Synthesis of Graphical Choreographies

Julien Lange[1], Nobuko Yoshida[1], and Emilio Tuosto[2]

[1] Imperial College London, UK [2] University of Leicester, UK

Abstract. Graphical choreographies, or global graphs, are general multiparty session specifications featuring expressive constructs such as forking, merging, and joining for representing application-level protocols. Global graphs can be directly translated into modelling notations such as BPMN and UML. This paper presents an algorithm whereby a global graph can be synthesised from asynchronous buffered behaviours represented by communicating finite state machines (CFSMs). Our results include: a sound and complete characterisation of a subset of safe CFSMs from which global graphs can be synthesised; a synthesis algorithm to translate CFSMs to global graphs; a time complexity analysis; and an implementation of our theory, as well as an experimental evaluation.

1 Introduction

Context Choreographies, models of interactions among software components from a global point of view, are becoming a paramount conceptual and practical tool to tackle the complexity of designing, analysing, and implementing modern applications (see e.g., [4, 8, 12, 22, 28]). As described in [12], not only a global perspective of the coordination of applications is useful to develop and test single components that conform to it, but also a global specification can be projected so to obtain the local behaviour of components. This is a distinctive element of choreographies and it appeals to industry [12, 28] since it enables the developers of a component to check it against the corresponding projection of the choreography.

One limitation of choreography-based development is its inherently uni-directional approach to software development life cycle (SDLC). Distributed service architectures envisage software as a provision made available, through a public interface that hides implementation details, to be searched by and composed with other similar software. Hence having a fixed set of choreographies for the whole life cycle of an application is not adequate. A challenge is to provide an automatic composition of interfaces (specifications) of such distributed pieces of software, which can then be compared to the original specification (if any) and be used to enforce desired safety properties. To tackle this problem, we develop an algorithm to synthesise choreographies from a set of behavioural specifications of components that interact by exchanging messages asynchronously. Our synthesis algorithm can generate expressive choreographies which resemble modelling notations used in industry, equipped with forking, merging, and joining constructs for controlling general flows of interactions. The synthesis now enables a bi-directional SDLC based on choreographies: a developer can visualise a global viewpoint of the interactions within a distributed applications via our synthesis algorithm; thus, when the synthesised choreography is not the expected one, either existing components or the global specification may be refined. If the synthesised choreography is modified, it can be projected again so to be compared with the original projections.

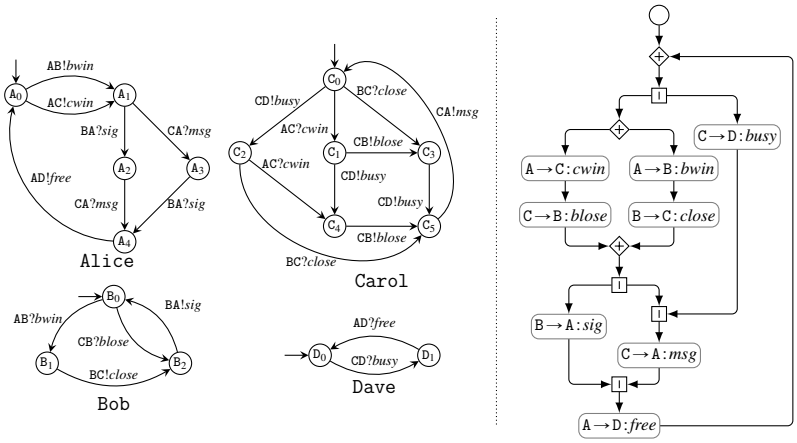


Fig. 1. Communicating System S_{re} (left) and its Global Graph G_{re} (right)

Our approach We adopt *communicating finite state machines* (CFSMs) as suitable behavioural specifications of distributed components for the synthesis. CFSMs are a conceptually simple model, based on asynchronous FIFO message-passing communication, and are well-established for analysing properties of distributed systems. They are also widely used in industry tools and can be seen as end-point specifications.

We define an algorithm that, given a set of CFSMs, yields a choreography expressed as a *global graph* [14], which are closely related to BPMN 2.0 Choreography, advocated as a suitable notation for services [7]. The system S_{re} on the left of Fig. 1 will be our running example to illustrate our approach; S_{re} consists of four CFSMs that realise a protocol of a fictive game where:

1. `Alice` (`A`) sends either *bwin* to `Bob` (`B`) or *cwin* to `Carol` (`C`) to decide who wins the game. In the former case, `A` fires the transition `AB`!*bwin* whereby the message *bwin* is put in the FIFO buffer `AB` from `A` to `B`, and likewise in the latter case.
2. If `B` wins (that is the message *bwin* is on top of the queue `AB` and `B` consumes it by taking the transition `AB`?*bwin*), then he sends a notification (*close*) to `C` to notify her that she has lost. Symmetrically, `C` notifies `B` of her victory (*blose*).
3. During the game, `C` notifies `Dave` (`D`) that she is *busy*.
4. After `B` and `C` have been notified of the outcome of the game, `B` sends a signal (*sig*) to `A`, while `C` sends a message (*msg*) to `A`.
5. Once the result is sent, `A` notifies `D` that `C` is now *free* and a new round starts.

The protocol above shows that CFSMs capture many coordination constructs: (*i*) in step 1 `A` (non-deterministically) chooses the winner; (*ii*) in step 2 `B` has a sequential behaviour; (*iii*) in step 3 the parallel behaviour of `C` is rendered with the interleaving of transition `CD`!*busy*; (*iv*) in step 4 and 5 threads join and finally the protocol loops.

Understanding the protocol of our running example from the CFSMs is not easy. A much clearer specification is given by the global graph G_{re} (synthesised by our algorithm) on the right of Fig. 1. There, the choreography of the four components is much more evident: initially `A` and `C` send messages in parallel to `B` and `D` respectively, `A` chooses the winner, etc. The global graph is synthesised through a transformation

of the CFSMs into a safe Petri net. The transformation preserves the original CFSMs, which can be recovered by projecting the global graph.

Analysing properties of CFSMs is computationally hard. Is S_{re} deadlock-free? Will any sent message be eventually consumed? Will each participant eventually receive any message s/he is waiting for? Answering such questions is generally undecidable and not immediate even for the simple scenario in Fig. 1. We establish a decidable condition, called *generalised multiparty compatibility* that characterises a set of systems for which the three questions can be efficiently decided. Our synthesis algorithm can produce a global graph from any set of generalised multiparty compatible CFSMs. We implement a tool and evaluate it with protocols from the literature.

Synopsis § 2 reviews CFSMs. § 3 defines generalised multiparty compatibility (GMC), analyses its complexity (Prop. 3.1 and Prop. 3.2), and its soundness (Th. 3.1). § 3.3 discusses how our condition can be used to suggest amendments to fix non-GMC systems. The synthesis algorithm, its complexity (Prop. 4.1), and its completeness (Th. 4.1) are in § 4. Related work and conclusions are in § 5. The tool and experimental evaluation are also in § 5 and available from [17]. Proofs and omitted definitions are in the appendix.

2 Communicating Finite State Machines

Fix a finite set $\mathcal{P}$ of *participants* (ranged over by $\mathtt{p}$, $\mathtt{q}$, $\mathtt{r}$, $\mathtt{s}$, etc.) and a finite alphabet $\mathbb{A}$. The set of *channels* is $C \stackrel{\text{def}}{=} \{\mathtt{pq} \mid \mathtt{p},\mathtt{q} \in \mathcal{P} \text{ and } \mathtt{p} \neq \mathtt{q}\}$ while $Act \stackrel{\text{def}}{=} C \times \{!,?\} \times \mathbb{A}$ is the set of *actions* (ranged over by ℓ), $\mathbb{A}^*$ (resp. Act^*, ranged over by φ) is the set of finite words on $\mathbb{A}$ (resp. Act). Also, ε is the empty word, $|\varphi|$ denotes the length of φ, and $\varphi\varphi'$ is the concatenation of φ and φ' (we overload these notations for words over $\mathbb{A}^*$).

Definition 2.1 (CFSM). A *communicating finite state machine* is a finite transition system given by a 4-tuple $M = (Q, q_0, \mathbb{A}, \delta)$ where Q is a finite set of *states*, $q_0 \in Q$ is the initial state, and $\delta \subseteq Q \times Act \times Q$ is a set of *transitions*. ◇

The transitions of a CFSM are labelled by actions; label $\mathtt{sr}!a$ represents the *sending* of message a from machine $\mathtt{s}$ to $\mathtt{r}$ and, dually, $\mathtt{sr}?a$ represents the *reception* of a by $\mathtt{r}$. We write $\mathcal{L}(M) \subseteq Act^*$ for the language on Act accepted by the automaton corresponding to machine M where each state of M is an accepting state. A state $q \in Q$ with no outgoing transition is *final*; q is a *sending* (resp. *receiving*) if all its outgoing transitions are labelled with sending (resp. receiving) actions, and q is a *mixed* state otherwise.

