Safeness and Soundness Checking in BPMN 2.0 Collaborations
– APPENDIX –

Flavio Corradini, Andrea Morichetta, Andrea Polini,
Barbara Re, Lorenzo Rossi, and Francesco Tiezzi

School of Science and Technology, University of Camerino, Italy
{*name.surname*}@unicam.it

Abstract. This online appendix reports the formal definitions and the theorem proofs regarding the framework presented in the companion manuscript.

1 Formal Framework

1.1 Syntax of BPMN Collaborations

To enable the formal treatment of collaborations' semantics, we defined a BNF syntax of their model structure (Fig. 1). In the proposed grammar, the non-terminal symbol C represents a *Collaboration Structure*, while the terminal symbols, denoted by the $\mathsf{sans\ serif}$ font, are the considered BPMN elements, i.e. events, tasks, sub-processes and gateways.

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

In the following $\mathsf{e} \in \mathbb{E}$ denotes a *sequence edge*, while $E \in 2^{\mathbb{E}}$ a set of edges; we require $|E| > 1$ when E is used in joining and splitting gateways. Similarly, we require that an event-based gateway should contain at least two message events, i.e. $k > 1$ in each $\mathsf{eventBased}$ term. For the convenience of the reader, we refer with e_i the edge incoming in an element and with e_o the edge outgoing from an element. In the edge set $\mathbb{E}$ we also include spurious edges for denoting the enabled status of start events and the complete status of end events, named *enabling* and *completing* edges, respectively. They are needed to guarantee multi-layer activation of sub-processes as well as to check their completion. Moreover,

$$\begin{aligned}C ::= {} & \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \ \mid\ \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}) \ \mid\ \mathsf{andSplit}(\mathsf{e}_i, E_o) \ \mid\ \mathsf{xorSplit}(\mathsf{e}_i, E_o)\\ \mid\ & \mathsf{andJoin}(E_i, \mathsf{e}_o) \ \mid\ \mathsf{xorJoin}(E_i, \mathsf{e}_o) \ \mid\ \mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_k, \mathsf{e}_{ok}))\\ \mid\ & \mathsf{task}(\mathsf{e}_i, \mathsf{e}_o) \ \mid\ \mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \ \mid\ \mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \ \mid\ \mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)\\ \mid\ & \mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \ \mid\ \mathsf{subProc}(\mathsf{e}_i, C, \mathsf{e}_o) \ \mid\ C_1 | C_2\end{aligned}$$

Fig. 1. Syntax of BPMN Collaboration Structures.

$\mathsf{m} \in \mathbb{M}$ denotes a *message edge*, enabling message exchanges between pairs of participants in the collaboration. Both, e and m denote names uniquely identifying a sequence edge and a message, respectively. The correspondence between the syntax used here 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 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}.
- $\mathsf{andSplit}(\mathsf{e}_i, E_o)$ (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.
- $\mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_k, \mathsf{e}_{ok}))$ 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.
- $\mathsf{task}(\mathsf{e}_i, \mathsf{e}_o)$ represents a task with incoming edge e_i and outgoing edge e_o; we can also observe $\mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$ - resp. $\mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$ - to consider a task receiving - resp. sending - a message m.
- $\mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$ - resp. $\mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$ - represents an intermediate receiving - resp. sending - event with an incoming edge e_i and an outgoing edge e_o that are able to receive - resp. sending - a message m.
- $\mathsf{subProc}(\mathsf{e}_i, C, \mathsf{e}_o)$ represents a sub-process element with incoming edge e_i and outgoing edge e_o. When activated, the enclosed sub-process C behaves according to the elements it consists of, including nested sub-process elements (used to describe multi-layer collaborations with a hierarchical structure).
- $C_1 | C_2$ 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. To avoid malformed structure models, we only consider structures in which for each edge labeled by e (resp. m) outgoing from an element, there exists only one corresponding edge labeled by e (resp. m) incoming into another element, and vice versa.

Since we consider collaborations with a multi-layer structure, to refer the start events of the current layer C we resort to function $start(C)$, which returns their enabling edges:

$$start(C_1 \mid C_2) = start(C_1) \cup start(C_2) \qquad start(\mathsf{start}(\mathsf{e}_{init}, \mathsf{e}_o)) = \{\mathsf{e}_{init}\}$$
$$start(C) = \emptyset \;\text{ for any element } C \neq \mathsf{start}(\mathsf{e}_{init}, \mathsf{e}_o)$$

Notably, we assume that each process in the collaboration has only one start. Thus, the above function applied on the abstract layer of the collaboration will return as many edges as the number of participants involved in the collaboration, while the application of the function in each process/sub-process returns only one edge.

We similarly define the function $end(C)$ on the structure of collaborations in order to refer to end events in the current layer:

$$end(C_1 \mid C_2) = end(C_1) \cup end(C_2) \qquad end(\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})) = \{\mathsf{e}_{cmp}\}$$
$$end(C) = \emptyset \;\text{ for any element } C \neq \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})$$

1.2 Semantics of BPMN Collaborations

The syntax presented so far permits to describe the mere structure of a collaboration. To describe its semantics we need to enrich it with a notion of execution state, defining the current marking of sequence and message edges. We call *collaboration configuration* this stateful description.

