E) \cdot \sigma$, where $E \cdot \sigma$ is such that $\forall \mathsf{e'.t.}(\mathsf{e}' \cdot \sigma') \in E$, $\mathsf{e'.t.nl} \in E$, and $M' \equiv P \cdot \sigma' = \mathsf{andSplit}(\mathsf{e.t.nl}, (E \cdot \sigma'')) = \mathsf{andSplit}(\mathsf{e}, E) \cdot \sigma''$, where $\forall \mathsf{e'.t.}(\mathsf{e}' \cdot \sigma') \in E$, $\mathsf{e'.t.l} \in E$. This means $\sigma(\mathsf{e}) = 1$, $\forall \mathsf{e}' \in E \sigma(\mathsf{e}') = 0$ and $\sigma''(\mathsf{e}) = 0$, $\forall \mathsf{e}' \in E\, \sigma''(\mathsf{e}') = 1$. Thus, considering $\langle P, \sigma \rangle = \langle \mathsf{andSplit}(\mathsf{e}, E), \sigma \rangle$ with $\sigma(\mathsf{e}) = 1$, by applying rule *G-AndSplit*, it follows $\langle P, \sigma \rangle \rightarrow_G \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}), E)$. It must be shown that $P \cdot \sigma'' = P \cdot \sigma'$. One can conclude as by definition of functions *inc* and *dec* it follows $\sigma'(\mathsf{e}) = 0$, $\forall \mathsf{e}' \in E\, \sigma'(\mathsf{e}') = 1$.

  - (*L-AndSplit-C*): By hypothesis M is a reachable marked process, thus it follows $M = \mathsf{andSplit}(\mathsf{e.c.l}, E_1 \sqcup E_2)$, $M' = \mathsf{andSplit}(\mathsf{e.c.w}, setLive(E))$, with $(\mathsf{w} : \mathsf{e}, \mathsf{l} : E)$. By definition of the syntax correspondence, proceeding as in the previous case, it follows $\sigma(\mathsf{e}) = 1$ and $\sigma''(\mathsf{e}) = 0$, $\forall \mathsf{e}' \in E\, \sigma''(\mathsf{e}') = 1$. Thus, considering $\langle P, \sigma \rangle = \langle \mathsf{andSplit}(\mathsf{e}, E), \sigma \rangle$, with $\sigma(\mathsf{e}) = 1$, by applying rule *G-AndSplit*, and by definition of *inc* and *dec* it follows that $\sigma'(\mathsf{e}) = 0$ and $\forall \mathsf{e}' \in E\, \sigma'(\mathsf{e}') = 1$, that permits to conclude.

- For the AND-Join gateway there are two rules, *L-AndJoin-NC*, *L-AndJoin-C*.

- (L-*AndJoin-NC*): By hypothesis M is a reachable marked process, thus it follows $M = \mathsf{andJoin}(E, \mathsf{e.nc.}\Sigma)$ with $isLive(E)$, $M' = \mathsf{andJoin}(setDead(E), \mathsf{e.nc.l})$. By definition of syntax correspondence it follows $M \equiv P \cdot \sigma = \mathsf{andJoin}((E \cdot \sigma), \mathsf{e.t.nl}) = \mathsf{andJoin}(E, \mathsf{e}) \cdot \sigma$, where $E \cdot \sigma$ is such that $\forall \mathsf{e}'.\mathsf{t}.(\mathsf{e}' \cdot \sigma') \in E\, \mathsf{e}'.\mathsf{t.l} \in E$, and $M' \equiv P \cdot \sigma'' = \mathsf{andJoin}((E \cdot \sigma''), \mathsf{e.t.l}) = \mathsf{andJoin}(E, \mathsf{e}) \cdot \sigma''$, where $\forall \mathsf{e}'.\mathsf{t}.(\mathsf{e}' \cdot \sigma'') \in E\mathsf{e}'.\mathsf{t.nl} \in E$. Thus, by syntax correspondence, $\forall \mathsf{e}' \in E\, \sigma(\mathsf{e}') = 1$ and $\sigma''(\mathsf{e}) = 1$, $\forall \mathsf{e}' \in E\, \sigma''(\mathsf{e}') = 0$. Considering the process configuration $\langle P, \sigma \rangle = \langle \mathsf{andJoin}(E, \mathsf{e}), \sigma \rangle$, with $\forall \mathsf{e}' \in E.\sigma(\mathsf{e}') = 1$, by applying rule *G-AndJoin*, $\langle P, \sigma \rangle \rightarrow_G \sigma'$ with $\sigma' = inc(dec(\sigma, E), \mathsf{e})$. It must be shown that $P \cdot \sigma'' = P \cdot \sigma'$. One can conclude as by definition of functions *inc* and *dec* it results $\sigma'(\mathsf{e}) = 1$ and $\forall \mathsf{e}' \in E\, \sigma'(\mathsf{e}') = 0$.
- (L-*AndJoin-C*): it follows $M = \mathsf{andJoin}(E_1 \sqcup E_2, \mathsf{e.c.}\Sigma)$, $M' = \mathsf{andJoin}(E_1 \sqcup setDead(E_2), \mathsf{e.c.l})$, with $isLive(E), isC(E_1), isNC(E_2)$. One can reason as in the previous case up to different instantiation of nl and t.

- For the OR-Join gateway there are three rules.
  - (L-*OrJoin-NC*): it follows that $M = \mathsf{orJoin}(E_1 \sqcup E_2, \mathsf{e.nc.}\Sigma)$, $M' = \mathsf{orJoin}(setDead(E), \mathsf{e.nc.l})$, with $E_1 \neq \emptyset$, $isLive(E_1)$, $isDead(E_2)$. By definition of syntax correspondence it follows $M \equiv P \cdot \sigma = \mathsf{orJoin}((E_1 \cdot \sigma) \sqcup (E_2 \cdot \sigma), \mathsf{e.t.nl}) = \mathsf{orJoin}(E_1 \sqcup E_2, \mathsf{e}) \cdot \sigma$ where $E_1 \cdot \sigma$ is such that $\forall \mathsf{e}'.\mathsf{t}.(\mathsf{e}' \cdot \sigma) \in E_1\, \mathsf{e}'.\mathsf{t.l} \in E_1$ and $E_2 \cdot \sigma$ is such that $\forall \mathsf{e}'.\mathsf{t}.(\mathsf{e}' \cdot \sigma) \in E_2\, \mathsf{e}'.\mathsf{t.nl} \in E_2$. Thus, by syntax correspondence, it follows $\forall \mathsf{e}' \in E_1.\sigma(\mathsf{e}') = 1$ and $\forall \mathsf{e}'' \in E_2.\sigma(\mathsf{e}'') = 0$. The transition produces $M' \equiv P \cdot \sigma'' = \mathsf{orJoin}((E_1 \cdot \sigma'') \sqcup (E_2 \cdot \sigma''), \mathsf{e.t.l})$ where $E_1 \cdot \sigma''$ and $E_2 \cdot \sigma''$ are such that $\forall \mathsf{e}'.\mathsf{t}.(\mathsf{e}' \cdot \sigma'') \in E_1 \sqcup E_2$ then $\mathsf{e}'.\mathsf{t.nl} \in E_1 \sqcup E_2$. Thus, $\sigma''(\mathsf{e}) = 1$ and $\forall \mathsf{e}' \in E_1 \sqcup E_2.\sigma''(\mathsf{e}') = 0$. Considering $\langle P, \sigma \rangle = \langle \mathsf{orJoin}(E_1 \sqcup E_2 \mathsf{e}, ,)\sigma \rangle$ with $E_1 \neq \emptyset$, $\forall \mathsf{e}' \in E_1.\sigma(\mathsf{e}') = 1$, by applying rule (*G-OrJoin*) it follows $\sigma' = inc(dec(\sigma, E_1), \mathsf{e})$. It must only be shown that $P \cdot \sigma'' = P \cdot \sigma'$. One can conclude as by definition of functions *inc* and *dec* it follows $\sigma'(\mathsf{e}) = 1$ and $\forall \mathsf{e}' \in E_1.E_2\, \sigma'(\mathsf{e}') = 0$.
  - (L_1-*OrJoin-C*): it follows $M = \mathsf{orJoin}(E_1 \sqcup E_2, \mathsf{e.c.}\Sigma)$, $M' = \mathsf{orJoin}(E_1 \sqcup setDead(E_2), \mathsf{e.c.l})$, with $E_1 \neq \emptyset, isLive(E_1), isDead(E_2), isC(E_1), isNC(E_2), Dep(e) = \emptyset$. One can reason as in the previous case.
  - (L_2-*OrJoin-C*): it follows $M = \mathsf{orJoin}(E_1 \sqcup E_2, \mathsf{e.c.}\Sigma)$, $M' = \mathsf{orJoin}(setDead(E_1) \sqcup E_2, \mathsf{e.c.l})$ with $E_1 \neq \emptyset, isLive(E_1)$, $isDead(E_2), isNC(E_1), isC(E_2), Dep(e) = \emptyset$. One can proceed as in the previous case.