A CFSM $M = (Q, q_0, \mathbb{A}, \delta)$ is *deterministic* if for all states $q \in Q$ and all actions $\ell \in Act$, if $(q,\ell,q'),(q,\ell,q'') \in \delta$ then $q' = q''$.[1] A CFSM M is *minimal* if there is no machine M' with fewer states than M such that $\mathcal{L}(M) = \mathcal{L}(M')$. Hereafter, we only consider deterministic and minimal CFSMs.

Definition 2.2 (Communicating system). Given a CFSM $M_{\mathtt{p}} = (Q_{\mathtt{p}}, q_{0\mathtt{p}}, \mathbb{A}, \delta_{\mathtt{p}})$ for each $\mathtt{p} \in \mathcal{P}$, the tuple $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ is a *communicating system* (*CS*). A *configuration* of S is a pair $s = (\vec{q};\vec{w})$ where $\vec{q} = (q_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ with $q_{\mathtt{p}} \in Q_{\mathtt{p}}$ and where $\vec{w} = (w_{\mathtt{pq}})_{\mathtt{pq}\in C}$ with $w_{\mathtt{pq}} \in \mathbb{A}^*$; component $\vec{q}$ is the *control state* and $q_{\mathtt{p}} \in Q_{\mathtt{p}}$ is the *local state* of machine $M_{\mathtt{p}}$. The *initial configuration* of S is $s_0 = (\vec{q_0};\vec{\varepsilon})$ with $\vec{q_0} = (q_{0\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$. ◇

[1] Sometimes, a CFSM is considered deterministic when $(q,\mathtt{sr}!a,q') \in \delta$ and $(q,\mathtt{sr}!a',q'') \in \delta$ then $a = a'$ and $q' = q''$. Here, we follow a different definition [11] in order to represent branching type constructs.

Hereafter, we fix a machine $M_{\mathtt{p}} = (Q_{\mathtt{p}}, q_{0\mathtt{p}}, \mathbb{A}, \delta_{\mathtt{p}})$ for each participant $\mathtt{p} \in \mathcal{P}$ and let $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ be the corresponding system.

Definition 2.3 (Reachable states and configurations). A configuration $s' = (\vec{q}'; \vec{w}')$ is *reachable* from another configuration $s = (\vec{q}; \vec{w})$ by *firing the transition* ℓ, written $s \xrightarrow{\ell} s'$ (or $s \to s'$), if there is $a \in \mathbb{A}$ such that either: (1) $\ell = \mathtt{sr}!a$ and $(q_{\mathtt{s}}, \ell, q'_{\mathtt{s}}) \in \delta_{\mathtt{s}}$ and (a) $q'_{\mathtt{p}} = q_{\mathtt{p}}$ for all $\mathtt{p} \neq \mathtt{s}$; and (b) $w'_{\mathtt{sr}} = w_{\mathtt{sr}}.a$ and $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ for all $\mathtt{pq} \neq \mathtt{sr}$; or (2) $\ell = \mathtt{sr}?a$ and $(q_{\mathtt{r}}, \ell, q'_{\mathtt{r}}) \in \delta_{\mathtt{r}}$ and (a) $q'_{\mathtt{p}} = q_{\mathtt{p}}$ for all $\mathtt{p} \neq \mathtt{r}$; and (b) $w_{\mathtt{sr}} = a.w'_{\mathtt{sr}}$ and $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ for all $\mathtt{pq} \neq \mathtt{sr}$.

The reflexive and transitive closure of $\to$ is $\to^*$. We write $s_1 \xrightarrow{\ell_1 \cdots \ell_m} s_{m+1}$ when, for some $s_2, \ldots, s_m$, $s_1 \xrightarrow{\ell_1} s_2 \cdots s_m \xrightarrow{\ell_m} s_{m+1}$. A sequence of transitions is *k-bounded* if no channel of any intermediate configuration on the sequence contains more than k messages. The *k-reachability set of S* is the largest subset $RS_k(S)$ of $RS(S)$ within which each configuration s can be reached by a k-bounded execution from s_0. The set of *reachable configurations of S* is $RS(S) = \{s \mid s_0 \to^* s\}$. $\diamond$

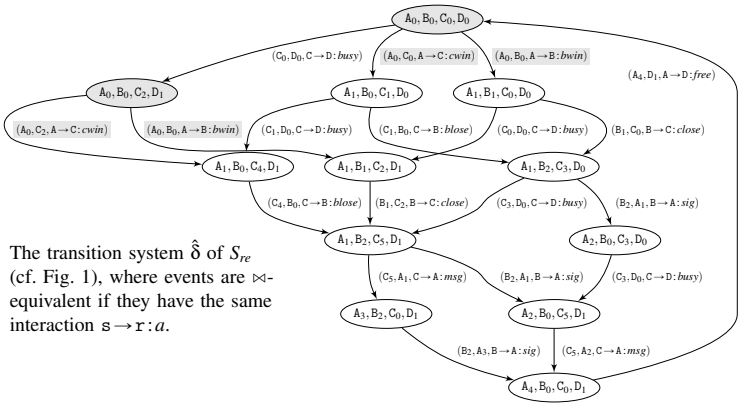
Condition (1-b) in Def. 2.3 puts a on channel $\mathtt{sr}$, while (2-b) gets a from channel $\mathtt{sr}$. Note that, for every integer k, the set $RS_k(S)$ is finite and computable.

We now give several definitions about communicating systems S and their configurations $s = (\vec{q}; \vec{w})$. We say that s is a *deadlock configuration* [11, Def. 12] if $\vec{w} = \vec{\varepsilon}$, there is $\mathtt{r} \in \mathcal{P}$ such that $(q_{\mathtt{r}}, \mathtt{sr}?a, q'_{\mathtt{r}}) \in \delta_{\mathtt{r}}$, and for every $\mathtt{p} \in \mathcal{P}$, $q_{\mathtt{p}}$ is a receiving or final state, i.e., all machines are blocked waiting for messages. Also, s is an *orphan message configuration* if all $q_{\mathtt{p}} \in \vec{q}$ are final but $\vec{w} \neq \vec{\varepsilon}$, i.e., there is at least a non-empty buffer and no machine is in a sending state. Finally, s is an *unspecified reception configuration* [11, Def. 12] if there exists $\mathtt{r} \in \mathcal{P}$ such that $q_{\mathtt{r}}$ is a receiving state, and $(q_{\mathtt{r}}, \mathtt{sr}?a, q'_{\mathtt{r}}) \in \delta_{\mathtt{r}}$ implies that $|w_{\mathtt{sr}}| > 0$ and $w_{\mathtt{sr}} \notin a\mathbb{A}^*$, i.e., $q_{\mathtt{r}}$ is prevented from receiving any message from buffer $\mathtt{sr}$. We say that S is *safe* if for each $s \in RS(S)$, s is not a deadlock, an orphan message, nor an unspecified reception configuration.

The following definitions are new and instrumental for § 3 where we characterise a subset of safe CS from which a global graph can be synthesised. Given $q, q' \in Q$, write $act(q, q') \stackrel{\text{def}}{=} \{\ell \mid (q, \ell, q') \in \delta\}$ and define $\diamond \subseteq \delta \times \delta$ and $\blacklozenge \subseteq \delta \times \delta$ as the smallest equivalence relations that respectively contain the relations $\underline{\diamond}$ and $\underline{\blacklozenge}$ where

$$(q_1, \ell, q_2) \underline{\diamond} (q'_1, \ell, q'_2) \iff \ell \notin act(q_1, q'_1) = act(q_2, q'_2) \neq \varnothing$$

and $(q_1, \ell, q_2) \underline{\blacklozenge} (q'_1, \ell, q'_2)$ when $(q_1, \ell, q_2) \underline{\diamond} (q'_1, \ell, q'_2)$ and for all $(q, \ell, q') \in [(q_1, \ell, q_2)]^{\diamond}$, $act(q_1, q) = act(q_2, q') \wedge act(q'_1, q) = act(q'_2, q')$. We let $[(q, \ell, q')]^{\diamond}$ denote the equivalence class of (q, ℓ, q') wrt $\diamond$. Intuitively, two transitions are $\blacklozenge$-related if they refer to the same action up-to interleaving. For example, in Fig. 1, $(\mathtt{C}_0, \mathtt{AC}?cwin, \mathtt{C}_1) \blacklozenge (\mathtt{C}_2, \mathtt{AC}?cwin, \mathtt{C}_4)$ since both transitions represent the same action interleaved with $\mathtt{CD}!busy$. In each machine in Fig. 1, a set of transitions (q, ℓ, q') with the same label ℓ forms a $\blacklozenge$-equivalence class, e.g., in $\mathtt{Alice}$, $\{(\mathtt{A}_1, \mathtt{CA}?msg, \mathtt{A}_3), (\mathtt{A}_2, \mathtt{CA}?msg, \mathtt{A}_4)\}$ is a $\blacklozenge$-equivalence class labelled by $\mathtt{CA}?msg$.