Formally, a configuration has the form $\langle C, \sigma, \delta \rangle$, where: C is a collaboration structure; σ is the first part of the execution state, storing for each sequence edge the current number of tokens marking it; and δ is the second part of the execution state, storing for each message edge the current number of message tokens marking it. Specifically, a state $\sigma : \mathbb{E} \rightarrow \mathbb{N}$ is a function mapping edges to numbers 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, $\delta : \mathbb{M} \rightarrow \mathbb{N}$ is a function mapping message edges to numbers 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 are defined in a way similar to σ's definitions. Moreover, in the *initial state* of a collaboration, the start event of each process in the abstract level must be enabled, i.e. it has a token in its enabling edge, while all other sequence edges (included the enabling edges for the activation of sub-processes in the lower layers) and messages edges must be unmarked.

Definition 1 (Initial state of collaboration). *Let C be a collaboration, the collaboration configuration $\langle C, \sigma, \delta \rangle$ is the initial one, i.e. predicate isInit$(\langle C, \sigma, \delta \rangle)$ holds, if $\forall\ \mathsf{e}_{enb} \in start(C)\ .\ \sigma(\mathsf{e}_{enb}) = 1$, $\forall\ \mathsf{e} \in \mathbb{E} \backslash start(C)\ .\ \sigma(\mathsf{e}) = 0$, and $\forall\ \mathsf{m} \in \mathbb{M}\ .\ \delta(\mathsf{m}) = 0$.*

The operational semantics is defined, as usual, by means of a *labelled transition system* (LTS). In our case, this is a triple $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ where: $\mathcal{C}$, ranged over by $\langle C, \sigma, \delta \rangle$, is a set of collaboration configurations; $\mathcal{L}$, ranged over by l, is a

$\langle \mathsf{start}(\mathsf{e}_{enb},\mathsf{e}_o),\sigma,\delta\rangle \xrightarrow{\mathsf{e}_{enb}} \langle \mathit{inc}(\mathit{dec}(\sigma,\mathsf{e}_{enb}),\mathsf{e}_o),\delta\rangle$	$\sigma(\mathsf{e}_{enb})>0$	(*C-Start*)
$\langle \mathsf{end}(\mathsf{e}_i,\mathsf{e}_{cmp}),\sigma,\delta\rangle \xrightarrow{\mathsf{e}_i} \langle \mathit{inc}(\mathit{dec}(\sigma,\mathsf{e}_i),\mathsf{e}_{cmp}),\delta\rangle$	$\sigma(\mathsf{e}_i)>0$	(*C-End*)
$\langle \mathsf{andSplit}(\mathsf{e}_i,E_o),\sigma,\delta\rangle \xrightarrow{\mathsf{e}_i} \langle \mathit{inc}(\mathit{dec}(\sigma,\mathsf{e}_i),E_o),\delta\rangle$	$\sigma(\mathsf{e}_i)>0$	(*C-AndSplit*)
$\langle \mathsf{xorSplit}(\mathsf{e}_i,\{\mathsf{e}\}\cup E_o),\sigma,\delta\rangle \xrightarrow{\mathsf{e}_i} \langle \mathit{inc}(\mathit{dec}(\sigma,\mathsf{e}_i),\mathsf{e}),\delta\rangle$	$\sigma(\mathsf{e}_i)>0$	(*C-XorSplit*)
$\langle \mathsf{andJoin}(E_i,\mathsf{e}_o),\sigma,\delta\rangle \xrightarrow{E_i} \langle \mathit{inc}(\mathit{dec}(\sigma,E_i),\mathsf{e}_o),\delta\rangle$	$\forall \mathsf{e}\in E_i \;.\; \sigma(\mathsf{e})>0$	(*C-AndJoin*)
$\langle \mathsf{xorJoin}(\{\mathsf{e}\}\cup E_i,\mathsf{e}_o),\sigma,\delta\rangle \xrightarrow{\mathsf{e}} \langle \mathit{inc}(\mathit{dec}(\sigma,\mathsf{e}),\mathsf{e}_o),\delta\rangle$	$\sigma(\mathsf{e})>0$	(*C-XorJoin*)
$\langle \mathsf{eventBased}(\mathsf{e}_i,(\mathsf{m}_1,\mathsf{e}_{o1}),\ldots,(\mathsf{m}_k,\mathsf{e}_{ok})),\sigma,\delta\rangle \xrightarrow{?\mathsf{m}_j}$ $\langle \mathit{inc}(\mathit{dec}(\sigma,\mathsf{e}_i),\mathsf{e}_{oj}),\mathit{dec}(\delta,\mathsf{m}_j)\rangle$	$\sigma(\mathsf{e}_i)>0,\delta(\mathsf{m}_j)>0,$ $1\leq j\leq k$	(*C-EventG*)
$\langle \mathsf{task}(\mathsf{e}_i,\mathsf{e}_o),\sigma,\delta\rangle \xrightarrow{\mathsf{e}_i} \langle \mathit{inc}(\mathit{dec}(\sigma,\mathsf{e}_i),\mathsf{e}_o),\delta\rangle$	$\sigma(\mathsf{e}_i)>0$	(*C-Task*)
$\langle \mathsf{taskRcv}(\mathsf{e}_i,\mathsf{m},\mathsf{e}_o),\sigma,\delta\rangle \xrightarrow{?\mathsf{m}}$ $\langle \mathit{inc}(\mathit{dec}(\sigma,\mathsf{e}_i),\mathsf{e}_o),\mathit{dec}(\delta,\mathsf{m})\rangle$	$\sigma(\mathsf{e}_i)>0,$ $\delta(\mathsf{m})>0$	(*C-TaskRcv*)
$\langle \mathsf{taskSnd}(\mathsf{e}_i,\mathsf{m},\mathsf{e}_o),\sigma,\delta\rangle \xrightarrow{!\mathsf{m}}$ $\langle \mathit{inc}(\mathit{dec}(\sigma,\mathsf{e}_i),\mathsf{e}_o),\mathit{inc}(\delta,\mathsf{m})\rangle$	$\sigma(\mathsf{e}_i)>0$	(*C-TaskSnd*)
$\langle \mathsf{interRcv}(\mathsf{e}_i,\mathsf{m},\mathsf{e}_o),\sigma,\delta\rangle \xrightarrow{?\mathsf{m}}$ $\langle \mathit{inc}(\mathit{dec}(\sigma,\mathsf{e}_i),\mathsf{e}_o),\mathit{dec}(\delta,\mathsf{m})\rangle$	$\sigma(\mathsf{e}_i)>0,$ $\delta(\mathsf{m})>0$	(*C-InterRcv*)
$\langle \mathsf{interSnd}(\mathsf{e}_i,\mathsf{m},\mathsf{e}_o),\sigma,\delta\rangle \xrightarrow{!\mathsf{m}}$ $\langle \mathit{inc}(\mathit{dec}(\sigma,\mathsf{e}_i),\mathsf{e}_o),\mathit{inc}(\delta,\mathsf{m})\rangle$	$\sigma(\mathsf{e}_i)>0$	(*C-InterSnd*)
$\langle \mathsf{subProc}(\mathsf{e}_i,C,\mathsf{e}_o),\sigma,\delta\rangle \xrightarrow{\mathsf{e}_i}$ $\langle \mathit{inc}(\mathit{dec}(\sigma,\mathsf{e}_i),\mathit{start}(C)),\delta\rangle$	$\sigma(\mathsf{e}_i)>0$	(*C-SubProc*$_{\mathit{Start}}$)
$\langle \mathsf{subProc}(\mathsf{e}_i,C,\mathsf{e}_o),\sigma,\delta\rangle \xrightarrow{\mathit{marked}(\sigma,\mathit{end}(C))}$ $\langle \mathit{inc}(\mathit{zero}(\sigma,\mathit{end}(C)),\mathsf{e}_o),\delta\rangle$	$\mathit{completed}(\langle C,\sigma,\delta\rangle)$	(*C-SubProc*$_{\mathit{End}}$)