Inductive case:

- (M-*StatusUpd*$_1$): $M = (M_1 \parallel M_2)$ and $M' = (M_1' \parallel M_2')$, with $M \equiv P \cdot \sigma = P_1 \cdot \sigma \parallel P_2 \cdot \sigma$. By inductive hypothesis, $M_1 \stackrel{s}{\Rightarrow}_L M_1'$, implies $\langle P_1, \sigma \rangle \rightarrow_G \sigma'$

and $M_1' \equiv P_1 \cdot \sigma'$. By applying (*G-Int$_1$*), $\langle P_1 \parallel P_2, \sigma\rangle \rightarrow_G \sigma'$. By the definition of syntax correspondence, $\langle P_1 \parallel P_2, \sigma\rangle \rightarrow_G \sigma' \equiv (P_1 \parallel P_2) \cdot \sigma' = P_1 \cdot \sigma' \parallel P_2 \cdot \sigma' = M_1' \parallel M_2''$. Since by inductive hypothesis $M_1' \equiv P_1 \cdot \sigma'$ it must be only shown that $M_2' = M_2''$. Since M_2' differs from M only for the annotations marking the edges in ℓ, by applying Theorem 5 one can conclude.

□

C.3 Sub-Processes

This appendix reports the proofs of the results presented in Chapter 6.

Lemma 6. *Let is$WSCore(P)$, and let $\langle P, \sigma\rangle$ be a core reachable and cs-safe process configuration, if $\langle P, \sigma\rangle \xrightarrow{\ell} \sigma'$ then $\langle P, \sigma'\rangle$ is cs-safe.*

Proof. It follows by induction on the structure of WSCore process elements. The proof is equal to the one reported in Appendix C.1, with the exception of considering the sub-process element. Thus, here only the inductive case including the sub-process is shown, that is $\mathsf{subProc}(\mathsf{e}, \mathsf{start}(\mathsf{e}', \mathsf{e}'') \| P_1' \parallel \mathsf{end}(\mathsf{e}''', \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}})$.

By hypothesis this is a cs-safe process configuration, then $edgesEl(\mathsf{subProc}(\mathsf{e}, \mathsf{start}(\mathsf{e}', \mathsf{e}'') \| P_1' \parallel \mathsf{end}(\mathsf{e}''', \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}})) = \{\mathsf{e}, \mathsf{e}'', \mathsf{e}''', \mathsf{e}^{\mathsf{v}}\} \cup edgesEl(P_1')$ are such that $\forall \mathsf{e}^{\mathsf{vi}} \in edgesEl(\mathsf{subProc}(\mathsf{e}, \mathsf{start}(\mathsf{e}', \mathsf{e}'') \| P_1' \parallel \mathsf{end}(\mathsf{e}''', \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}})) \,.\, \sigma(\mathsf{e}^{\mathsf{vi}}) \leqslant 1$. There are the following possibilities:

- $\langle \mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{\mathsf{v}}), \sigma\rangle$ 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. $\sigma(\mathsf{e}) = 1$. It can be exploited the fact that this is a reachable process configuration to prove that $\sigma(\mathsf{e}^{\mathsf{v}}) = 0$ and $\sigma(edges(P_1)) = 0$. Thus, $\langle \mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{\mathsf{v}}), \sigma\rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}), start(P_1))$. Hence, $\forall \mathsf{e}^{\mathsf{vi}} \in edgesEl(\mathsf{subProc}(\mathsf{e}, \mathsf{start}(\mathsf{e}', \mathsf{e}'') \,\|\, P_1' \parallel \mathsf{end}(\mathsf{e}''', \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}})) \,.\, \sigma'(\mathsf{e}^{\mathsf{vi}}) \leqslant 1$. By hypothesis $\langle \mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{\mathsf{v}}), \sigma\rangle$ 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-SubProcEvolution*, *P-SubProcEnd* or *P-SubProcKill*. In all the cases one can conclude by relying on the inductive hypothesis and on the fact only core reachable configurations are considered.

□

Lemma 7. *Let is$WSCore(P)$ 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. By induction on the structure of $isWSCore(P)$. The proof is equal to the one reported in Appendix C.1, with the exception of considering the sub-process element. Thus, here only the inductive case including the sub-process is shown.

Thus, it results $\mathsf{subProc}(\mathsf{e}, \mathsf{start}(\mathsf{e}', \mathsf{e}'') \parallel P_1' \parallel \mathsf{end}(\mathsf{e}''', \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}})$ with $isWSCore(P_1')$, $in(P_1') = \{\mathsf{e}''\}, out(P_1') = \{\mathsf{e}'''\}$. Called $P_1 = \mathsf{start}(\mathsf{e}', \mathsf{e}'') \parallel P_1' \parallel \mathsf{end}(\mathsf{e}''', \mathsf{e}^{\mathsf{iv}})$, the overall term becomes $\mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{\mathsf{v}})$. Now:

- $\mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{\mathsf{v}})$ evolves by means of rule *P-SubProcStart*. It can be exploited 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 $\sigma_3'(end(P_1)) = \sigma_3'(\mathsf{e}^{\mathsf{iv}}) = 1$ and $\forall \mathsf{e} \in edges(P_1)\backslash end(P_1) \,.\, \sigma(\mathsf{e}) = 0$. Indeed, predicate $completed(P_1, \sigma_3')$ holds. Now rule *P-SubProcEnd* can be applied, producing $\langle \mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{\mathsf{v}}), \sigma_3' \rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(reset(\sigma, end(P_1)), \mathsf{e}^{\mathsf{v}})$ that permits to conclude.

□

Theorem 12. *Let P be a collaboration, P is unsafe does not imply P is unsound.*

Proof. Let P be a unsafe collaboration. Supposing that P is unsound, it is sufficient to show a counter example, i.e. a unsafe collaboration that is sound. One can consider the process in Figure C.4. It is unsafe since the AND-Split gateway creates two tokens that are then merged by the XOR-Join gateway producing two tokens on the outgoing edge of the XOR-Join. However, after *Task C* is executed and one token enables the terminate end event, the *kill* label is produced and the second token in the sequence flow is removed (rule *P-Terminate*), rendering the process sound. □

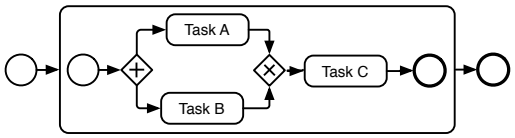


Figure C.4: Example of Unsafe but Sound Process.

Theorem 13. *Let C be a collaboration, C is unsafe does not imply C is unsound.*

Proof. Let C be a unsafe collaboration. Supposing that C is unsound, it is sufficient to show a counter example, i.e. a unsafe collaboration that is sound. One can consider the collaboration in Figure C.5. 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. □

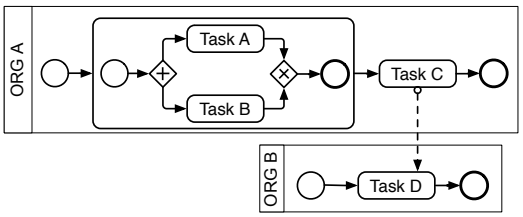


Figure C.5: Example of Unsafe but Sound Collaboration.

Theorem 14. *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. Supposing that $P = \mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}') \parallel P_2$, by contradiction let P be safe, i.e. given σ such that $\mathit{isInit}(P, \sigma)$, for all σ' such that $\langle P, \sigma\rangle \rightarrow^* \sigma'$ then $\langle P, \sigma'\rangle$ is cs-safe. By hypothesis P_1 is unsafe, i.e. given σ'_1 such that $\mathit{isInit}(P_1, \sigma'_1)$, 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, $\exists \mathsf{e}''' \in \mathit{edgesEl}(P_1)\ .\ \sigma'_2(\mathsf{e}''') \geqslant 1$. By definition of function $\mathit{edgesEl}(\cdot)$, $\mathit{edgesEl}(P) = \mathit{edgesEl}(\mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}')) \cup \mathit{edgesEl}(P_2)$. By safeness of P it follows that given σ such that $\mathit{isInit}(P, \sigma)$, for all σ' such that $\langle P, \sigma\rangle \rightarrow^* \sigma'$ it follows that $\langle P, \sigma'\rangle$ is such that $\forall \mathsf{e} \in \mathit{edgesEl}(P)\,.\,\sigma'(\mathsf{e}) \leqslant 1$. Choosing $\sigma' = \sigma'_2$ it follows that $\exists \mathsf{e}''' \in \mathit{edgesEl}(P)\ .\ \sigma'_2(\mathsf{e}''') \geqslant 1$. Thus, P is not cs-safe, which is a contradiction. □

Theorem 15. *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. Supposing that P is unsound., then, it is sufficient to show a counter example, i.e. an sound process including an unsound sub-process. One can consider process in Figure C.6. 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. Including the part of the model generating multiple tokens in the scope of a sub-process, as it is shown in Figure C.7, 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. □

C.4 Multiple Instances and Data

This appendix reports the proofs of the results presented in Chapter 7.

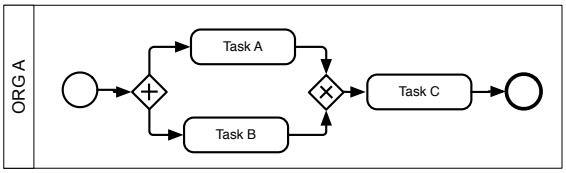


Figure C.6: Example of Unsound Process.