The transition system $\hat{\delta}$ of S_{re} (cf. Fig. 1), where events are $\bowtie$-equivalent if they have the same interaction $\mathtt{s} \to \mathtt{r} : a$.

Fig. 2. Transition graph of $\hat{\delta}$ and $TS(S_{re})$

3 CFSMs Characterisation of Global Graphs

3.1 Synchronous transition system

Systems amenable to be synthesised into global graphs are identified through *their synchronous transition system* (cf. Def. 3.2) where nodes consist of a vector of local states and transitions are labelled by elements in the set of *events* $\mathcal{E} \stackrel{\text{def}}{=} \bigcup_{\mathtt{s},\mathtt{r} \in \mathcal{P}} Q_{\mathtt{s}} \times Q_{\mathtt{r}} \times \{(\mathtt{s},\mathtt{r})\} \times \mathbb{A}\}$. Intuitively, an event $(q_{\mathtt{s}}, q_{\mathtt{r}}, \mathtt{s}, \mathtt{r}, a) \in \mathcal{E}$ ($(q_{\mathtt{s}}, q_{\mathtt{r}}, \mathtt{s} \to \mathtt{r} : a)$ for short) indicates that machines $\mathtt{s}$ and $\mathtt{r}$ can exchange message a when they are respectively in state $q_{\mathtt{s}}$ and $q_{\mathtt{r}}$. Indexing events with the local states of the machines permits to distinguish two occurrences of the same communication at two different points in a global graph. To single out parallelism at the machine level, we introduce an equivalence relation over events that identifies events whose underlying local transitions are $\blacklozenge$-equivalent.

Definition 3.1 ($\mathcal{E}$-equivalence). The *event equivalence* is the relation $\bowtie \stackrel{\text{def}}{=} \bowtie_{\mathtt{s}} \cap \bowtie_{\mathtt{r}} \subseteq \mathcal{E} \times \mathcal{E}$ where, for $(q_1, \mathtt{sr}!a, q_3), (q_1', \mathtt{sr}!a, q_4) \in \delta_{\mathtt{s}}$ and $(q_2, \mathtt{sr}?a, q_5), (q_2', \mathtt{sr}?a, q_6) \in \delta_{\mathtt{r}}$,

$$(q_1, q_2, \mathtt{s} \to \mathtt{r} : a) \bowtie_{\mathtt{s}} (q_1', q_2', \mathtt{s} \to \mathtt{r} : a) \iff (q_1, \mathtt{sr}!a, q_3) \blacklozenge (q_1', \mathtt{sr}!a, q_4)$$
$$(q_1, q_2, \mathtt{s} \to \mathtt{r} : a) \bowtie_{\mathtt{r}} (q_1', q_2', \mathtt{s} \to \mathtt{r} : a) \iff (q_2, \mathtt{sr}?a, q_5) \blacklozenge (q_2', \mathtt{sr}?a, q_6)$$

We let $[e]$ denote the $\bowtie$-equivalence class of event e. $\diamond$

For example (cf. Fig. 1), we have $(\mathtt{C}_5, \mathtt{A}_2, \mathtt{C} \to \mathtt{A} : msg) \bowtie (\mathtt{C}_5, \mathtt{A}_1, \mathtt{C} \to \mathtt{A} : msg)$ since the underlying transitions of $\mathtt{A}$ are $\blacklozenge$-equivalent, i.e., $(\mathtt{A}_1, \mathtt{CA}?msg, \mathtt{A}_3) \blacklozenge (\mathtt{A}_2, \mathtt{CA}?msg, \mathtt{A}_4)$, and the underlying transition of $\mathtt{C}$ is the same for both events, i.e., $(\mathtt{C}_5, \mathtt{CA}!msg, \mathtt{C}_0)$.

We let $n, n', \ldots$ denote vectors of local states and $n[\mathtt{p}]$ denote the state of $\mathtt{p} \in \mathcal{P}$ in n.

Definition 3.2 (Synchronous transition system). Given a system $S = (M_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$, let $N \stackrel{\text{def}}{=} \{\vec{q} \mid (\vec{q}; \vec{\varepsilon}) \in RS_1(S)\}$ (ranged over by n), $\hat{\delta} \stackrel{\text{def}}{=} \{(n, e, n') \mid (n; \vec{\varepsilon}) \xrightarrow{\mathtt{sr}!a} \xrightarrow{\mathtt{sr}?a} (n'; \vec{\varepsilon})$ and $e = (n[\mathtt{s}], n[\mathtt{r}], \mathtt{s} \to \mathtt{r} : a)\}$, and $E \stackrel{\text{def}}{=} \{e \mid \exists n, n' \in N : \ (n, e, n') \in \hat{\delta}\} \subseteq \mathcal{E}$. The *synchronous transition system of* S is $TS(S) = (N, n_0, E/\bowtie, \Rightarrow)$ where $n_0 = \vec{q}_0$ is the initial state, and $n \stackrel{[e]}{\Rightarrow} n' \iff (n, e, n') \in \hat{\delta}$. We fix a set $\hat{E}$ of representative elements of each $\bowtie$-equivalence class (i.e., $\hat{E} \subseteq E$ and $\forall e \in E \ \exists! e' \in \hat{E} : \ e' \in [e]$) and write $n \stackrel{e'}{\Rightarrow} n'$ for $n \stackrel{[e]}{\Rightarrow} n'$

when $e' \in [e] \cap \hat{E}$. Sequences of events are ranged over by π and we extend the notation on $\rightarrow$ in Def. 2.3 to $\Rightarrow$ (e.g., if $\pi = e_1 \cdots e_k$, $n_1 \stackrel{\pi}{\Rightarrow} n_{k+1}$ iff $n_1 \stackrel{e_1}{\Rightarrow} n_2 \stackrel{e_2}{\Rightarrow} \cdots \stackrel{e_k}{\Rightarrow} n_{k+1}$). $\diamond$

Definition 3.3 (Projections). The projection of an event $(q_{\mathtt{s}}, q_{\mathtt{r}}, \mathtt{s} \rightarrow \mathtt{r} : a)$ onto participant $\mathtt{p}$, denoted by $(q_{\mathtt{s}}, q_{\mathtt{r}}, \mathtt{s} \rightarrow \mathtt{r} : a)\!\downarrow_{\mathtt{p}}$, is defined as: (1) $\mathtt{pr}!a$ if $\mathtt{p} = \mathtt{s}$; (2) $\mathtt{sp}?a$ if $\mathtt{p} = \mathtt{r}$; and (3) ε otherwise (this definition is extended to sequence of events π in the obvious way). The projection of $TS(S) = (N, n_0, \hat{E}, \Rightarrow)$ onto participant $\mathtt{p}$, written $TS(S)\!\downarrow_{\mathtt{p}}$, is the automaton $(Q, q_0, \mathbb{A}, \delta)$ where (*i*) $Q = N$, (*ii*) $q_0 = n_0$, and (*iii*) $\delta \subseteq Q \times Act \cup \{\varepsilon\} \times Q$ is defined as $(n_1, e\!\downarrow_{\mathtt{p}}, n_2) \in \delta \iff n_1 \stackrel{e}{\Rightarrow} n_2$. $\diamond$

For $n \in N$, $TS(S)\langle n \rangle$ is the transition system $TS(S)$ where the initial state n_0 is replaced by n. We write $LT(S, n, \mathtt{p})$ for $\mathcal{L}(TS(S)\langle n \rangle\!\downarrow_{\mathtt{p}})$.