$$\frac{\langle C_1,\sigma,\delta\rangle \xrightarrow{l} \langle\sigma',\delta'\rangle}{\langle C_1 \mid C_2,\sigma,\delta\rangle \xrightarrow{l} \langle\sigma',\delta'\rangle}\,(\textit{C-Int}_1) \qquad \frac{\langle C_2,\sigma,\delta\rangle \xrightarrow{l} \langle\sigma',\delta'\rangle}{\langle C_1 \mid C_2,\sigma,\delta\rangle \xrightarrow{l} \langle\sigma',\delta'\rangle}\,(\textit{C-Int}_2)$$

Fig. 2. BPMN Collaboration Semantics.

set of *labels* (of transitions that collaboration configurations can perform); and $\rightarrow\subseteq \mathcal{C}\times\mathcal{L}\times\mathcal{C}$ is a *transition relation*. We will write $\langle C,\sigma,\delta\rangle \xrightarrow{l} \langle C,\sigma',\delta'\rangle$ to indicate that $(\langle C,\sigma,\delta\rangle, l, \langle C,\sigma',\delta'\rangle)\in\rightarrow$ and say that 'the collaboration in the configuration $\langle C,\sigma,\delta\rangle$ can do a transition labelled l and become the collaboration configuration $\langle C,\sigma',\delta'\rangle$ in doing so'. Since collaboration execution only affects the current states, and not the collaboration structure, for the sake of readability we omit the structure from the target configuration of the transition. Thus, a transition $\langle C,\sigma,\delta\rangle \xrightarrow{l} \langle C,\sigma',\delta'\rangle$ is written as $\langle C,\sigma,\delta\rangle \xrightarrow{l} \langle\sigma',\delta'\rangle$.

The transition relation over collaboration configurations formalizes the execution of a collaboration in terms of edge and message marking evolution. It is defined by the rules in Fig. 2. Labels l represent computational steps and are defined as follows: $!\mathsf{m}$ and $?\mathsf{m}$ denote sending and receiving actions, respectively, and E denotes the set of edges from which a token is moved, thus permitting to identify the current position in the execution flow (for the sake of readability, we write the set $\{\mathsf{e}\}$ as e). Notably, despite the presence of labels, this has to be thought of as a reduction semantics, because labels are not used for synchronization (as instead it usually happens in labeled semantics), but only for keeping track of the performed action in order to enable the verification.