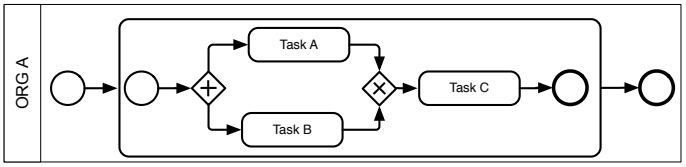


Figure C.7: Example of Sound Process with Unsound Sub-Process.

Lemma 8. *Let P be a process, if $isInit(P, \sigma, \alpha)$ then $\langle P, \sigma, \alpha \rangle$ is cs-safe.*

Proof. Trivially, from definition of $isInit(P, \sigma, \alpha)$. By definition it results 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, it follows that $\forall \mathsf{e} \in edgesEl(P)\ .\ \sigma(\mathsf{e}) \leqslant 1$, which allows to conclude. □

Lemma 9. *Let $isWSCore(P)$, and let $\langle P, \sigma, \alpha \rangle$ be a core reachable and cs-safe process configuration, if $\langle P, \sigma, \alpha \rangle \xrightarrow{\ell} \langle \sigma', \alpha' \rangle$ then $\langle P, \sigma', \alpha' \rangle$ is cs-safe.*

Proof. By induction on the structure of WSCore process elements. Since the data state function α does not affect the above definitions, its evolution will be not taken into account in the proof.
Base cases: since by hypothesis $isWSCore(P)$, it can only be either a task or an intermediate event.

- $P = \mathsf{task}(\mathsf{e}, \mathsf{exp}, A, \mathsf{e}')$. By hypothesis $\langle P, \sigma, \alpha \rangle$ is cs-safe, then $edgesEl(P) = edgesEl(\mathsf{task}(\mathsf{e}, \mathsf{exp}, A, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$ 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 $\langle P, \sigma, \alpha \rangle \xrightarrow{\ell} \langle \sigma', \alpha' \rangle$ is *P-Task*. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$, $eval(\mathsf{exp}, \alpha, true)$, $upd(\alpha, A, \alpha')$; hence $0 < \sigma(\mathsf{e}) \leqslant 1$, i.e. $\sigma(\mathsf{e}) = 1$. It can be exploited the fact that $\langle P, \sigma, \alpha \rangle$ be is a core reachable configuration to prove that $\sigma(\mathsf{e}') = 0$. In terms of edge state fuhnction evolution, 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, it follows that $\forall \mathsf{e}''' \in edgesEl(P)\ .\ \sigma'(\mathsf{e}''') \leqslant 1$, which allows to conclude.

- $P = \mathsf{taskRcv}(\mathsf{e}, \mathsf{exp}, A, \mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}')$. By hypothesis $\langle P, \sigma, \alpha \rangle$ is cs-safe, then $edgesEl(P) = edgesEl(\mathsf{taskRcv}(\mathsf{e}, \mathsf{exp}, A, \mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$ 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 $\langle P, \sigma, \alpha \rangle \xrightarrow{\ell} \langle \sigma', \alpha' \rangle$ is *P-TaskRcv*. In order to apply the rule there

must be $\sigma(\mathsf{e}) > 0$,$eval(\mathsf{exp}, \alpha, true)$, $eval(\tilde{\mathsf{t}}, \alpha, \tilde{\mathsf{et}})$; hence $0 < \sigma(\mathsf{e}) \leqslant 1$, i.e. $\sigma(\mathsf{e}) = 1$. It can be exploited the fact that $\langle P, \sigma, \alpha\rangle$ is a core reachable configuration to prove that $\sigma(\mathsf{e}') = 0$. In terms of edge state function evolution, 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, it follows that $\forall \mathsf{e}''' \in edgesEl(P)$. $\sigma'(\mathsf{e}''') \leqslant 1$, which allows to conclude.

- $P = \mathsf{taskSnd}(\mathsf{e}, \mathsf{exp}', A, \mathsf{m}:\tilde{\mathsf{exp}}, \mathsf{e}')$. By hypothesis $\langle P, \sigma, \alpha\rangle$ is cs-safe, then $edgesEl(P) = edgesEl(\mathsf{taskSnd}(\mathsf{e}, \mathsf{exp}', A, \mathsf{m}:\tilde{\mathsf{exp}}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$ 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 $\langle P, \sigma, \alpha\rangle \xrightarrow{\ell} \langle \sigma', \alpha'\rangle$ is *P-TaskSnd*. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$,$eval(\mathsf{exp}', \alpha, true)$, $upd(\alpha, A, \alpha')$, $eval(\tilde{\mathsf{exp}}, \alpha, \tilde{\mathsf{v}})$; hence $0 < \sigma(\mathsf{e}) \leqslant 1$, i.e. $\sigma(\mathsf{e}) = 1$. It can be exploited the fact that $\langle P, \sigma\rangle$ 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, it follows that $\forall \mathsf{e}''' \in edgesEl(P)$. $\sigma'(\mathsf{e}''') \leqslant 1$, which allows to conclude.

- $P = \mathsf{interRcv}(\mathsf{e}, \mathsf{m}:\tilde{\mathsf{t}}, \mathsf{e}')$. By hypothesis $\langle P, \sigma\rangle$ is cs-safe, then $edgesEl(P) = edgesEl(\mathsf{interRcv}(\mathsf{e}, \mathsf{m}:\tilde{\mathsf{t}}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$ 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 $\langle P, \sigma, \alpha\rangle \xrightarrow{\ell} \langle \sigma', \alpha'\rangle$ is *P-InterRcv*. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$,$eval(\tilde{\mathsf{t}}, \alpha, \tilde{\mathsf{et}})$; hence $0 < \sigma(\mathsf{e}) \leqslant 1$, i.e. $\sigma(\mathsf{e}) = 1$. It can be exploited the fact that $\langle P, \sigma\rangle$ 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, it follows that $\forall \mathsf{e}''' \in edgesEl(P)$. $\sigma'(\mathsf{e}''') \leqslant 1$, which allows to conclude.

- $P = \mathsf{interSnd}(\mathsf{e}, \mathsf{m}:\tilde{\mathsf{exp}}, \mathsf{e}')$. By hypothesis $\langle P, \sigma, \alpha\rangle$ is cs-safe, then $edgesEl(P) = edgesEl(\mathsf{interSnd}(\mathsf{e}, \mathsf{m}:\tilde{\mathsf{exp}}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$ 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 $\langle P, \sigma, \alpha\rangle \xrightarrow{\ell} \langle \sigma', \alpha'\rangle$ is *P-InterSnd*. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$, $eval(\tilde{\mathsf{exp}}, \alpha, \tilde{\mathsf{v}})$; hence $0 < \sigma(\mathsf{e}) \leqslant 1$, i.e. $\sigma(\mathsf{e}) = 1$. It can be exploited 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, it follows that $\forall \mathsf{e}''' \in edgesEl(P)$. $\sigma'(\mathsf{e}''') \leqslant 1$, which allows 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 $\langle P, \sigma, \alpha\rangle \xrightarrow{\ell} \langle \sigma', \alpha'\rangle$ is *P-Empty*. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$; hence $0 < \sigma(\mathsf{e}) \leqslant 1$, i.e. $\sigma(\mathsf{e}) = 1$. It can be exploited 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, it follows that $\forall \mathsf{e}''' \in \mathit{edgesEl}(P)\ .\ \sigma'(\mathsf{e}''') \leqslant 1$, which allows to conclude.

Inductive cases:

- $\langle \mathsf{andSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{andJoin}(E', \mathsf{e}''), \sigma, \alpha \rangle$, with $\forall j \in [1..n]$ $\mathit{isWSCore}(P_j)$, $\mathit{in}(P_j) \subseteq E$, $\mathit{out}(P_j) \subseteq E'$. There are the following possibilities:

  - $\langle \mathsf{andSplit}(\mathsf{e}, E), \sigma, \alpha \rangle$ evolves by means of rule *P-AndSplit*. It can be exploited 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, $\langle \mathsf{andSplit}(\mathsf{e}, E), \sigma, \alpha \rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = \mathit{inc}(\mathit{dec}(\sigma, \mathsf{e}), E)$. Hence, $\forall \mathsf{e}''' \in \mathit{edgesEl}(\mathsf{andSplit}(\mathsf{e}, E))\ .\ \sigma(\mathsf{e}''') \leqslant 1$. By hypothesis $\langle \mathsf{andSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{andJoin}(E', \mathsf{e}''), \sigma, \alpha \rangle$ 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 $\langle \mathsf{andSplit}(\mathsf{e}, E), \sigma', \alpha' \rangle$, then all the other edges are unmarked. This means that cs-safeness is not affected. Therefore, the overall term $\langle \mathsf{andSplit}(E, \mathsf{e}) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{andJoin}(E', \mathsf{e}''), \sigma', \alpha' \rangle$ is cs-safe.
  - Node $P_1 \parallel \ldots \parallel P_n$ evolves without affecting the split and join gateways. In this case it easily follows by inductive hypothesis.
  - Node $P_1 \parallel \ldots \parallel P_n$ evolves and affects the split and/or join gateways. In this case one 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*. It can be exploited 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 $\langle \mathsf{andJoin}(E', \mathsf{e}''), \sigma, \alpha \rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = \mathit{inc}(\mathit{dec}(\sigma, E'), \mathsf{e}'')$. Hence, $\forall \mathsf{e}''' \in \mathit{edgesEl}(\mathsf{andJoin}(E', \mathsf{e}''))\ .\ \sigma(\mathsf{e}''') \leqslant 1$. By hypothesis $\langle \mathsf{andSplit}(E, \mathsf{e}) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{andJoin}(E', \mathsf{e}''), \sigma, \alpha \rangle$ is cs-safe, i.e. if there is a token on the outgoing edge of the AND-Join in the state $\langle \mathsf{andJoin}(E', \mathsf{e}''), \sigma', \alpha' \rangle$ all the other edges do not have tokens. This means that cs-safeness is not affected. Therefore, the overall term $\langle \mathsf{andSplit}(E, \mathsf{e}) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{andJoin}(E', \mathsf{e}''), \sigma', \alpha' \rangle$ is cs-safe.