3.2 Generalised multiparty compatibility

We introduce *generalised multiparty compatibility* (GMC) as a sound and complete condition for synthesising global graphs. Hereafter, we fix a system $S = (M_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$ with $TS(S) = (N, n_0, \hat{E}, \Rightarrow)$. Basically, GMC relies on two conditions, (1) *representability* (cf. Def. 3.4): for each machine, each trace and each choice are represented in $TS(S)$ and (2) *branching property* (Def. 3.5): whenever there is a choice in $TS(S)$, a unique machine takes the decision and each of the other participants is either made aware of which branch was chosen or not involved in the choice. For a language $\mathcal{L}$, $hd(\mathcal{L})$ returns the first actions of $\mathcal{L}$ (if any).

$$hd(\mathcal{L}) \stackrel{\text{def}}{=} \{\ell \mid \exists \varphi \in Act^* \,:\, \ell \cdot \varphi \in \mathcal{L}\} \qquad\qquad hd(\{\varepsilon\}) \stackrel{\text{def}}{=} \{\varepsilon\}$$

Definition 3.4 (Representability). System S is *representable* if for all $\mathtt{p} \in \mathcal{P}$, (*i*) $\mathcal{L}(M_{\mathtt{p}}) = LT(S, n_0, \mathtt{p})$ and (*ii*) $\forall q \in Q_{\mathtt{p}}\ \exists n \in N \,:\, n[\mathtt{p}] = q \ \wedge\ \bigcup_{(q,\ell,q') \in \delta_{\mathtt{p}}} \{\ell\} \subseteq hd(LT(S, n, \mathtt{p}))$. $\diamond$

Condition (*i*) in Def. 3.4 is needed to ensure that each trace of each machine is represented in $TS(S)$; while condition (*ii*) is necessary to ensure that every choice in each machine is represented in $TS(S)$.

Proposition 3.1. *Given a system $S = (M_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$, checking whether S satisfies the representability condition is computable in $O(\sum_{\mathtt{p} \in \mathcal{P}} 2^{|N| + |Q_{\mathtt{p}}|})$ time, with $|N| = \prod_{\mathtt{p} \in \mathcal{P}} |Q_{\mathtt{p}}|$.*

In the worst case, the time complexity of checking the representability of S is exponential. This is solely due to the language equivalence check (condition (1) in Def. 3.4) between each machine and its projection from $TS(S)$. However, as observed in [6], in practice algorithms for language equivalence behave very efficiently. In addition, we can remove some states from the projection of $TS(S)$, e.g., those that are on chains of ε-transitions only, while preserving its language, thus reducing the exponent $|N|$.

We give a few auxiliary definitions before formalising the branching property. For $n \neq n' \in N$, we define $n \prec n'$ iff $n \Rightarrow^* n'$ and for all paths $n_0 \Rightarrow n_1 \Rightarrow \ldots \Rightarrow n_{k-1} \Rightarrow n_k = n$ in $TS(S)$, if $0 \leqslant i \neq j \leqslant k \wedge n_i \neq n_j$ then $\forall 0 \leqslant h \leqslant k \,:\, n' \neq n_h$. Intuitively, $n \prec n'$ holds if

n' is reachable from n and no simple path from n_0 to n goes through n'; note that $\prec$ is not a preorder in general. The *last nodes* reachable from $n \in N$ with $e_1 \neq e_2 \in \hat{E}$ are

$$ln(n,e_1,e_2) \stackrel{\text{def}}{=} \{(n_1,n_2) \mid \exists n' \in N : \forall i \in \{1,2\} : n \Rightarrow^* n' \stackrel{e_i}{\Rightarrow} n_i \wedge \forall n' \Rightarrow n'' : \neg(n' \prec n'' \stackrel{e_i}{\Rightarrow})\}$$

If $(n_1,n_2) \in ln(n,e_1,e_2)$, then n_i is a $\stackrel{e_i}{\Rightarrow}$-successor ($i = 1,2$) of a node n' on a path from n whose successors are either not able to fire both e_1 and e_2 or not $\prec$-related to n'. In Fig. 2, we have $ln((\mathtt{A}_0,\mathtt{B}_0,\mathtt{C}_0,\mathtt{D}_0),(\mathtt{A}_0,\mathtt{B}_0,\mathtt{A}\to\mathtt{B}:bwin),(\mathtt{A}_0,\mathtt{C}_0,\mathtt{A}\to\mathtt{C}:cwin)) = \{((\mathtt{A}_1,\mathtt{B}_1,\mathtt{C}_2,\mathtt{D}_1),(\mathtt{A}_1,\mathtt{B}_0,\mathtt{C}_4,\mathtt{D}_1))\}$, note that $(\mathtt{A}_0,\mathtt{C}_2,\mathtt{A}\to\mathtt{C}:cwin) \bowtie (\mathtt{A}_0,\mathtt{C}_0,\mathtt{A}\to\mathtt{C}:cwin)$.

For an event $e = (q_\mathtt{s}, q_\mathtt{r}, \mathtt{s}\to\mathtt{r}:a) \in \mathcal{E}$, let $\iota(e) = \mathtt{s}\to\mathtt{r}:a$ and define a dependency relation $\lhd \subseteq \mathcal{E}\times\mathcal{E}$ on events:

$$e \lhd e' \iff \iota(e) = \mathtt{s}\to\mathtt{r}:a \ \wedge\ (\iota(e') = \mathtt{s}\to\mathtt{r}:a' \vee \iota(e') = \mathtt{r}\to\mathtt{r}':a')$$

Intuitively, e and e' are $\lhd$-related if there exists a dependency relation between the two interactions, from the point of view of the receiver. We define a relation $e \blacktriangleleft e'$ in π if there is a $\lhd$-relation between e and e' in π, i.e.,

$$e \blacktriangleleft e' \text{ in } \pi \iff \begin{cases} (e \lhd e'' \wedge e'' \blacktriangleleft e' \text{ in } \pi') \vee e \blacktriangleleft e' \text{ in } \pi' & \text{if } \pi = e''\cdot\pi' \\ e \lhd e' & \text{otherwise} \end{cases}$$

also, $dep(\iota(e),\pi,\iota(e'))$ iff π has the form $\pi_1\cdot e\cdot\pi_2\cdot e'\cdot\pi'$ and

$$(\_,\_,\iota(e)) \notin \pi_1 \ \wedge\ (\_,\_,\iota(e')) \notin \pi_2 \implies e \blacktriangleleft e' \text{ in } \pi_2$$

which checks whether there is a dependency between two interactions on a path π (if these interactions do appear in π). Below we give the second condition for GMC.

Definition 3.5 (Branching property). System S has the *branching property* if for all $n \in N$ and for all $e_1 \neq e_2 \in \hat{E}$ such that $n \stackrel{e_1}{\Rightarrow} n_1$ and $n \stackrel{e_2}{\Rightarrow} n_2$, then we have that

1. either there is $n' \in N$ such that $n_1 \stackrel{e_2}{\Rightarrow} n'$ and $n_2 \stackrel{e_1}{\Rightarrow} n'$, or
2. for each $(n_1',n_2') \in ln(n,e_1,e_2)$, let $L^i_\mathtt{p} \stackrel{\text{def}}{=} hd\left(\{e_i\!\downarrow_\mathtt{p}\cdot\varphi \mid \varphi \in LT(S,n_i',\mathtt{p})\}\right)$ with $i \in \{1,2\}$ and $\mathtt{p} \in \mathcal{P}$, conditions (2a), (2b), and (2c) below hold.
   (a) $\forall \mathtt{p} \in \mathcal{P}$: either (*i*) $L^1_\mathtt{p} \cap L^2_\mathtt{p} \subseteq \{\varepsilon\}$ and $\varepsilon \in L^1_\mathtt{p} \iff \varepsilon \in L^2_\mathtt{p}$, or
   (*ii*) $\exists n' \in N :\ n_1' \stackrel{\pi_1}{\Rightarrow} n' \wedge n_2' \stackrel{\pi_2}{\Rightarrow} n' \wedge (e_1\cdot\pi_1)\!\downarrow_\mathtt{p} = (e_2\cdot\pi_2)\!\downarrow_\mathtt{p}$
   (b) $\exists!\mathtt{s} \in \mathcal{P} :\ L^1_\mathtt{s} \cap L^2_\mathtt{s} = \varnothing \wedge \exists \mathtt{sr}!a \in L^1_\mathtt{s} \cup L^2_\mathtt{s}$
   (c) $\forall \mathtt{r} \in \mathcal{P} :\ L^1_\mathtt{r} \cap L^2_\mathtt{r} = \varnothing \implies \forall \mathtt{s}_1\mathtt{r}?a_1 \in L^1_\mathtt{r}, \forall \mathtt{s}_2\mathtt{r}?a_2 \in L^2_\mathtt{r} :$
   $\forall i \neq j \in \{1,2\} :\ n_i' \stackrel{\pi_i}{\Rightarrow} \implies dep(\mathtt{s}_i\to\mathtt{r}:a_i,\ e_i\cdot\pi_i,\ \mathtt{s}_j\to\mathtt{r}:a_j)$ ◇

Def. 3.5 ensures that every branching either is (1) the concurrent execution of two events; or, for each participant $\mathtt{p}$, (2a-*i*) the first actions of $\mathtt{p}$ in two different branches are disjoint or $\mathtt{p}$ terminates before n; or (2a-*ii*) $\mathtt{p}$ is not involved in the choice, i.e., the branches merge and $\mathtt{p}$ can exhibit the same behaviour in both branches; (2b) there is a unique participant $\mathtt{s}$ making the decision; and (2c) for each participant $\mathtt{r}$ involved in the choice, there cannot be a race condition between the messages that $\mathtt{r}$ can receive.