Before commenting on the rules, we introduce the auxiliary functions they exploit. 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: $inc(\sigma, \emptyset) = \sigma$ and $inc(\sigma, \{\mathsf{e}\} \cup E)) = inc(inc(\sigma, \mathsf{e}), E)$; the cases for dec are similar. As usual, the update finction for δ are defined in a way similar to σ's definitions. We also use the function $zero : \mathbb{S} \times \mathbb{E} \to \mathbb{S}$ that allows updating a state by setting to zero the number of tokens marking an edge in the state. Formally, it is defined as follows: $zero(\sigma, \mathsf{e}) = \sigma \cdot \{\mathsf{e} \mapsto 0\}$. Also in this case the function extends in a natural ways to sets of edges as follows: $zero(\sigma, \emptyset) = \sigma$ and $zero(\sigma, \{\mathsf{e}\} \cup E)) = zero(zero(\sigma, \mathsf{e}), E)$.

To check the completion of a sub-process, and more in general of processes and collaborations, we exploit the boolean predicate $completed(\langle C, \sigma, \delta \rangle)$. It is defined according to the prescriptions of the BPMN standard, which states that "a process instance is completed if and only if [...] there is no token remaining within the process instance; no activity of the process is still active. For a process instance to become completed, all tokens in that instance must reach an end node" and "a sub-process instance completes when there are no more tokens in the Sub-Process and none of its Activities is still active" [1, pp. 426, 431]. Then, a collaboration completes when all involved processes complete. The fact that in our formalisation we do not provide a specific construct for identifying processes does not raise any issue related to the collaboration completion check, as tokens cannot pass from one process to another and edge names are unique in the model. Thus, the collaboration/process/sub-process completion can be formalised as follows:

Definition 2. *Let C be a collaboration, having the form $\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}) \mid C'$, the predicate $completed(\langle C, \sigma, \delta \rangle)$ is defined as*

$$\sigma(\mathsf{e}_{cmp}) > 0 \wedge \ \sigma(\mathsf{e}_i) = 0 \wedge \ isZero(C', \sigma)$$

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

- *$isZero(\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}), \sigma)$ if $\sigma(\mathsf{e}_i) = 0$;*
- *$isZero(\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o), \sigma)$ if $\sigma(\mathsf{e}_{enb}) = 0$ and $\sigma(\mathsf{e}_o) = 0$;*
- *$isZero(\mathsf{andSplit}(\mathsf{e}_i, E_o), \sigma)$ if $\sigma(\mathsf{e}_i) = 0$ and $\forall \mathsf{e} \in E_o \ . \ \sigma(\mathsf{e}) = 0$;*
- *$isZero(\mathsf{xorSplit}(\mathsf{e}_i, E_o), \sigma)$ if $\sigma(\mathsf{e}_i) = 0$ and $\forall \mathsf{e} \in E_o \ . \ \sigma(\mathsf{e}) = 0$;*

- $isZero(\mathsf{andJoin}(E_i, \mathsf{e}_o), \sigma)$ *if* $\forall \mathsf{e} \in E_i$. $\sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{xorJoin}(E_i, \mathsf{e}_o), \sigma)$ *if* $\forall \mathsf{e} \in E_i$. $\sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_k, \mathsf{e}_{ok})), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\forall i \in \{1..k\}$. $\sigma(\mathsf{e}_{oi}) = 0$*;*
- $isZero(\mathsf{task}(\mathsf{e}_i, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{subProc}(\mathsf{e}_i, C, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(C_1 | C_2, \sigma)$ *if* $isZero(C_1, \sigma)$ *and* $isZero(C_2, \sigma)$*.*

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

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

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

We now briefly comment on some of the operational rules in Fig. 2. Rule *C-Start* starts the execution of a collaboration (sub-)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. Rule *C-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 *C-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. Similarly, rule *C-XorSplit* is applied when a token is available in the incoming edge of a XOR split gateway, the rule decrements the token in the incoming edge and increment the token in one of the outgoing edges, non-deterministically chosen. Rule *C-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 *C-XorJoin* is activated every time there is a token in one of the incoming edges, which is then moved to the outgoing edge. Rule *C-EventBased* 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, whose number of tokens in the meantime is decreased (i.e., a message from the corresponding queue is consumed). Rule *C-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 *C-TaskRcv* is activated not only when there is a token in the incoming edge, like the one related to simple tasks, but also when there is a message to be consumed. Similarly, rule *C-TaskSnd*, instead of consuming, send a message before moving the token to the outgoing edge. Rule *C-InterRcv* (resp. *C-InterSnd*) follows the same behavior of rule *C-TaskRcv* (resp. *C-TaskSnd*). Rules *C-SubProc*$_{Start}$ and *C-SubProc*$_{End}$ deal with a subprocess element. The former rule is activated only when there is a token in the incoming edge of the sub-process, which is then moved to the enabling edge of the start event in the sub-process body. Then, the sub-process behaves as its body till it completes, according to the completion check performed by the rule *C-SubProc*$_{End}$. When this rule is applied, it removes all tokens from the sequence edges of the sub-process body[1], and adds a token to the outgoing edge of the sub-process. Finally, Rules *C-Int*$_1$ and *C-Int*$_2$ deal with interleaving in a standard way.