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

  - $\langle \mathsf{xorSplit}(\mathsf{e}, G), \sigma, \alpha \rangle$ evolves by means of rule *P-XorSplit$_1$* or *P-XorSplit$_1$*. Now it is considered the case it evolves by means of rule *P-XorSplit$_1$*, since the otehr is similar. It can be exploited the fact that this is a core reachable well-structured configuration to prove that $\sigma(\mathsf{e}) = 1$ and $\forall (\mathsf{e}', \mathsf{exp}) \in G\ .\sigma(\mathsf{e}') = 0$. Thus, $\mathsf{xorSplit}(\mathsf{e}, \{(\mathsf{e}''', \mathsf{exp})\} \cup G), \sigma, \alpha \rangle \xrightarrow{\epsilon} \sigma'$, with $\sigma' = \mathit{inc}(\mathit{dec}(\sigma, \mathsf{e}), \mathsf{e}''')$. Hence, $\forall \mathsf{e}^{\mathsf{iv}} \in \mathit{edgesEl}(\mathsf{xorSplit}(\mathsf{e}, G))\ .\ \sigma(\mathsf{e}^{\mathsf{iv}}) \leqslant 1$. By hypothesis

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

- Node $P_1 \parallel \ldots \parallel P_n$ evolves without affecting the split and join gateways. In this case it easily follows by inductive hypothesis.
- Node $P_1 \parallel \ldots \parallel P_n$ evolves and affects the split and/or join gateways. In this case one can reason like in the first case, by relying on inductive hypothesis.
- $\mathsf{xorJoin}(\{\mathsf{e}\} \cup E, \mathsf{e}''), \sigma, \alpha \rangle$ evolves by means of rule *P-XorJoin*. It can be exploited 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 $\langle \mathsf{xorJoin}(\{\mathsf{e}\} \cup E, \mathsf{e}''), \sigma, \alpha \rangle \xrightarrow{\epsilon} \sigma'$, 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 $\langle \mathsf{xorSplit}(\mathsf{e}, G) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E, \mathsf{e}''), \sigma, \alpha \rangle$ is cs-safe, i.e. if there is a token on the outgoing edge of the XOR-Join in the state $\langle \mathsf{xorJoin}(\{\mathsf{e}\} \cup E, \mathsf{e}''), \sigma', \alpha' \rangle$ all the other edges do not have tokens. This means that cs-safeness is not affected. Therefore, the overall term $\langle \mathsf{xorSplit}(\mathsf{e}, G) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E, \mathsf{e}''), \sigma', \alpha' \rangle$ is cs-safe.

- $\mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j : \tilde{\mathsf{t}_j}, \mathsf{e}'_j) | j \in [1..n]\}) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E, \mathsf{e}'')$, with $\forall j \in [1..n] \quad isWSCore(P_j), \quad in(P_j) = \mathsf{e}'_j, \quad out(P_j) \subseteq E$. There are the following possibilities:
  - $\langle \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j : \tilde{\mathsf{t}_j}, \mathsf{e}'_j) | j \in [1..n]\}), \sigma, \alpha \rangle$ evolves by means of rule *P-EventG*. It can be exploited 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, $\langle \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j : \tilde{\mathsf{t}_j}, \mathsf{e}'_j) | j \in [1..n]\}), \sigma, \alpha \rangle \xrightarrow{?\mathsf{m}_j : \tilde{\mathsf{et}_j}, \epsilon} \sigma', \alpha'$, with $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}'_j)$. Thus, $\forall \mathsf{e}''' \in edgesEl(\mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j : \tilde{\mathsf{t}_j}, \mathsf{e}'_j) | j \in [1..n]\}))$. $\sigma(\mathsf{e}''') \leqslant 1$. By hypothesis $\langle \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j : \tilde{\mathsf{t}_j}, \mathsf{e}'_j) | j \in [1..n]\}), \sigma, \alpha \rangle$ 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 $\langle \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j : \tilde{\mathsf{t}_j}, \mathsf{e}'_j) | j \in [1..n]\}), \sigma', \alpha' \rangle$, then all the other edges are unmarked. This means that cs-safeness is not affected. Therefore, the overall term $\langle \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j : \tilde{\mathsf{t}_j}, \mathsf{e}'_j) | j \in [1..n]\}), \parallel P_1 \parallel \ldots \mid P_n \parallel \mathsf{xorJoin}(E, \mathsf{e}''), \sigma', \alpha' \rangle$ is cs-safe.
  - Node $P_1 \parallel \ldots \parallel P_n$ evolves without affecting the split and join gateways. In this case it easily follows by inductive hypothesis.
  - Node $P_1 \parallel \ldots \parallel P_n$ evolves and affects the split and/or join gateways. In this case one can reason like in the first case, by relying on inductive hypothesis.

– $\langle \mathsf{xorJoin}(\{\mathsf{e}\} \cup E, \mathsf{e}''), \sigma, \alpha\rangle$ evolves by means of rule *P-XorJoin*. It can be exploited 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, $\langle \mathsf{xorJoin}(\{\mathsf{e}\} \cup E, \mathsf{e}''), \sigma, \alpha\rangle \xrightarrow{\epsilon} \sigma', \alpha'$, 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 $\langle \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j : \tilde{\mathsf{t}_\mathsf{j}}, \mathsf{e}'_j) | j \in [1..n]\}), \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E', \mathsf{e}''), \sigma, \alpha\rangle$ is cs-safe, i.e. if there is a token on the outgoing edge of the XOR-Join in the state $\langle \mathsf{xorJoin}(\{\mathsf{e}\} \cup E, \mathsf{e}'), \sigma', \alpha'\rangle$ all the other edges do not have tokens. This means that cs-safeness is not affected. Therefore, the overall term $\langle \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j : \tilde{\mathsf{t}_\mathsf{j}}, \mathsf{e}'_j) | j \in [1..n]\}), \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E, \mathsf{e}''), \sigma', \alpha'\rangle$ is cs-safe.

- $\mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{(\mathsf{e}^{\mathsf{v}}, \mathsf{exp}_1), (\mathsf{e}^{\mathsf{vi}}, \mathsf{exp}_2)\})$ 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:

  – $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}'), \sigma, \alpha\rangle$ evolves by means of rule *P-XorJoin*. It can be exploited 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 $\sigma(\mathsf{e}''') = 1$; assuming that the marking is $\sigma(\mathsf{e}''') = 1$ (since the other case is similar), then $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}'), \sigma, \alpha\rangle \xrightarrow{\epsilon} \sigma'$ 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 $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{(\mathsf{e}^{\mathsf{v}}, \mathsf{exp}_1), (\mathsf{e}^{\mathsf{vi}}, \mathsf{exp}_2)\}), \sigma, \alpha\rangle$ is cs-safe, i.e. if there is a token on e' in the state $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}'), \sigma', \alpha'\rangle$ all the other edges do not have token. This means that cs-safeness is not affected. Therefore, the overall term $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{(\mathsf{e}^{\mathsf{v}}, \mathsf{exp}_1), (\mathsf{e}^{\mathsf{vi}}, \mathsf{exp}_2)\}), \sigma', \alpha'\rangle$ is cs-safe.

  – Node $P_1 \parallel P_2$ evolves without affecting the split and join gateways. In this case it easily follows by inductive hypothesis.

  – Node $P_1 \parallel P_2$ evolves and affects the xor join and xor split gateways. In this case one can reason like in the first case, by relying on inductive hypothesis.