Note that if a machine $\mathtt{r}$ receives all its messages from a same sender, then there is a $\lhd$-relation between all its actions.

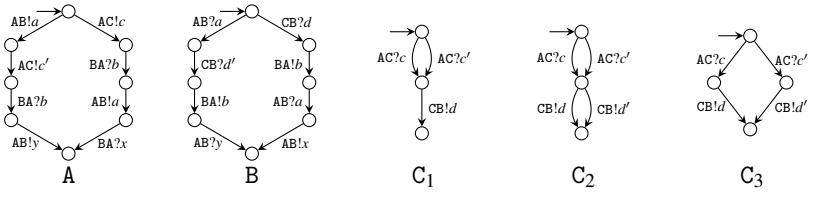
In system S_{re}, case (1) of Def. 3.5 applies to all branching nodes except $n_0 = (\mathtt{A}_0, \mathtt{B}_0, \mathtt{C}_0, \mathtt{D}_0)$ and $n = (\mathtt{A}_0, \mathtt{B}_0, \mathtt{C}_2, \mathtt{D}_1)$, highlighted in Fig. 2, for which case (2) applies since, for $e_1 = (\mathtt{A}_0, \mathtt{B}_0, \mathtt{A} \to \mathtt{B} : \mathit{bwin})$ and $e_2 = (\mathtt{A}_0, \mathtt{C}_0, \mathtt{A} \to \mathtt{C} : \mathit{cwin})$, we have $ln(n_0, e_1, e_2) = ln(n, e_1, e_2) = \{((\mathtt{A}_1, \mathtt{B}_1, \mathtt{C}_2, \mathtt{D}_1), (\mathtt{A}_1, \mathtt{B}_0, \mathtt{C}_4, \mathtt{D}_1))\}$. Hence, case (2a) holds for n_0 iff it holds for n. Following (2a), we check that every participant satisfies either (*i*) or (*ii*): $\mathtt{A}$ executes different (sending) actions in both branches ($\mathtt{AB}!\mathit{bwin}$ and $\mathtt{AC}!\mathit{cwin}$), $\mathtt{B}$ executes different (receiving) actions ($\mathtt{AB}?\mathit{bwin}$ and $\mathtt{CB}?\mathit{blose}$), $\mathtt{C}$ executes different (receiving) actions ($\mathtt{AC}?\mathit{cwin}$ and $\mathtt{BC}?\mathit{close}$), hence case (*i*) applies to $\mathtt{A}$, $\mathtt{B}$, and $\mathtt{C}$. While case (*ii*) applies to $\mathtt{D}$ since there is a node $n' = (\mathtt{A}_1, \mathtt{B}_2, \mathtt{C}_5, \mathtt{D}_1)$ such that $\mathtt{D}$ does not execute any action on either path from n to n' (through nodes $(\mathtt{A}_1, \mathtt{B}_1, \mathtt{C}_2, \mathtt{D}_1)$ and $(\mathtt{A}_1, \mathtt{B}_0, \mathtt{C}_4, \mathtt{D}_1)$, respectively). Also, condition (2b) is satisfied since $\mathtt{A}$ is the unique sender that executes different actions in both branches e_1 and e_2. Condition (2c) is satisfied for $\mathtt{B}$ and $\mathtt{C}$ due to the existence of dependency chains from $\mathtt{AB}?\mathit{bwin}$ to $\mathtt{CB}?\mathit{blose}$ (and vice-versa) and from $\mathtt{AC}?\mathit{cwin}$ to $\mathtt{BC}?\mathit{close}$ (and vice-versa). For instance, the dependency chain $\mathtt{B} \to \mathtt{C} : \mathit{close} \lhd \mathtt{C} \to \mathtt{A} : \mathit{msg} \lhd \mathtt{A} \to \mathtt{C} : \mathit{cwin}$ prevents $\mathtt{C}$ to delay the reception of *close* (sent by $\mathtt{B}$) until she can receive message *cwin* (sent by $\mathtt{A}$); $\mathtt{C}$ must send a message *msg* (to $\mathtt{A}$) before she can receive the outcome of a new round of the game. Finally, $ln(n_0, e_1, e_2)$ ensures that checking the branching between e_1 and e_2 at node n_0 is delayed until the interaction $\mathtt{C} \to \mathtt{D} : \mathit{busy}$ does not interfere with the choice. Hence, the behaviours of $\mathtt{C}$ and $\mathtt{D}$ are checked only once they have exchanged the *busy* message.

Proposition 3.2. *Given a system $S = (M_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$, checking whether S satisfies the branching property is computable in time $O\Big(|\Rightarrow|^2 \times |\Rightarrow|! \times \sum_{\mathtt{r} \in \mathcal{P}} \Big(|\delta_{\mathtt{r}}|^2\Big)\Big)$.*

Checking the branching property is factorial in the size of $TS(S)$ because it requires the enumeration of paths of $TS(S)$ (cf. (2a)(*ii*) and (2c) of Def. 3.5). We remark that the above is a rather coarse approximation obtained under worst case assumptions oblivious of the typical structure of $TS(S)$; our experiments show good performances (cf. § 5).

Definition 3.6 (Generalised multiparty compatibility). A system S is *generalised multiparty compatible* (*GMC*) if it is representable and has the branching property. $\diamond$

Example 3.1. We show the interplay between the representability and branching properties by giving examples of unsafe systems violating one property but not the other. Consider the following machines:



(1) System $S_1 = (\mathtt{A}, \mathtt{B}, \mathtt{C}_1)$ with $d = d'$ is not safe: whenever the left-hand side branch of $\mathtt{A}$ and the right-hand side branch of $\mathtt{B}$ are taken in a same execution, S_1 will reach

an orphan message configuration where messages x and y are never consumed. In fact, S_1 is not GMC because there is a branching node from which B can execute, as first actions, either AB?a or CB?d, and there is no dependency between the reception of a and that of d' in the left-hand side branch, i.e., $\neg(\mathtt{A}\to\mathtt{B}:a \lhd \mathtt{A}\to\mathtt{C}:c' \lhd \mathtt{C}\to\mathtt{B}:d')$. Thus the branching property does not hold.

(2) System $S_2 = (\mathtt{A},\mathtt{B},\mathtt{C}_2)$ is not safe: as before, whenever the left-hand side branch of A and the right-hand side branch of B are taken in a same execution this system reaches an orphan message configuration. These two branches are not mutually exclusive since $\mathtt{C}_2$ can receive c' then send d. This system is not GMC since there is no node in $TS(S_2)$ such that actions CB!d and CB!d' are the first actions executed by C. Hence the representability condition does not hold.

(3) System $S_3 = (\mathtt{A},\mathtt{B},\mathtt{C}_3)$ is safe and is GMC. In S_3, the left-hand side branch of A and the right-hand side branch of B are always mutually exclusive, while in S_1 and S_2 they are only mutually exclusive in synchronous executions.

Theorem 3.1 (Soundness). *If S is GMC, then it is* safe*, i.e., free from orphan message, deadlock and unspecified reception configurations.*

The proof (in appendix) requires to show that, for each branching node n, the function $ln(n,e_1,e_2)$ allows enough branches to be verified against the branching property.

3.3 Amending communicating systems

When a system is not GMC, our approach naturally suggests different ways of transforming it, so to validate the condition. By Def. 3.6, we first note:

Proposition 3.3. *If a system S satisfies all but condition (i) in Def. 3.4, then the system consisting of the projections of $TS(S)$ is GMC.*

This means that, in such a case, a new *safe* system may be automatically obtained from the projections of *TS*. For instance, the system S_2 in Ex. 3.1 is not GMC because (*i*) in Def. 3.4 does not hold. However, the system corresponding to the projections of $TS(S_2)$ is exactly the system S_3, which is GMC. In case the projections of TS does not provide a viable alternative, then the language equivalence check allows to highlight which transitions (or paths) of each machine are not represented in $TS(S)$. Similarly, for condition (*ii*) in Def. 3.4, local states and transitions violating it can be highlighted.

When the branching property (Def. 3.5) is violated, then our analysis permits to give precise information on where the problem occurs. If condition (2a) is violated, then a safe system may be obtained simply by renaming some messages in the original machines. These renamings can be automatically suggested while checking for the branching property. If condition (2b) is violated, we can highlight the node in which the problem occurs and the offending machines (i.e., the set of machine sending messages at this branching node) and a synchronous execution that leads to it. Finally, if condition (2c) is violated, then we can highlight on which messages a race condition may occur (for specific machines); and suggest to add an acknowledgement message between the two corresponding actions.