1.3 Safeness and Soundness

We now provide a formal definition of the correctness properties we verify on multi-layer collaboration models.

Safeness refers to the occurrence of no more than one token along the same sequence edge of each process in the collaboration. Safeness formalisation is an important criterion of correctness for business process models, since an unsafe model could lead to errors in the execution, as shown in the following examples. The formalisation of the property is based on the following auxiliary function determining the maximum number of tokens marking the sequence edges of a process (this function relies on the standard function $max(\cdot)$ returning the maximum in a list of natural numbers).

$$\begin{array}{l}
\mathit{maxMarking}(\langle \mathsf{start}(\mathsf{e}_{init}, \mathsf{e}_o), \sigma, \delta\rangle) = \sigma(\mathsf{e}_o)\\
\mathit{maxMarking}(\langle \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}), \sigma, \delta\rangle) = \sigma(\mathsf{e}_i)\\
\mathit{maxMarking}(\langle \mathsf{andSplit}(\mathsf{e}_i, \{\mathsf{e}_{o1}, \ldots, \mathsf{e}_{ok}\}), \sigma, \delta\rangle) = \mathit{max}(\sigma(\mathsf{e}_i), \sigma(\mathsf{e}_{o1}), ..., \sigma(\mathsf{e}_{ok}))\\
\mathit{maxMarking}(\langle \mathsf{xorSplit}(\mathsf{e}_i, \{\mathsf{e}_{o1}, \ldots, \mathsf{e}_{ok}\}), \sigma, \delta\rangle) = \mathit{max}(\sigma(\mathsf{e}_i), \sigma(\mathsf{e}_{o1}), ..., \sigma(\mathsf{e}_{ok}))\\
\mathit{maxMarking}(\langle \mathsf{andJoin}(\{\mathsf{e}_{i1}, \ldots, \mathsf{e}_{ik}\}, \mathsf{e}_o), \sigma, \delta\rangle) = \mathit{max}(\sigma(\mathsf{e}_{i1}), ..., \sigma(\mathsf{e}_{ik}), \sigma(\mathsf{e}_o))\\
\mathit{maxMarking}(\langle \mathsf{xorJoin}(\{\mathsf{e}_{i1}, \ldots, \mathsf{e}_{ik}\}, \mathsf{e}_o), \sigma, \delta\rangle) = \mathit{max}(\sigma(\mathsf{e}_{i1}), ..., \sigma(\mathsf{e}_{ik}), \sigma(\mathsf{e}_o))\\
\mathit{maxMarking}(\langle \mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_k, \mathsf{e}_{ok})), \sigma, \delta\rangle) = \mathit{max}(\sigma(\mathsf{e}_i), \sigma(\mathsf{e}_{o1}), ..., \sigma(\mathsf{e}_{ok}))\\
\mathit{maxMarking}(\langle \mathsf{task}(\mathsf{e}_i, \mathsf{e}_o), \sigma, \delta\rangle) = \mathit{max}(\sigma(\mathsf{e}_i), \sigma(\mathsf{e}_o));\\
\mathit{maxMarking}(\langle \mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma, \delta\rangle) = \mathit{max}(\sigma(\mathsf{e}_i), \sigma(\mathsf{e}_o));\\
\mathit{maxMarking}(\langle \mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma, \delta\rangle) = \mathit{max}(\sigma(\mathsf{e}_i), \sigma(\mathsf{e}_o));\\
\mathit{maxMarking}(\langle \mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma, \delta\rangle) = \mathit{max}(\sigma(\mathsf{e}_i), \sigma(\mathsf{e}_o));\\
\mathit{maxMarking}(\langle \mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma, \delta\rangle) = \mathit{max}(\sigma(\mathsf{e}_i), \sigma(\mathsf{e}_o));\\
\mathit{maxMarking}(\langle \mathsf{subProc}(\mathsf{e}_i, C, \mathsf{e}_o), \sigma, \delta\rangle) = \mathit{max}(\sigma(\mathsf{e}_i), \sigma(\mathsf{e}_o), \mathit{maxMarking}(\langle C, \sigma, \delta\rangle));\\
\mathit{maxMarking}(\langle C_1 | C_2, \sigma, \delta\rangle) = \mathit{max}(\mathit{maxMarking}(\langle C_1, \sigma, \delta\rangle), \mathit{maxMarking}(\langle C_2, \sigma, \delta\rangle));
\end{array}$$

Now, a collaboration is defined to be safe if it is preserved that the maximum marking does not exceed one along the collaboration execution. We use $\rightarrow^*$ to denote the reflexive and transitive closure of $\rightarrow$.