  – $\langle \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{(\mathsf{e}^{\mathsf{v}}, \mathsf{exp}_1), (\mathsf{e}^{\mathsf{vi}}, \mathsf{exp}_2)\}), \sigma, \alpha\rangle$ evolves by means of rule *P-XorSplit$_1$* (the case it evolves by means of rule *P-XorSplit$_2$* is similar). It can be exploited the fact that this is a core reachable well-structured configuration to prove that the term is marked as $\sigma(\mathsf{e}^{\mathsf{iv}}) = 1$. Hence, it evolves in a cs-safe term; in fact let us assume that it evolves in this way $\langle \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{(\mathsf{e}^{\mathsf{v}}, \mathsf{exp}_1), (\mathsf{e}^{\mathsf{vi}}, \mathsf{exp}_2)\}), \sigma, \alpha\rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}})$. Hence, $edgesEl(\mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{(\mathsf{e}^{\mathsf{v}}, \mathsf{exp}_1), (\mathsf{e}^{\mathsf{vi}}, \mathsf{exp}_2)\})) = \{\mathsf{e}^{\mathsf{iv}}, \mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\}$ and $\sigma'(\mathsf{e}^{\mathsf{iv}}) = 0, \sigma'(\mathsf{e}^{\mathsf{v}}) = 1, \sigma'(\mathsf{e}^{\mathsf{vi}}) = 0$, that is $\forall \mathsf{e} \in edgesEl(\mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{(\mathsf{e}^{\mathsf{v}}, \mathsf{exp}_1), (\mathsf{e}^{\mathsf{vi}}, \mathsf{exp}_2)\}))$. $\sigma'(\mathsf{e}) \leqslant 1$. By hypothesis

$\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{(\mathsf{e}^{\mathsf{v}}, \mathsf{exp}_1), (\mathsf{e}^{\mathsf{vi}}, \mathsf{exp}_2)\}), \sigma, \alpha\rangle$ is cs-safe, i.e. if there is a token on e^{v} in the state $\langle \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{(\mathsf{e}^{\mathsf{v}}, \mathsf{exp}_1), (\mathsf{e}^{\mathsf{vi}}, \mathsf{exp}_2)\}), \sigma', \alpha'\rangle$ all the other edges do not have token. This means that cs-safeness is not affected. Therefore, the overall term $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{(\mathsf{e}^{\mathsf{v}}, \mathsf{exp}_1), (\mathsf{e}^{\mathsf{vi}}, \mathsf{exp}_2)\}), \sigma', \alpha'\rangle$ is cs-safe.

- Be $\langle P, \sigma, \alpha\rangle = \langle P_1 \parallel P_2, \sigma, \alpha\rangle$. The relevant case for cs-safeness is when P evolves by applying *P-Int₁*. It results that $\langle P_1 \parallel P_2, \sigma, \alpha\rangle \xrightarrow{\ell} \langle \sigma', \alpha'\rangle$ with $\langle P_1, \sigma, \alpha\rangle \xrightarrow{\ell} \langle \sigma', \alpha'\rangle$. By definition of $edgesEl(\cdot)$ function it follows that $edgesEl(P) = edgesEl(P_1) \cup edgesEl(P_2)$. By inductive hypothesis $\forall \mathsf{e} \in edgesEl(P_1)\ .\ \sigma(\mathsf{e}) \leqslant 1$ which is cs-safe. Since P_2 is well structured and cs-safe, then also $\langle P_2, \sigma', \alpha'\rangle$ is cs-safe, which permits to conclude.

□

Lemma 10. *Let $isWS(P)$, and let $\langle P, \sigma, \alpha\rangle$ be a process configuration reachable and cs-safe, if $\langle P, \sigma, \alpha\rangle \xrightarrow{\ell} \sigma', \alpha'$ then $\langle P, \sigma', \alpha'\rangle$ is cs-safe.*

Proof. According to Definition 27, P can have 6 different forms. The proof is given by case analysis on the parallel component of $\langle P, \sigma, \alpha\rangle$ that causes the transition $\langle P, \sigma, \alpha\rangle \xrightarrow{\ell} \sigma', \alpha'$.
It is considered now the case P= $\mathsf{start}(\mathsf{e}, \mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'', \mathsf{e}''')$.

- $\mathsf{start}(\mathsf{e}, \mathsf{e}')$ evolves by means of the rule *P-Start*. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$, hence, by cs-safeness, $\sigma(\mathsf{e}) = 1$. It can be exploited the fact that this is a reachable well-structured configuration to prove that $\sigma(\mathsf{e}') = 0$. The rule produces the following transition $\langle \mathsf{start}(\mathsf{e}, \mathsf{e}'), \sigma, \alpha\rangle \xrightarrow{\epsilon} \langle \sigma'_1, \alpha'_1\rangle$ with $\sigma'_1 = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$ where $\sigma'_1(\mathsf{e}) = 0$ and $\sigma'_1(\mathsf{e}') = 1$. Now, $\langle P, \sigma'_1, \alpha'_1\rangle = \langle \mathsf{start}(\mathsf{e}, \mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'', \mathsf{e}'''), \sigma'_1, \alpha'_1\rangle$ can evolve only through the application of *P-Int₁* producing $\langle P, \sigma', \alpha'\rangle$ with $\sigma'(in(P')) = 1$.

  By hypothesis $\langle P, \sigma, \alpha\rangle$ 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, $\langle P, \sigma', \alpha'\rangle$ is cs-safe.

- $\mathsf{end}(\mathsf{e}'', \mathsf{e}''')$ evolves by means of the rule *P-End*. It can be exploited 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 $\langle \mathsf{end}(\mathsf{e}'', \mathsf{e}'''), \sigma, \alpha\rangle \xrightarrow{\epsilon} \langle \sigma'_1, \alpha'_1\rangle$ with $\sigma'_1 = inc(dec(\sigma, \mathsf{e}''), \mathsf{e}''')$. Now, $\langle P, \sigma'_1, \alpha'_1\rangle$ can only evolve by applying *P-Int₁* producing $\langle P, \sigma', \alpha'\rangle$. By hypothesis $\langle P, \sigma, \alpha\rangle$ is cs-safe, then $\sigma(\mathsf{e}'') \leqslant 1$, $\sigma(\mathsf{e}''') \leqslant 1$ and P' is cs-safe. Reasoning as previously it can be concluded that $\langle P, \sigma', \alpha'\rangle$ is cs-safe.

- P' moves, that is $\langle P', \sigma, \alpha\rangle \xrightarrow{\ell} \langle \sigma', \alpha'\rangle$. By Lemma 2 $\langle P', \sigma', \alpha'\rangle$ is safe, thus $\forall \mathsf{e} \in edgesEl(P')\ .\ \sigma'(\mathsf{e}) \leqslant 1$. By hypothesis, P is cs-safe, hence

$edgesEl(\mathsf{start}(\mathsf{e},\mathsf{e}')) = \{\mathsf{e}'\}$ is such that $\sigma'(\mathsf{e}') \leqslant 1$ and $edgesEl(\mathsf{end}(\mathsf{e}'',\mathsf{e}''')) = \{\mathsf{e}''\}$ is such that $\sigma'(\mathsf{e}'') \leqslant 1$. Thus, it can be concluded that $\langle P, \sigma', \alpha' \rangle$ is safe.

It is considered now 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 it can be applied is *P-Terminate*. It can be exploited 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 $\langle \mathsf{terminate}(\mathsf{e}''), \sigma, \alpha \rangle \xrightarrow{kill} \langle \sigma'_1, , \alpha'_1 \rangle$ with $\sigma'_1 = dec(\sigma, \mathsf{e}'')$. Now, $\langle P, \sigma'_1, \alpha'_1 \rangle$ can only evolve by applying *P-Kill$_1$* producing $\langle P, \sigma', \alpha' \rangle$ where σ' is completed unmarked; therefore it is cs-safe.
- P' moves: similar to the previous case.

It is considered now the case P= $\mathsf{start}(\mathsf{e},\mathsf{e}') \parallel P' \parallel \mathsf{endSnd}(\mathsf{e}'', \mathsf{m} : \mathsf{e}\tilde{\mathsf{x}}\mathsf{p}, \mathsf{e}''')$.

- The start event evolves: like the previous case.
- The end message event evolves: the only transition one can apply is *P-EndSnd*. It can be exploited 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 it follows that $\langle \mathsf{endSnd}(\mathsf{e}'', \mathsf{m} : \mathsf{e}\tilde{\mathsf{x}}\mathsf{p}, \mathsf{e}'''), \sigma, \alpha \rangle \xrightarrow{!\mathsf{m} : \tilde{\mathsf{v}}} \langle \sigma'_1, \alpha'_1 \rangle$ with $\sigma'_1 = inc(dec(\sigma, \mathsf{e}''), \mathsf{e}''')$ Now, $\langle P, \sigma'_1, \alpha'_1 \rangle$ can only evolve by applying *P-Int$_1$* producing $\langle P, \sigma', \alpha' \rangle$. By hypothesis $\langle P, \sigma, \alpha \rangle$ is cs-safe, then $\sigma(\mathsf{e}'') \leqslant 1$, $\sigma(\mathsf{e}''') \leqslant 1$ and P' is cs-safe. Reasoning as previously it can be concluded that $\langle P, \sigma' \rangle$ is cs-safe.
- P' moves: similar to the previous cases.

It is considered now the case P= $\mathsf{startRcv}(\mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'', \mathsf{e}''')$.