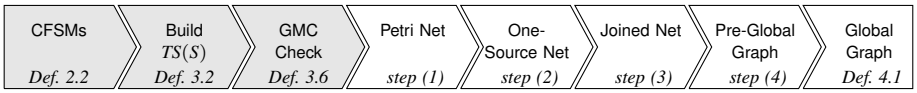


Fig. 3. Work-flow of the synthesis

4 Synthesising Global Graphs

In § 3, we construct the synchronous transition system $TS(S)$ of a communicating system S, and check whether it is GMC. We now describe the synthesis algorithm and its properties; Fig. 3 summarises the work-flow of the transformations.

The algorithm to synthesise a global graph G from a synchronous transition system $TS(S)$ consists of the following steps: (1) we apply the algorithm of Cortadella et al. [13] to synthesise a Petri net $\mathbb{N}$ from $TS(S)$; (2) we transform $\mathbb{N}$ so that its initial marking consists of exactly one place; (3) we join transitions whenever possible, so to make joins and forks explicit; (4) we transform the net of (3) into a pre-global graph; finally, we "clean-up" the pre-global graph of any unnecessary vertexes so to obtain a global graph. For the sake of space and because the transformations are rather mechanical, we only explain them through our running example. The formal definitions of the transformations and additional results are given in appendix.

For (1), it is enough for the reader to know that the algorithm in Cortadella et al. [13] is based on the theory of regions [2] and transforms a transition system into a safe and extended free-choice *labelled* Petri net; basically, this algorithm transforms events of $TS(S)$ into transitions of $\mathbb{N}$ while the places are built out of *regions*, i.e., sets of states having a uniform behaviour wrt events. We assume in this section that each $TS(S)$ is *self-loop free*, i.e., $\forall n,n' \in N :\ n \Rightarrow n' \implies n \neq n'$.[2] The algorithm of [13] is applicable on a self-loop free $TS(S)$, since every event $e \in \hat{E}$ has an occurrence in $TS(S)$, by construction, and every state n is reachable from n_0, by GMC. The Petri net obtained from $TS(S_{re})$ in Fig. 1 is given in Fig. 4 (top left).

In step (2), we transform a Petri net obtained from Cortadella's algorithm into a Petri net whose initial marking consists of exactly one place. This allows us to construct a global graph that has a unique starting point. In our running example, the Petri net at the top left of Fig. 4 is transformed by adding a fresh place (p_0), initially marked, and a fresh (silent) transition (t_0) connected to places p_1 and p_2.

In step (3), a transformation ensures that parallel gates are used "as much as possible" in the graph (instead of mixing choice and parallel gates). In fact, the transformation joins sets of places that have the same preset or postset to minimise the number of choice gates. The Petri net at the bottom left of Fig. 4 is the net obtained from the top left-hand side net after applying step (2) and (3). In the second transformation, we add (*i*) t_1 and p_{11} so to join p_1 and p_2 which have the same preset, i.e., t_0 and the transition with label $(\mathtt{A}_4, \mathtt{D}_1, \mathtt{A} \to \mathtt{D} : \mathit{free})$; and (*ii*) we add t_2 and p_{10} so to join p_5 and p_6 which have the same preset, i.e., the transitions with labels $(\mathtt{C}_1, \mathtt{B}_0, \mathtt{C} \to \mathtt{B} : \mathit{blose})$ and $(\mathtt{B}_1, \mathtt{C}_0, \mathtt{B} \to \mathtt{C} : \mathit{close})$. Both t_1 and t_2 are silent transitions.

[2] In $TS(S)$, if an event e self-loops, then any transition labelled by e is a self-loop. Hence, we can easily lift the self-loop free assumption by decomposing each self-loop into two (pointed) transitions in $TS(S)$ and recompose them once the global graph is synthesised.

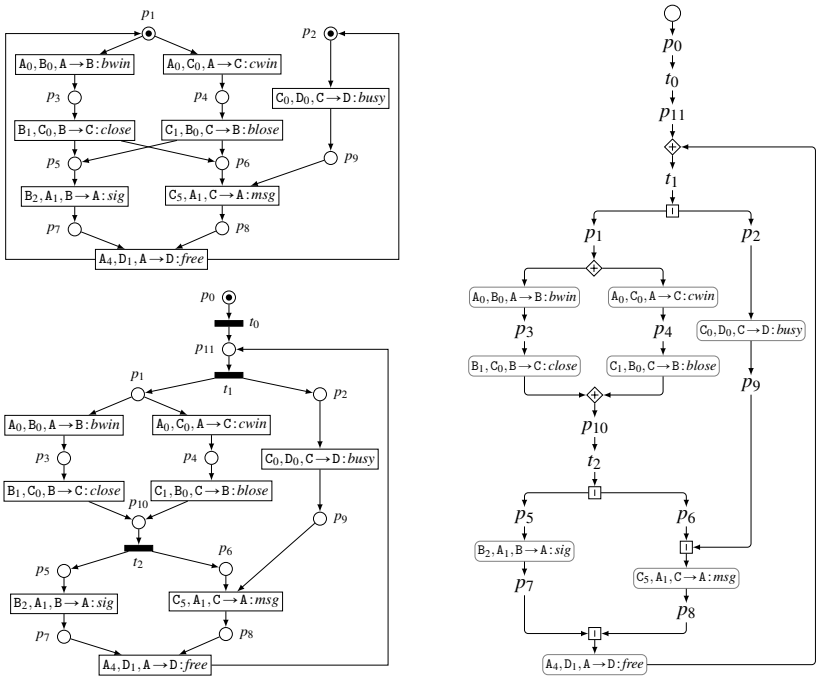


Fig. 4. Synthesised net (top left), net after transformations (bottom left), and pre-global graph

Lemma 4.1. *Given two Petri nets $\mathbb{N}$ and $\mathbb{N}'$, and their respective reachability graphs T and T', we write $T \approx T'$ if they are weakly bisimilar, i.e., they are bisimilar up-to silent transitions. If T (resp. T') is the reachability graph of the Petri net $\mathbb{N}$ obtained after step (1) (resp. (3)), then $T \approx T'$.*

We now define global graphs (a superclass of the generalised global types of [14] where each gate may be connected to more than two predecessors or successors).

Definition 4.1 (Global graph). A *global graph* (over $\mathcal{P}$ and $\mathbb{A}$) is a labelled graph $\langle V, A, \Lambda \rangle$ with set of vertexes V, set of edges $A \subseteq V \times V$, and labelling function Λ from V to $\{\bigcirc, ◎, ⟐, ⊡\} \cup \{\mathtt{s} \to \mathtt{r} : a \mid \mathtt{s}, \mathtt{r} \in \mathcal{P} \wedge a \in \mathbb{A}\}$ such that, $\Lambda^{-1}(\bigcirc)$ is a singleton, and for each $v \in V$, if $\Lambda(v)$ is of the form $\mathtt{s} \to \mathtt{r} : a$ then v has unique incoming and unique outgoing edges, and if $\Lambda(v) \in \{⟐, ⊡\}$, v has at least one incoming and one outgoing edge while v has no outgoing edges if $\Lambda(v) = ◎$. ◇

Label $\mathtt{s} \to \mathtt{r} : a$ represents an interaction where machine $\mathtt{s}$ sends a message a to machine $\mathtt{r}$. A vertex with label $\bigcirc$ represents the source of the global graph, ◎ represents the termination of a branch or thread, ⊡ indicates forking or joining threads, and ⟐ marks vertexes corresponding to branch or merge points, or to entry points of loops.

In step (4), a *pre-global graph* is obtained from the Petri net obtained after step (3) via a transformation which consists in creating a vertex in the global graph for each place, transition, and element of the flow relation. These vertexes are then connected

via gates: a source vertex is connected to a vertex without predecessor, a sink vertex is connected to any vertex without successors, while transitions (resp. places) are connected to ⊡-gate (resp. ⟐-gate) if they have more than one predecessors or successors. Each component of the graph is then connected by merging "ports" corresponding to element of the flow relation. The pre-global graph for Fig. 1 is given in Fig 4 (right).

A global graph is obtained from a pre-global graph by removing all unnecessary nodes (i.e., former places and transitions) and relabelling events into interactions (e is replaced by $\iota(e)$); e.g., the pre-global graph in Fig. 4 becomes the global graph in Fig. 1

Proposition 4.1. *Steps (2) to (4) are computable in polynomial time in the size of* $\mathbb{N}$.