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

Definition 3 (Safe collaborations). *A collaboration C is* safe *if and only if, given σ and δ such that isInit$(\langle C, \sigma, \delta\rangle)$, for all σ' and δ' such that $\langle C, \sigma, \delta\rangle \rightarrow^* \langle \sigma', \delta'\rangle$ we have that maxMarking$(\langle C, \sigma', \delta'\rangle) \leq 1$.*

Soundness is a more elaborated property, which is based on a notion of proper completion of a collaboration. Intuitively, it requires that from any reachable configuration it is possible to arrive in a (completed) configuration where all marked end events are marked exactly by a single token and all sequence edges are unmarked. However, this notion does not take into account enqueued messages that will never be consumed. Considering this aspect, as mentioned in Sections **??** and **??**, we provide two notions of soundness: one that requires message queues to be empty for a proper completion, and another that relaxes this requirement.

Definition 4 (Sound collaboration). *A collaboration C is* sound *if and only if, given σ and δ such that isInit$(\langle C, \sigma, \delta\rangle)$, 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 $\langle C, \sigma', \delta'\rangle \rightarrow^* \langle \sigma'', \delta''\rangle$, $\forall\, \mathsf{e}_{cmp} \in$ marked$(\sigma''$,end$(C))$. $\sigma''(\mathsf{e}_{cmp}) = 1$, isZero$(C, \sigma'')$, and $\forall\, \mathsf{m} \in \mathbb{M}$. $\delta''(\mathsf{m}) = 0$.*

Definition 5 (Message-Disregarding Sound collaboration). *A collaboration C is* message-disregarding sound *if and only if, given σ and δ such that isInit$(\langle C, \sigma, \delta\rangle)$, 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 $\langle C, \sigma', \delta'\rangle \rightarrow^* \langle \sigma'', \delta''\rangle$, $\forall\, \mathsf{e}_{cmp} \in$ marked$(\sigma''$,end$(C))$. $\sigma''(\mathsf{e}_{cmp}) = 1$, and isZero$(C, \sigma'')$.*

2 The $\mathcal{S}^3$ Supporting Tool: From Theory to Practice

As shown in Sec. 1.2, our BPMN operational semantics is defined in terms of an LTS. The LTS generated by our semantics implementation is minimal and deterministic. In fact, during the construction of the LTS, each time a new state has to be added we check if a state representing the same collaboration configuration (i.e., same σ and δ) is already present. In such a case, we connect the transition edge under construction to the existing state, without introducing a separate state corresponding to the same configuration. This characteristic of the generated LTS is then exploited (see Definition 10) for putting in relation the soundness of the collaboration with the existence in the LTS of a unique 'final' state (representing a configuration where all tokens are consumed and no other sequence or message edge is enabled).

Considering a collaboration C, we now formally introduce the notions of safeness, soundness and message-disregarding soundness related to the LTS induced by the semantics. Consequently, we prove the correspondence between this definitions and Def. 3, 4 and 5.

Definition 6 (Safe Labelled Transition System). *A LTS $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ of a collaboration is safe if and only if $\forall \langle C, \sigma, \delta \rangle \in \mathcal{C}$ we have that $\forall\ \mathtt{e}\ \in\ \mathbb{E}\ .\ \sigma(\mathtt{e}) \leq 1$.*

Theorem 1 (Safeness correspondence). *Let C be a collaboration and $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ its LTS, then C is safe if and only if $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is safe.*

Proof. We prove below the *if* and the *only if* parts of the theorem.

- (*if* part) In this case we have to show that if $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is safe then C is safe. The proof proceeds by contradiction. Suppose that C is unsafe. By Def. 3, given σ and δ such that $\mathit{isInit}(\langle C, \sigma, \delta \rangle)$, there exist σ' and δ' such that $\langle C, \sigma, \delta \rangle \overset{*}{\rightarrow} \langle \sigma', \delta' \rangle$ and $\mathit{maxMarking}(\langle C, \sigma', \delta' \rangle) > 1$. This means that $\exists\ \mathtt{e}\ \in\ \mathbb{E}\ .\ \sigma'(\mathtt{e}) > 1$. Hence, by Def. 6, $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is unsafe, which is a contradiction.
- (*only if* part) In this case we have to show that if C is safe then $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is safe. The proof proceeds by contradiction. Suppose that $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is unsafe. This means, by Def. 6, that there exists $\langle C, \sigma, \delta \rangle \in \mathcal{C}$ such that $\exists\ \mathtt{e}\ \in\ \mathbb{E}\ .\ \sigma(\mathtt{e}) > 1$. So that, there exists a state of the collaboration in which $\mathit{maxMarking}(\langle C, \sigma, \delta \rangle) > 1$. Hence, by Def. 3, C is unsafe, which is a contradiction. □

The formal definition of soundness requires the definition of the following auxiliary functions determining the incoming labels of a state in the LTS, the presence of an execution trace of the LTS where given labels occur more than one time, and the set of edge labels incoming to the end events of a collaboration.