- $\mathsf{startRcv}(\mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}')$ evolves by means of the rule *P-StartRcv*. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$, $eval(\mathsf{e}\tilde{\mathsf{x}}\mathsf{p}, \alpha, \tilde{\mathsf{v}})$, hence, by cs-safeness, $\sigma(\mathsf{e}) = 1$. It can be exploited the fact that this is a reachable well-structured configuration to prove that $\sigma(\mathsf{e}') = 0$. The rule produces the following transition $\langle \mathsf{startRcv}(\mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}'), \sigma, \alpha \rangle \xrightarrow{?\mathsf{m}} \langle \sigma'_1, \alpha'_1 \rangle$ with $\sigma'_1 = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$ where $\sigma'_1(\mathsf{e}) = 0$ and $\sigma'_1(\mathsf{e}') = 1$. Now, $\langle P, \sigma'_1, \alpha'_1 \rangle = \mathsf{startRcv}(\mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'', \mathsf{e}'''), \sigma'_1, \alpha'_1 \rangle$ can evolve only through the application of *P-Int$_1$* producing $\langle P, \sigma', \alpha' \rangle$ with $\sigma'(in(P')) = 1$. By hypothesis $\langle P, \sigma, \alpha \rangle$ 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 $\langle P', \sigma\rangle \xrightarrow{\ell} \langle \sigma', \alpha'\rangle$. By Lemma 2 $\langle P', \sigma', \alpha'\rangle$ is safe, thus $\forall \mathsf{e} \in edgesEl(P')\ .\ \sigma'(\mathsf{e}) \leqslant 1$. By hypothesis P is cs-safe, thus $edgesEl(\mathsf{startRcv}(\mathsf{m}:\tilde{\mathsf{t}}, \mathsf{e}'))= \{\mathsf{e}'\}$ is such that $\sigma'(\mathsf{e}') \leqslant 1$ and $edgesEl(\mathsf{end}(\mathsf{e}'', \mathsf{e}''')) = \{\mathsf{e}''\}$ is such that $\sigma'(\mathsf{e}'') \leqslant 1$. It can be concluded that $\langle P, \sigma', \alpha'\rangle$ is safe.

It is considered now the case P= $\mathsf{startRcv}(\mathsf{m}:\tilde{\mathsf{t}}, \mathsf{e}') \parallel P' \parallel \mathsf{terminate}(\mathsf{e}'')$.

- $\mathsf{startRcv}(\mathsf{m}:\tilde{\mathsf{t}}, \mathsf{e}')$ evolves by means of the rule *P-StartRcv*: like in the previous case.

- The end terminate event evolves: the only transition it can be applied is *P-Terminate*: like in the case P= $\mathsf{startRcv}(\mathsf{m}:\tilde{\mathsf{t}}, \mathsf{e}') \parallel P' \parallel \mathsf{terminate}(\mathsf{e}'')$.

- P' moves, that is $\langle P', \sigma\rangle \xrightarrow{\ell} \sigma'$. 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 thus $edgesEl(\mathsf{startRcv}(\mathsf{m}:\tilde{\mathsf{t}}, \mathsf{e}'))= \{\mathsf{e}'\}$ is such that $\sigma'(\mathsf{e}') \leqslant 1$ and $edgesEl(\mathsf{terminate}(\mathsf{e}'')) = \{\mathsf{e}''\}$ is such that $\sigma'(\mathsf{e}'') \leqslant 1$. It follows that $\langle P, \sigma'\rangle$ is safe.

It is considered now the case P= $\mathsf{startRcv}(\mathsf{m}:\tilde{\mathsf{t}}, \mathsf{e}') \parallel P' \parallel \mathsf{endSnd}(\mathsf{e}'', \mathsf{m}:\tilde{\mathsf{exp}}, \mathsf{e}''')$.

- $\mathsf{startRcv}(\mathsf{m}:\tilde{\mathsf{t}}, \mathsf{e}')$ evolves by means of the rule *P-StartRcv*: like in the previous case.

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

- P' moves, that is $\langle P', \sigma, \alpha\rangle \xrightarrow{\ell} \langle \sigma', \alpha'\rangle$. By Lemma 2 $\langle P', \sigma', \alpha'\rangle$ is safe, thus $\forall \mathsf{e} \in edgesEl(P')\ .\ \sigma'(\mathsf{e}) \leqslant 1$. By hypothesis, P is cs-safe thus $edgesEl(\mathsf{startRcv}(\mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}')) = \{\mathsf{e}'\}$ is such that $\sigma'(\mathsf{e}') \leqslant 1$ and $edgesEl(\mathsf{endSnd}(\mathsf{e}'', \mathsf{m} : \tilde{\mathsf{exp}}, \mathsf{e}''')) = \{\mathsf{e}''\}$ is such that $\sigma'(\mathsf{e}'') \leqslant 1$. It follows that $\langle P, \sigma', \alpha'\rangle$ is safe.

□

Theorem 16. *Let P be a process, if P is well-structured then P is safe.*

Proof. It has to be shown that if $\langle P, \sigma, \alpha\rangle \rightarrow^* \langle \sigma', \alpha'\rangle$ then $\langle P, \sigma', \alpha'\rangle$ is cs-safe. It proceeds by induction on the length n of the sequence of transitions from $\langle P, \sigma, \alpha\rangle$ to $\langle P, \sigma', \alpha'\rangle$.
Base Case ($n = 0$): In this case $\sigma = \sigma'$, $\alpha = \alpha'$ then $isInit(P, \sigma', \alpha')$ is satisfied. By Lemma 1 it can be concluded $\langle P, \sigma', \alpha'\rangle$ is cs-safe.
Inductive Case: In this case $\langle P, \sigma, \alpha\rangle \rightarrow^* \langle P, \sigma'', \alpha''\rangle \xrightarrow{\ell} \langle P, \sigma', \alpha'\rangle$ for some process $\langle P, \sigma'', \alpha''\rangle$. By induction, $\langle P, \sigma'', \alpha''\rangle$ is cs-safe. By applying Lemma 3 to $\langle P, \sigma'', \alpha''\rangle \xrightarrow{\ell} \langle P, \sigma', \alpha'\rangle$, it can be concluded that $\langle P, \sigma', \alpha'\rangle$ is cs-safe. □

Theorem 17. *Let C be a collaboration, if C is well-structured then C is safe.*

Proof. By contradiction, let C be well-structured and C unsafe. By Definition 31, given ι and δ such that $isInit(C, \iota, \delta)$ there exists a collaboration configuration $\langle C, \iota', \delta' \rangle$ such that $\langle C, \iota, \delta \rangle \rightarrow^* \langle C, \iota', \delta' \rangle$ and $\exists \mathsf{pool}(\mathsf{p}, P)$ in C, given $\iota'(\mathsf{p}) = \{\langle \sigma', \alpha' \rangle\}$, such that $\langle P, \sigma', \alpha' \rangle$ is not cs-safe (the case of multi-instance pools is equal). From hypothesis $isInit(C, \iota, \delta)$, it follows $isInit(P, \sigma, \alpha)$, with $\iota(\mathsf{p}) = \{\langle \sigma, \alpha \rangle\}$. Thus, also $\langle P, \sigma', \alpha' \rangle$ is reachable. From hypothesis C is well-structured, it follows that P is WS. Therefore, by Theorem 1, P is safe. By Definition 30, $\langle P, \sigma', \alpha' \rangle$ is cs-safe, which is a contradiction. □

Lemma 11. *Let $isWSCore(P)$ and let $\langle P, \sigma, \alpha \rangle$ be core reachable, then there exist σ', α' such that $\langle P, \sigma, \alpha \rangle \rightarrow^* \langle \sigma', \alpha' \rangle$ and $isCompleteEl(P, \sigma', \alpha')$.*

Proof. 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.

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

- $P = \mathsf{taskRcv}(\mathsf{e}, \mathsf{exp}, A, \mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}')$. The only rule that can be applied is *P-TaskRcv*. In order to apply it there must be $\sigma(\mathsf{e}) > 0$, $eval(\mathsf{exp}, \alpha, true)$, $eval(\tilde{\mathsf{t}}, \alpha, \tilde{\mathsf{et}})$. Since $isWSCore(P)$, $\langle P, \sigma, \alpha \rangle$ is safe, hence $\sigma(\mathsf{e}) = 1$. Since the process configuration is core reachable it follows $\sigma(\mathsf{e}') = 0$. The application of the rule produces $\langle \mathsf{taskRcv}(\mathsf{e}, \mathsf{exp}, A, \mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}')\sigma, \alpha \rangle \xrightarrow{?\mathsf{m} : \tilde{\mathsf{et}}, A} \langle \sigma', \alpha' \rangle$ with $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$. Thus, it follows $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$, which permits to conclude.

- $P = \mathsf{taskSnd}(\mathsf{e}, \mathsf{exp}', A, \mathsf{m} : \tilde{\mathsf{exp}}, \mathsf{e}')$,. The only rule that can be applied is *P-TaskSnd*. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$, $eval(\mathsf{exp}', \alpha, true)$, $upd(\alpha, A, \alpha')$, $eval(\tilde{\mathsf{exp}}, \alpha, \tilde{\mathsf{v}})$. Since $isWSCore(P)$, $\langle P, \sigma, \alpha \rangle$ is safe, hence $\sigma(\mathsf{e}) = 1$. Since the process configuration is core reachable it follows $\sigma(\mathsf{e}') = 0$. The application of the rule produces $\langle \mathsf{taskSnd}(\mathsf{e}, \mathsf{exp}', A, \mathsf{m} : \tilde{\mathsf{exp}}, \mathsf{e}'), \sigma, \alpha \rangle \xrightarrow{!\mathsf{m} : \tilde{\mathsf{v}}} \langle \sigma', \alpha' \rangle$ with $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$. Thus, it follows $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$, which permits to conclude.