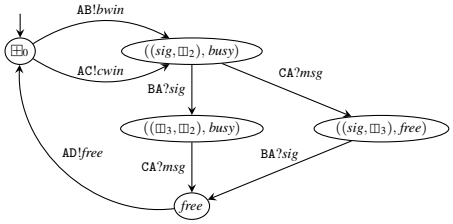


Fig. 5. Projection of G_{re} onto A

We give the main result regarding the synthesis of a global graph from CFSMs. In Th. 4.1, we formalise the relationship between the machines from which a global graph is synthesised and its projections. Projecting a global graph G can be done in two ways: (*i*) G can be transformed into a Petri net whose reachability graph may be projected, similarly to the projection of $TS(S)$ (cf. Def. 3.3); or (*ii*) G can be transformed into an automaton whose states are the nodes of G and each transition is labelled by $(\mathtt{s} \to \mathtt{r} : a)\!\downarrow_{\mathtt{p}}$ if the source state corresponds to a vertex with label $\mathtt{s} \to \mathtt{r} : a$, and by ε otherwise. In order to recover local concurrency, we take the parallel composition of the automata resulting of the projection of each successor of a ⊡-gate. Finally, the resulting automaton is minimised wrt. language equivalence. We write $G\!\downarrow_{\mathtt{p}}$ for the projection of G onto p, and give the formal definition in appendix. As an example, Fig. 5 shows the minimised projection of G_{re} (cf. Fig. 1) onto A.

Theorem 4.1 (Completeness). *Let* $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ *be GMC and* $TS(S) = (N, n_0, \hat{E}, \Longrightarrow)$. *If* $TS(S)$ *is such that* $\forall n, n' \in N : \; n \Rightarrow n' \implies n \neq n'$, *then the projection of the global graph G synthesised from S is such that S is isomorphic to* $(G\!\downarrow_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$.

The proof of Th. 4.1 is given in appendix. Essentially, it relies on the fact that each machine is preserved during the synthesis, i.e., (1) the projection of $TS(S)$ onto each p is language equivalent with $M_{\mathtt{p}}$, (2) the net obtained from $TS(S)$ via the algorithm in [13] is bisimilar to $TS(S)$, (3) each transformation preserves (weak) bisimilarity with the synthesised net, cf. Lemma 4.1, and (4) the transformation to a global graph is sound since the net is extended free choice.

5 Conclusions and Related Work

We have introduced a new class of communicating systems, called *generalised multiparty compatible* (GMC). We have proved that any system in this class is safe and there exist efficient algorithms to check GMC. We have given a complete algorithm whereby one can build (synthesise) a global graph (choreography) from any GMC system. Our work effectively uses the theory of regions [13], bridging a gap between a set of distributed uncontrolled behaviours (represented by CFSMs) and well-structured graphical

session types, while offering a scalable implementation for the synthesis. We are currently collaborating with the Savara project [28] to apply our framework to support the software development life cycle with tools based on BPMN 2.0 Choreography [7].

Experimental evaluation. In order to assess the applicability of our work to real-world protocols, and to evaluate the cost of checking for the GMC condition as well as synthesising a global graph, we have implemented a prototype tool [17] for our theory. The tool (implemented in Haskell) takes as input a textual representation of a CS, then builds a *TS* on which each part of the GMC condition is checked for, concurrently; then a global graph is synthesised from the *TS*. The tool relies on HKC [6] (to check for language equivalence), Petrify [26] (to synthesise a Petri net from *TS*), and Graphviz (to render global graphs).

The benchmarks were executed on a 3.40GHz Intel i7 CPU computer, with 16GB of RAM. The table below shows, for each protocol, the number of machines, the number of nodes and transitions in its *TS*, whether it validates the GMC condition, and the time it takes to check the condition *and* render its global graph.

S	$\lvert\mathcal{P}\rvert$	$\lvert N\rvert$	$\lvert\Longrightarrow\rvert$	GMC	Time (s)	S	$\lvert\mathcal{P}\rvert$	$\lvert N\rvert$	$\lvert\Longrightarrow\rvert$	GMC	Time (s)
Running Ex.	4	12	19	✓	0.169	Sanitary Agency [27]	4	17	21	✓	0.284
Bargain	3	4	4	✓	0.080	Health System [9]	6	10	11	✓	0.152
Alternating 2-bit [14]	2	8	12	✓	0.134	Filter Collaboration [30]	2	3	5	✓	0.093
Alternating 3-bit [14]	2	24	48	✓	2.969	Logistic [7]	4	13	17	✓	0.234
TPMContract v2 [19]	2	5	8	✓	0.129	Cloud System v4 [18]	4	7	8	✓	0.115

Most of the protocols are taken from the literature and all are checked within seconds. Graphical representations of these protocols are given in appendix. We note that it is slightly more expensive to check the Alternating 3-bit protocol due to larger $\blacklozenge$-equivalence classes.

Related work. In the context of multiparty session types, [23] first gave a construction of a global protocol from a set of local session types, up to asynchronous sub-typing. A typing system which infers a global type [20] from a set of session types is given in [21]. Recursive constructions are restricted in this work, due to an inherently syntax driven typing system, and multi-threaded participants are not supported. [15] studies the synthesis of global types from *basic* CFSMs, that is deterministic, non-mixed (each state is either sending or receiving), and directed (for each state, its outgoing transitions are all labelled by an action sending to, or receiving from, the same participant). The present work covers a much larger set of global protocols than [15, 21, 23]: we support mixed and non-directed states (hence, multi-threaded participants are allowed), recursive protocols are no longer restricted by a syntax oriented formalism, and explicit fork/join constructions may be synthesised.

The first translation from generalised global types into CFSMs was given in [14], where only sound properties were presented. The generalised global types of [14] are strictly included in GMC systems (Def. 3.6). The complete characterisation of global graphs and a synthesis were left as open problems. This paper closed these problems.

The term *synthesis* of CFSMs has been used to describe the reduction of CS to a more manageable (and decidable) model, e.g., with partial order approaches (see [25] for a summary of recent results). The acceptation of the term *synthesis* in this context is

to identify a system of CFSMs that realises a protocol described by an incomplete specification (such as in [3, 16, 24]). These approaches do not yield a global specification as instead achieved by our algorithm. Also, our approach enables the verification of trace-based properties surveyed in [25]. For instance, the closed synthesis of CFSMs can be reduced to the construction from a regular language L of a machine satisfying certain conditions related to buffer boundedness, deadlock-freedom, and words swapping.

In [22] a tool chain is given to synthesise an orchestrator (i.e., a message forwarder) from choreographies envisaged as a set of finite state machines communicating synchronously. This is transformed into a BPMN diagram via a Petri net transformation based on [13]. The work [29] gives an algorithm to compose several services. Each service is presented as an automaton and a set of automata are composed by a parallel product. The composite automaton is then transformed into a Petri net, using [13]. In both works, no result regarding safety or preservation of the behaviour of the original machines is given. The work [1] studies whether Message Sequence Charts (MSC) imply unspecified scenarios (where MSCs are implemented by concurrent automata, but do not necessarily feature order-preserving communications). It gives conditions on MSCs for their implementation to be deadlock-free and realisable. MSCs are realisable if no other MSC may be inferable from them. It does not attempt to give an exhaustive global view of a distributed system, but focus on identifying its possible misbehaviour.

Other recent works [4, 5, 10] study the relationship between global and local specifications, but do not consider the problem of synthesising global specifications from local ones. In [5], *synchronisable* systems are shown to preserve some reachability properties regardless the communication being asynchronous or synchronous. However, a subtle difference between our machines and the machines in [5] is that each of the latter machines has a unique buffer from which it can receive messages. Namely, their model is not suitable to reason about a CS as the interleaving of several multiparty sessions (where each participant has different receiving buffers in each session). Observe that the system $(\mathtt{A}, \mathtt{B}, \mathtt{C}_1)$, from Ex. 3.1, is unsafe in our communication model, but safe in theirs. In [4], the authors tackle the problem of determining whether a choreography is realisable. Essentially, a choreography is realisable if "it is possible to build a distributed system that communicates exactly as the choreography specifies". Choreographies in their work take the form of *conversation protocols*, that are finite state machines specifying the allowable sequence of interactions. A conversation protocol is akin to a global graph but without explicit construct for concurrent interactions. The realisability condition requires strong properties on message ordering, e.g., the choreography $\mathtt{A} \to \mathtt{B} : a ; \mathtt{C} \to \mathtt{B} : b$ is not realisable, however the system consisting of its projection is GMC. Note that the fundamental difference between this set of work and ours is that the former applies to global specifications, while GMC applies to local specifications (CFSMs).