Definition 7 (Incoming Labels). *Let $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ be an LTS and $\langle C, \sigma, \delta \rangle \in \mathcal{C}$, $\mathit{incoming}(\langle C, \sigma, \delta \rangle) = \{ l \in \mathcal{L} \mid \exists\, \sigma', \delta' : \langle C, \sigma', \delta' \rangle \xrightarrow{l} \langle \sigma, \delta \rangle \}$.*

Definition 8 (Labels Duplication). *Let* $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ *be an LTS and* $I \subseteq \mathcal{L}$ *a set of labels, predicate* $isNotDuplicated(\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle, I)$ *holds true if* $\forall\, l_i \in I$ *and* $\langle C, \sigma_1, \delta_1 \rangle, \langle C, \sigma_2, \delta_2 \rangle, \langle C, \sigma_3, \delta_3 \rangle \in \mathcal{C}$ *the sequence* $\langle C, \sigma_1, \delta_1 \rangle \xRightarrow{l_i} \langle C, \sigma_2, \delta_2 \rangle \xRightarrow{l_i} \langle C, \sigma_3, \delta_3 \rangle$*, where* $\xRightarrow{l_i} = \rightarrow^* \xrightarrow{l_i} \rightarrow^*$*, never holds.*

Definition 9 (End Events Incoming Labels). *Let* C *be a collaboration, then* $endIn(\cdot)$ *is inductively defined as follows:*

$$endIn(C_1 \mid C_2) = endIn(C_1) \cup endIn(C_2) \qquad endIn(\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})) = \{\mathsf{e}_i\}$$

$$endIn(C) = \emptyset \;\; \textit{for any element } C \neq \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})$$

Now, our notions of soundness on LTSs and their correspondence with the definitions on collaborations can be defined as follows.

Definition 10 (Soundness Labelled Transition System). *A LTS* $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ *of a collaboration* C *is* sound *if and only if* $\exists! \langle C, \sigma, \delta \rangle \in \mathcal{C}$ *such that:*

(i) $\langle C, \sigma, \delta \rangle \not\rightarrow$ *(i.e.,* $\nexists\, l, \sigma', \delta'$ *such that* $\langle C, \sigma, \delta \rangle \xrightarrow{l} \langle \sigma', \delta' \rangle$*)*
(ii) $isNotDuplicated(\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle, incoming(\langle C, \sigma, \delta \rangle))$
(iii) $incoming(\langle C, \sigma, \delta \rangle) = endIn(C)$
(iv) $\forall\ \mathsf{e} \in \mathbb{E}\ .\ \sigma(\mathsf{e}) = 0$
(v) $\forall\ \mathsf{m} \in \mathbb{M}\ .\ \delta(\mathsf{m}) = 0$

Definition 11 (Message-Disregarding Soundness Labelled Transition System). *A LTS* $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ *of a collaboration* C *is* message-disregarding sound *if and only if* $\forall \langle C, \sigma, \delta \rangle \in \mathcal{C}$ *such that* $\langle C, \sigma, \delta \rangle \not\rightarrow$ *we have that:*

(i) $isNotDuplicated(\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle, incoming(\langle C, \sigma, \delta \rangle))$
(ii) $incoming(\langle C, \sigma, \delta \rangle) = endIn(C)$
(iii) $\forall\ \mathsf{e} \in \mathbb{E}\ .\ \sigma(\mathsf{e}) = 0$

Theorem 2 (Soundness Correspondence). *Let* C *be a collaboration and* $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ *its LTS, then* C *is sound if and only if* $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ *is sound.*

Proof. We prove below the *if* and the *only if* parts of the theorem.

- (*if* part) In this case we have to show that if $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is sound then C is sound. The proof proceeds by contradiction.
  By Def. 10, $\exists! \langle C, \sigma'', \delta'' \rangle \in \mathcal{C}$ such that $\langle C, \sigma'', \delta'' \rangle \overset{l}{\not\rightarrow}$ and $isNotDuplicated(\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle, incoming(\langle C, \sigma'', \delta'' \rangle))$ and $incoming(\langle C, \sigma'', \delta'' \rangle) = endIn(C)$ and $\forall\ \mathsf{m} \in \mathbb{M}\ .\ \delta''(\mathsf{m}) = 0$
  By contradiction, suppose C is unsound, then $\forall \langle C', \sigma', \delta' \rangle : \langle C, \sigma, \delta \rangle \overset{*}{\rightarrow} \langle \sigma', \delta' \rangle$ $\exists \langle C, \sigma'', \delta'' \rangle$ such that:
  - $\langle C, \sigma', \delta' \rangle \overset{l}{\not\rightarrow} \langle \sigma'', \delta'' \rangle$. This implies that the collaboration never reaches a final configuration, hence the generated LTS do not show an unique final state, contradicting the hypothesis. Hence, by Def. 10, $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is unsound, which is a contradiction.