- $P = \mathsf{interRcv}(\mathsf{e}, \mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}')$. The only rule that can be applied is *P-InterRcv*. In order to apply it there must be $\sigma(\mathsf{e}) > 0$, $eval(\tilde{\mathsf{t}}, \alpha, \tilde{\mathsf{et}})$. Since $isWSCore(P)$, $\langle P, \sigma, \alpha \rangle$ is safe, hence $\sigma(\mathsf{e}) = 1$. Since the process configuration is core reachable it follows $\sigma(\mathsf{e}') = 0$. By applying the rule produces $\langle \mathsf{interRcv}(\mathsf{e}, \mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}'), \sigma, \alpha \rangle \xrightarrow{?\mathsf{m} : \tilde{\mathsf{et}}, \epsilon} \langle \sigma', \alpha' \rangle$ with $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$. Thus, it follows $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$, which permits to conclude.

- $P = \mathsf{interSnd}(\mathsf{e}, \mathsf{m} : \tilde{\mathsf{exp}}, \mathsf{e}')$. The only rule that can be applied is *P-InterSnd*. In order to apply it there must be $\sigma(\mathsf{e}) > 0$, $eval(\tilde{\mathsf{exp}}, \alpha, \tilde{\mathsf{v}})$.

Since $isWSCore(P)$, $\langle P, \sigma, \alpha\rangle$ is safe, hence $\sigma(\mathsf{e}) = 1$. Since the process configuration is core reachable it follows $\sigma(\mathsf{e}') = 0$. By applying the rule $\langle \mathsf{interSnd}(\mathsf{e}, \mathsf{m}:\tilde{\mathsf{exp}}, \mathsf{e}'), \sigma, \alpha\rangle \xrightarrow{!\mathsf{m}:\tilde{\mathsf{v}}} \langle \sigma', \alpha'\rangle$ with $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$. Thus, it follows $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$, which permits to conclude.

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

Inductive cases:

- $P = \langle \mathsf{andSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{andJoin}(E', \mathsf{e}''), \sigma\rangle$. There are the following possibilities:
  - $\langle \mathsf{andSplit}(\mathsf{e}, E), \sigma\rangle$ evolves by means of rule *P-AndSplit*. It can be exploited the fact that this is a core reachable well-structured configuration to prove that $\sigma(\mathsf{e}) = 1$ and $\forall \mathsf{e}'' \in E \,.\sigma(\mathsf{e}'') = 0$. Thus, $\langle \mathsf{andSplit}(\mathsf{e}, E), \sigma\rangle \xrightarrow{\epsilon} \langle \sigma'_1, \alpha'_1\rangle$ with $\sigma'_1 = inc(dec(\sigma, \mathsf{e}), E)$. Now, P can evolve only through the application of *P-Int₁* producing $\langle P, \sigma'_2, \alpha'_2\rangle$ with $\sigma'_2(in(P_1)) = \ldots = \sigma'_2(in(P_n)) = 1$. By inductive hypothesis there exists a state σ'_3 such that $isCompleteEl(P_1 \parallel \ldots \parallel P_n, \sigma'_3, \alpha'_3)$. Now, P can only evolve by applying rule *P-Int₁*, producing $\langle P, \sigma'_4, \alpha'_4\rangle$ where $\forall \mathsf{e}''' \in E' \,.\, \sigma'_4(\mathsf{e}''') = 1$. Now, $\langle \mathsf{andJoin}(E', \mathsf{e}''), \sigma'_4, \alpha'_4\rangle$ can evolve by means of rule *P-AndJoin*. The application of the rule produces $\langle \mathsf{andJoin}(E', \mathsf{e}''), \sigma'_4, \alpha'_4\rangle \xrightarrow{\epsilon} \langle \sigma', \alpha'\rangle$ with $\sigma' = inc(dec(\sigma'_4, E'), \mathsf{e}'')$, i.e. $\sigma'(\mathsf{e}'') = 1$ and $\forall \mathsf{e}''' \in E' \,.\, \sigma'(\mathsf{e}''') = 0$. This permits to conclude.
  - $P_1 \parallel \ldots \parallel P_n$ evolves without affecting the split and join gateways. In this case it easily follows by inductive hypothesis.
  - $P_1 \parallel \ldots \parallel P_n$ evolves and affects the split and/or join gateways. In this case one can reason like in the first case.

- $P = \langle \mathsf{xorSplit}(\mathsf{e}, G) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E, \mathsf{e}''), \sigma, \alpha\rangle$. There are the following possibilities:
  - $\langle \mathsf{xorSplit}(\mathsf{e}, G), \sigma, \alpha\rangle$ evolves by means of rule *P-XorSplit₁* (the case it evolves by means of rule *P-XorSplit₂* is similar). It can be exploited the fact that this is a core reachable well-structured configuration to prove that $\sigma(\mathsf{e}) = 1$ and $\forall(\mathsf{e}', \mathsf{exp}) \in G \,.\sigma(\mathsf{e}') = 0$. Thus, $\mathsf{xorSplit}(\mathsf{e}, \{(\mathsf{e}''', \mathsf{exp})\} \cup G), \sigma, \alpha\rangle \xrightarrow{\epsilon} \langle \sigma'_1, \alpha'_1\rangle$, with $\sigma'_1 = inc(dec(\sigma, \mathsf{e}), \mathsf{e}''')$. Now, P can evolve only through the application of *P-Int₁* producing $\langle P, \sigma'_2, \alpha'_2\rangle$ with $\sigma'_2(in(P_1)) = \ldots = \sigma'_2(in(P_n)) = 1$. By inductive hypothesis there exists a state σ'_3 such that $isCompleteEl(P_1 \parallel \ldots \parallel P_n, \sigma'_3, \alpha'_3)$. Now, P can only evolve by applying rule *P-Int₁*, producing $\langle P, \sigma'_4, \alpha'_4\rangle$ where $\exists \mathsf{e}''' \in E \,.\, \sigma'_4(\mathsf{e}''') = 1$,

let us say $\sigma'_4(\mathsf{e}^{\mathsf{iv}}) = 1$. Now, $\langle \mathsf{xorJoin}(\{\mathsf{e}^{\mathsf{iv}}\} \cup E, \mathsf{e}''), \sigma'_4, \alpha'_4\rangle$ can evolve by means of rule *P-XorJoin*. The application of the rule produces $\langle \mathsf{xorJoin}(\{\mathsf{e}^{\mathsf{iv}}\} \cup E, \mathsf{e}''), \sigma'_4, \alpha'_4\rangle \xrightarrow{\epsilon} \langle \sigma', \alpha'\rangle$ 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 to conclude.

- $P_1 \parallel \ldots \parallel P_n$ evolves without affecting the split and join gateways. In this case if easily follows by inductive hypothesis.
- $P_1 \parallel \ldots \parallel P_n$ evolves and affects the split and/or join gateways. In this case one can reason like in the first case.

- $P = \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j : \tilde{\mathsf{t}}_\mathsf{j}, \mathsf{e}'_j) | j \in [1..n]\}) \parallel P_1 \parallel \ldots \mid P_n \parallel \mathsf{xorJoin}(E, \mathsf{e}'')$. There are the following possibilities:
  - $\langle \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j : \tilde{\mathsf{t}}_\mathsf{j}, \mathsf{e}'_j) | j \in [1..n]\}), \sigma, \alpha\rangle$ evolves by means of rule *P-EventG*. It can be exploited 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, $\langle \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j : \tilde{\mathsf{t}}_\mathsf{j}, \mathsf{e}'_j) | j \in [1..n]\}), \sigma, \alpha\rangle \xrightarrow{?\mathsf{m}_j \,:\, \tilde{\mathsf{et}}_\mathsf{j}, \epsilon} \langle \sigma'_1, \alpha'_1\rangle$ with $\sigma'_1 = inc(dec(\sigma, \mathsf{e}), \mathsf{e}'_j)$. Now, P can evolve only through the application of *P-Int1* producing $\langle P, \sigma'_2, \alpha'_2\rangle$ with $\sigma'_2(in(P_1)) = \ldots = \sigma'_2(in(P_n)) = 1$. By inductive hypothesis there exists a state σ'_3 such that $isCompleteEl(P_1 \parallel \ldots \parallel P_n, \sigma'_3, \alpha'_3)$. Now, P can only evolve by applying rule *P-Int1*, producing $\langle P, \sigma'_4, \alpha'_4\rangle$ 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-XorJoin*. The application of the rule produces $\langle \mathsf{xorJoin}(\{\mathsf{e}^{\mathsf{iv}}\} \cup E, \mathsf{e}''), \sigma'_4\rangle \xrightarrow{\epsilon} \langle \sigma'\alpha'\rangle$ 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 to conclude.
  - $P_1 \parallel \ldots \parallel P_n$ evolves without affecting the split and join gateways. In this case it easily follows by inductive hypothesis.
  - $P_1 \parallel \ldots \parallel P_n$ evolves and affects the split and/or join gateways. In this case one can reason like in the first case.

- $\mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{(\mathsf{e}^{\mathsf{v}}, \mathsf{exp}_1), (\mathsf{e}^{\mathsf{vi}}, \mathsf{exp}_2)\})$ 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:
  - $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}'), \sigma, \alpha\rangle$ evolves by means of rule *P-XorJoin*. It can be exploited 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 $\sigma(\mathsf{e}''') = 1$; assuming that the marking is $\sigma(\mathsf{e}''') = 1$ (since the other case is similar), then $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}'), \sigma, \alpha\rangle \xrightarrow{\epsilon} \langle \sigma'_1, \alpha'_1\rangle$ with $\sigma'_1 = inc(dec(\sigma, \mathsf{e}'''), \mathsf{e}')$. Now, P can evolve only through the application of *P-Int1* producing $\langle P, \sigma'_2, \alpha'_2\rangle$ with $\sigma'_2(in(P_1)) = \sigma'_2(in(P_2)) = 1$. By inductive hypothesis there exists a state σ'_3 such that $isCompleteEl(P_1 \parallel P_2, \sigma'_3, \alpha'_3)$. Now, P can only evolve by applying rule *P-Int1*, producing $\langle P, \sigma'_4, \alpha'_4\rangle$ with, $\sigma'_4(\mathsf{e}^{\mathsf{iv}}) = 1$. Now, $\langle \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{(\mathsf{e}^{\mathsf{v}}, \mathsf{exp}_1), (\mathsf{e}^{\mathsf{vi}}, \mathsf{exp}_2)\}), \sigma'_4, \alpha'_4\rangle$ can evolve by means of rule *P-XorSplit1*. The application of

the rule produces $\langle \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{(\mathsf{e}^{\mathsf{v}}, \mathsf{exp}_1), (\mathsf{e}^{\mathsf{vi}}, \mathsf{exp}_2)\}), \sigma', \alpha' \rangle \xrightarrow{\epsilon} \langle \sigma', \alpha' \rangle$ 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 to conclude.

- $P_1 \parallel P_2$ evolves without affecting the split and join gateways. In this case it easily follows by inductive hypothesis.
- $P_1 \parallel P_2$ evolves and affects the split and/or join gateways. In this case onecan reason like in the first case.

- Be $\langle P, \sigma, \alpha \rangle = \langle P_1 \parallel P_2, \sigma, \alpha \rangle$, 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*. Thus, $\langle P_1 \parallel P_2, \sigma, \alpha \rangle \xrightarrow{\ell} \sigma'_1$ with $\langle P_1, \sigma, \alpha \rangle \xrightarrow{\ell} \langle \sigma'_1, \alpha'_1 \rangle$. By inductive hypothesis there exist σ', α' such that $isCompleteEl(P_1, \sigma', \alpha')$. By hypothesis $out(P_1) = in(P_2)$ thus, $isCompleteEl(\mathsf{getOutEl}(\mathsf{e}, P_1 \parallel P_2, \sigma', \alpha')) = isCompleteEl(\mathsf{getOutEl}(\mathsf{e}, P_1), \sigma', \alpha')$, that holds by inductive hypothesis. By hypothesis P_2 is well structured and core reachable, then it follows that $edges(P_2) \backslash out(P_2)) : \sigma'(\mathsf{e}) = 0$ By definition of $isCompleteEl(P_1, \| P_2, \sigma', \alpha')$ one can conclude.

□

Theorem 18. *Let $\langle P, \sigma, \alpha \rangle$ be a WS process configuration, then $\langle P, \sigma, \alpha \rangle$ is sound.*

Proof. According to Definition 27, P can have 6 different forms. It is now considered the case P= $\mathsf{start}(\mathsf{e}, \mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'', \mathsf{e}''')$.

Assuming that $isInit(P, \sigma, \alpha)$, it follows 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 it can be applied only the rule *P-Start*. The rule produces the following transition, $\langle \mathsf{start}(\mathsf{e}, \mathsf{e}'), \sigma, \alpha \rangle \xrightarrow{\epsilon} \langle \sigma', \alpha' \rangle$ with $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$ where $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$. Now $\langle P, \sigma', \alpha' \rangle$ can evolve through the application of rule *P-Int₁* producing $\langle P, \sigma'_1, \alpha'_1 \rangle$, with $\sigma'_1(in(P')) = 1$. Now P' moves. By hypothesis $isWSCore(P')$, thus by Lemma 4 there exists a process configuration $\langle P', \sigma'_2, \alpha'_2 \rangle$ such that $\langle P', \sigma'_1, \alpha'_1 \rangle \rightarrow^* \langle \sigma'_2, \alpha'_2 \rangle$ and $isCompleteEl(P', \sigma'_2, \alpha'_2)$. The process can now evolve thorough rule *P-Int₁*. By hypothesis the process is WS, thus, after the application of the rule it follows that $\langle \mathsf{start}(\mathsf{e}, \mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'', \mathsf{e}'''), \sigma'_3, \alpha'_3 \rangle$, where $\sigma'_3(\mathsf{e}'') = 1$ and $\forall \mathsf{e}^{\mathsf{v}} \in edges(P')\ .\ \sigma'_3(\mathsf{e}^{\mathsf{v}}) = 0$. Now it can be applied the rule *P-End* that decrements the token in e'' and produces a token in e''', which permits to conclude. □

Theorem 19. *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. It can be considered the same collaboration used in the proof of Theorem 4 (Appendix C.1). □

Theorem 20. *Let C be a collaboration, $isWS(C)$ does not imply C is message-relaxed sound.*

Proof. Let C be a WS collaboration,. Supposing 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. One can consider again the collaboration in Figure C.1, as in the proof of Theorem 5. □

Theorem 21. *Let P be a collaboration, P is unsafe does not imply P is unsound.*

Proof. Let P be a unsafe collaboration. Supposing that P is unsound, then, it is sufficient to show a counter example, i.e. a unsafe collaboration that is sound. Again, it can be considered the example used in the proof of Theorem 6. □

Theorem 22. *Let C be a collaboration, C is unsafe does not imply C is unsound.*

Proof. Let C be a unsafe collaboratio. Supposing that C is unsound, then it is sufficient to show a counter example, i.e. a unsafe collaboration that is sound. One can consider againthe collaboration in Figure C.3, used to prove Theorem 7. □

Theorem 23. *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 $\langle C, \iota', \delta' \rangle$ such that $\langle C, \iota, \delta \rangle \rightarrow^* \langle \iota', \delta' \rangle$, with $\mathsf{pool}(\mathsf{p}, P)$ in C, for $\iota'(\mathsf{p}) = \{\langle \sigma', \alpha' \rangle\}$ and $\langle P, \sigma', \alpha' \rangle$ not cs-safe (the case of multi-instance pools is equal). By hypothesis all processes of C are safe, hence it is safe the process, say P, of organisation p. As $\langle C, \iota', \delta' \rangle$ results from the evolution of $\langle C, \iota, \delta \rangle$, the process $\langle P, \sigma', \alpha' \rangle$ must derive from $\langle P, \sigma, \alpha \rangle$ with $\iota(\mathsf{p}) = \{\langle \sigma, \alpha \rangle\}$ as well, that is $\langle P, \sigma, \alpha \rangle \rightarrow^* \langle \sigma', \alpha' \rangle$. By safeness of P, it follows that $\langle P, \sigma', \alpha' \rangle$ is cs-safe, which is a contradiction. □

Theorem 24. *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 $isInit(C, \iota, \delta)$, for all ι' and δ' such that $\langle C, \iota, \delta \rangle \rightarrow^* \langle \iota', \delta' \rangle$ it follows that there exist ι'' and δ'' such that $\langle C, \iota', \delta' \rangle \rightarrow^* \langle \iota'', \delta'' \rangle$, and

- $\forall\, \mathsf{pool}(\mathsf{p}, P)$ in C, given $\iota''(\mathsf{p}) = \{\langle \sigma, \alpha \rangle\}$, $\langle P, \sigma, \alpha \rangle$ is cs-sound
- $\forall\, \mathsf{miPool}(\mathsf{p}, P)$ in C, given $\{\langle \sigma, \alpha \rangle\} \in \iota''(\mathsf{p})$, $\langle P, \sigma, \alpha \rangle$ is cs-sound

and $\forall\, \mathsf{m} \in \mathbb{M}\ .\ \delta''(\mathsf{m}) = 0$. Since P_1 is unsound, it follows that, given σ_1', such that $isInit(P_1, \sigma_1', \alpha_1')$, for all σ_2' , α_2' such that $\langle P_1, \sigma, \alpha \rangle \rightarrow^* \langle \sigma_2', \alpha_2' \rangle$ it follows that does not exist σ_3', α_3' such that $\langle P_1, \sigma_2' \rangle \rightarrow^* \langle \sigma_3', \alpha_3' \rangle$, and $\langle P_1, \sigma_3', \alpha_3' \rangle$ is cs-sound. Choosing $\langle C, \iota', \delta' \rangle$ such that $\mathsf{pool}(\mathsf{p}, P_1)$ in C,with $\iota''(\mathsf{p}) = \{\langle \sigma_3', \alpha_3' \rangle\}$ by unsoundness of P_1 it follows that there exists a process in C that is not cs-sound, which is a contradiction. □