  - $\mathit{completed}(\langle C, \sigma'', \delta''\rangle)$ is *false*. Hence, there exists e such that $\sigma''(\mathsf{e}) > 0$, contradicting the hypothesis. Hence, by Def. 10, $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ is unsound, which is a contradiction.
  - $\exists\, \mathsf{e}_{cmp} \in \mathit{marked}(\sigma'',\mathit{end}(C))\ .\ \sigma''(\mathsf{e}_{cmp}) > 1$. Due to this, there exists $\langle \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}), \sigma, \delta\rangle$ reached by more than one token. It means that $\exists \langle C, \sigma_1, \delta_1\rangle, \langle C, \sigma_2, \delta_2\rangle$ and $\langle C, \sigma_3, \delta_3\rangle \in \mathcal{C}$ such that $\langle C, \sigma_1, \delta_1\rangle \xRightarrow{\mathsf{e}_i} \langle C, \sigma_2, \delta_2\rangle \xRightarrow{\mathsf{e}_i} \langle C, \sigma_3, \delta_3\rangle$ where $\mathsf{e}_i \in \mathit{endIn}(C)$ is the incoming edge this end event. Hence, by Def. 10, $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ is unsound, which is a contradiction.
  - $\mathit{isZero}(C, \sigma'')$ is *false*. Then there exists an edge $\mathsf{e} \in \mathbb{M}\ .\ \sigma''(\mathsf{e}) > 0$. Hence, by Def. 10, $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ is unsound, which is a contradiction.
  - $\exists\, \mathsf{m} \in \mathbb{M}\ .\ \delta''(\mathsf{m}) \neq 0$. Hence, by Def. 10, $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ is unsound, which is a contradiction.

- (*only if* part) In this case we have to show that if C is sound then $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ is sound. The proof proceeds by contradiction.

  By Def. 4, let $\langle C, \sigma, \delta\rangle$ be the collaboration configuration such that $\mathit{isInit}(\langle C, \sigma, \delta\rangle)$ then $\forall \sigma'$ and δ' such that $\langle C, \sigma, \delta\rangle \rightarrow^* \langle \sigma', \delta'\rangle$ there must be σ'' and δ'' such that $\langle C, \sigma', \delta'\rangle \rightarrow^* \langle \sigma'', \delta''\rangle$, $\forall\, \mathsf{e}_{cmp} \in \mathit{marked}(\sigma'',\mathit{end}(C))\ .\ \sigma''(\mathsf{e}_{cmp}) = 1$, $\mathit{isZero}(C, \sigma'')$, and $\forall\, \mathsf{m} \in \mathbb{M}\ .\ \delta''(\mathsf{m}) = 0$.

  By contradiction, suppose $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ is unsound, then $\forall \langle C, \sigma'', \delta''\rangle \in \mathcal{C}$ such that:

  - $(\langle C, \sigma'', \delta''\rangle, l, \langle C, \sigma''', \delta'''\rangle) \in \rightarrow$. In this case each configuration of C can perform an action $l : \sigma''(l) > 0$ if $l \in \mathbb{E}$ or $\delta''(l) > 0$ if $l \in \mathbb{M}$. Consequently each state shows either $\sigma''(l) > 0$, contradicting $\mathit{isZero}(\langle C, \sigma'', \delta''\rangle)$ that become *false*, or $\delta''(l) > 0$. Hence, by Def. 4, C is unsound, which is a contradiction.
  - $\mathit{isNotDuplicated}(\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle, \mathit{incoming}(\langle C, \sigma'', \delta''\rangle))$ is *false*. This means that there exists $l \in \mathit{incoming}(\langle C, \sigma'', \delta''\rangle)$ and $\langle C, \sigma_1, \delta_1\rangle, \langle C, \sigma_2, \delta_2\rangle$ and $\langle C, \sigma_3, \delta_3\rangle \in \mathcal{C}$ such that $\langle C, \sigma_1, \delta_1\rangle \xRightarrow{l_i} \langle C, \sigma_2, \delta_2\rangle \xRightarrow{l_i} \langle C, \sigma_3, \delta_3\rangle$ holds. By hypothesis, $\mathit{incoming}(\langle C, \sigma'', \delta''\rangle) = \mathit{endInput}(C)$ so that $l \in \mathit{endInput}(C)$, but more precisely l is an incoming edge of and end event. Having a repetition of this label in a sequence of execution means that the related end event is reached by more than one token and that $\sigma''(\mathsf{e}_{cmp}) > 1$. Hence, by Def. 4, C is unsound, which is a contradiction.
  - $\mathit{incoming}(\langle C, \sigma'', \delta''\rangle) \neq \mathit{endInput}(C)$. Let l be a sequence flow label such that $l \in \mathit{endInput}(C)$, then l is an incoming of an end event. On the contrary, $l \notin \mathit{incoming}(\langle C, \sigma'', \delta''\rangle)$, then $\nexists \langle C, \sigma'', \delta''\rangle$ in which this end event is reached, so that $\sigma''(\mathsf{e}_{cmp}) > 1$. Hence, by Def. 4, C is unsound, which is a contradiction.
  - $\exists\, \mathsf{e} \in \mathbb{E}\ .\ \sigma''(\mathsf{e}) \neq 0$. Hence, there exists an element of the collaboration with a token in the incoming sequence flow, so that $\mathit{isZero}(\langle C, \sigma'', \delta''\rangle)$ is *false*. Hence, by Def. 4, C is unsound, which is a contradiction.
  - $\exists\, \mathsf{m} \in \mathbb{M} : \delta''(\mathsf{m}) \neq 0$. Hence, by Def. 4, C is unsound, which is a contradiction.

□

Theorem 3 (Message-Disregarding Soundness Correspondence). *Let C be a collaboration and $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ its LTS, then C is message-disregarding sound if and only if $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is message-disregarding sound.*

Proof. The proof proceeds similar to that of Theorem 2. □

References

1. OMG: Business Process Model and Notation (BPMN V 2.0) (2011)