References

1. R. Alur, K. Etessami, and M. Yannakakis. Inference of Message Sequence Charts. *IEEE Trans. Software Eng.*, 29(7):623–633, 2003.
2. E. Badouel and P. Darondeau. Theory of regions. In *Petri Nets*, volume 1491 of *LNCS*, pages 529–586. Springer, 1996.
3. C. Baier, J. Klein, and S. Klüppelholz. Synthesis of reo connectors for strategies and controllers. *Fundam. Inform.*, 130(1):1–20, 2014.

4. S. Basu, T. Bultan, and M. Ouederni. Deciding choreography realizability. In *POPL*, pages 191–202. ACM, 2012.
5. S. Basu, T. Bultan, and M. Ouederni. Synchronizability for verification of asynchronously communicating systems. In *VMCAI*, volume 7148 of *LNCS*. Springer, 2012.
6. F. Bonchi and D. Pous. Checking nfa equivalence with bisimulations up to congruence. In *POPL*, pages 457–468. ACM, 2013.
7. Business Process Model and Notation. `http://www.bpmn.org`.
8. M. Bravetti, I. Lanese, and G. Zavattaro. Contract-driven implementation of choreographies. In *TGC*, volume 5474 of *LNCS*, pages 1–18. Springer, 2008.
9. A. Bucchiarone, H. Melgratti, and F. Severoni. Testing service composition. In *Proceedings of the 8th Argentine Symposium on Software Engineering (ASSE07)*, 2007.
10. G. Castagna, M. Dezani-Ciancaglini, and L. Padovani. On global types and multi-party session. *LMCS*, 8(1), 2012.
11. G. Cécé and A. Finkel. Verification of programs with half-duplex communication. *I&C*, 202(2):166–190, 2005.
12. W. W. W. Consortium. Web services choreography description language version 1.0. `http://www.w3.org/TR/ws-cdl-10/`, 2005.
13. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri Nets for Finite Transition Systems. *IEEE Trans. Computers*, 47(8):859–882, 1998.
14. P. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *ESOP*, volume 7211 of *LNCS*, pages 194–213. Springer, 2012.
15. P.-M. Deniélou and N. Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP*, volume 7966 of *LNCS*, pages 174–186, 2013.
16. B. Genest, A. Muscholl, H. Seidl, and M. Zeitoun. Infinite-state high-level mscs: Model-checking and realizability. *J. Comput. Syst. Sci.*, 72(4):617–647, 2006.
17. GMC-Synthesis. `https://bitbucket.org/julien-lange/gmc-synthesis`.
18. M. Güdemann, G. Salaün, and M. Ouederni. Counterexample guided synthesis of monitors for realizability enforcement. In *ATVA*, volume 7561 of *LNCS*, pages 238–253, 2012.
19. S. Hallé and T. Bultan. Realizability analysis for message-based interactions using shared-state projections. In *SIGSOFT FSE*, pages 27–36. ACM, 2010.
20. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284. ACM, 2008.
21. J. Lange and E. Tuosto. Synthesising Choreographies from Local Session Types. In *CONCUR*, volume 7454 of *LNCS*, pages 225–239. Springer, 2012.
22. S. McIlvenna, M. Dumas, and M. T. Wynn. Synthesis of orchestrators from service choreographies. In *APCCM*, volume 96 of *CRPIT*, pages 129–138. ACS, 2009.
23. D. Mostrous, N. Yoshida, and K. Honda. Global principal typing in partially commutative asynchronous sessions. In *ESOP*, volume 5502 of *LNCS*, pages 316–332. Springer, 2009.
24. M. Mukund, K. N. Kumar, and M. A. Sohoni. Synthesizing distributed finite-state systems from mscs. In *CONCUR*, volume 1877 of *LNCS*, pages 521–535. Springer, 2000.
25. A. Muscholl. Analysis of communicating automata. In *LATA*, volume 6031 of *LNCS*, pages 50–57. Springer, 2010.
26. Petrify. `http://www.lsi.upc.edu/~jordicf/petrify/`.
27. G. Salaün, L. Bordeaux, and M. Schaerf. Describing and reasoning on web services using process algebra. In *ICWS*, pages 43–. IEEE Computer Society, 2004.
28. SAVARA and testable architecture. `http://www.jboss.org/savara`.
29. Y. Wang, A. Nazeem, and R. Swaminathan. On the optimal petri net representation for service composition. In *ICWS*, pages 235–242. IEEE Computer Society, 2011.
30. D. M. Yellin and R. E. Strom. Protocol specifications and component adaptors. *ACM Trans. Program. Lang. Syst.*, 19(2):292–333, 1997.

A Appendix: From Petri Nets to Global Graphs

In this section we give the detailed transformations omitted in Section 4. The algorithm to synthesise a global graph G from a synchronous transition system $TS(S)$ consists of the following steps: (1) using the algorithm of Cortadella et al. [13], we synthesise a Petri net $\mathbb{N}$ from $TS(S)$; (2) we transform $\mathbb{N}$ so that its initial marking consists of exactly one place (Transf. A.1 below); (3) we join transitions whenever possible, so to make joins and forks explicit (Transf. A.2 below); (4) we transform the net of (3) into a pre-global graph (Transf. A.3 below); finally, we "clean-up" the pre-global graph of any unnecessary vertexes so to obtain a global graph (Transf. A.4 below).

Definition A.1 (Labelled Net). A *labelled Petri net*, or net, $\mathbb{N}$ is a quadruple (P,T,F,m_0) with P a set of places (ranged over by p), T a set of transitions (ranged over by t), $F \subseteq (P \times T) \cup (T \times P)$ the flow relation, and m_0 the initial marking. Each transition $t \in T$ is labelled with an event $e \in \hat{E}$, or marker ε (the latter representing a silent transition). We let x range over elements of $P \cup T$. As usual, ${}^\bullet x$ (resp. $x^\bullet$) is the preset (resp. postset) of x. A net is called *safe* if no more than one token can appear in all reachable markings, in which case the reachable markings (including m_0) are sets of places. A net is *extended free-choice* if $\forall p \in P, \forall t \in T : \ (p,t) \in F \implies ({}^\bullet t \times p^\bullet) \subseteq F$. $\diamond$

In the second step (2), we transform a Petri net obtained from Cortadella's algorithm into a Petri net whose initial marking consists of exactly one place. This allows us to construct a global graph that has a unique starting point (source).

Transformation A.1 (One-source net). Given a labelled Petri net $\mathbb{N} = (P,T,F,m_0)$, the *one-source net* of $\mathbb{N}$ is $\mathbb{N}' = (P \cup \{p_0\}, T \cup \{t'\}, F', \{p_0\})$ such that $p_0 \notin P$, $t' \notin T$ is labelled by ε, and $F' = F \cup \{(p_0,t')\} \cup \bigcup_{p \in m_0} \{(t',p)\}$.

Proposition A.1. *Transf A.1 is computable in linear time in the size of m_0.*

The *reachability graph* of a net $\mathbb{N}$ is a transition system whose states are the reachable markings of $\mathbb{N}$ and there is a transition $(m_1, lab(t), m_2)$ iff t is enabled at marking m_1 and t produces the new marking m_2; where $lab(t)$ returns ε if the label of t is ε, and returns the label e of t otherwise. Given two reachability graphs T and T', we write $T \approx T'$ if they are weakly bisimilar, i.e., they are bisimilar up-to ε transitions. The definitions of reachability graphs and weak bisimulation are standard and given in Section C. We can now state the following result, formalising the soundness of Transf. A.1.

Lemma A.1. *If T is the reachability graph of the Petri net $\mathbb{N}$ obtained from $TS(S)$ via the algorithm in [13], and T' is the reachability graph of the Petri net obtained after applying Transf. A.1, then $T \approx T'$.*

Next, Transf. A.2 ensures that parallel gates are used "as much as possible" in the graph (instead of mixing choice and parallel gates). In fact, Transf. A.2 joins sets of places that have the same preset or postset to minimise the number of choice gates.

Transformation A.2 (Joined net). The *joined net* of $\mathbb{N} = (P,T,F,m_0)$ is a net $\mathbb{N}' = (P',T',F',m_0)$ such that the following transformations are applied repeatedly:

1. for all maximal $X \subseteq P$ s.t. $|X| > 1$ and $\forall p_1, p_2 \in X : {}^\bullet p_1 = {}^\bullet p_2 \wedge |{}^\bullet p_1| > 1$, $P' = P \cup \{p'\}$ and $T' = T \cup \{t'\}$ with $p' \notin P$ and $t' \notin T$ and labelled by ε; also, chosen $p \in X$, $F' = \{(p', t')\} \cup ({}^\bullet p \times \{p'\}) \cup (\{t'\} \times X) \cup F \setminus \bigcup_{x \in X} {}^\bullet x \times \{x\}$
2. for all maximal $X \subseteq P$ s.t. $|X| > 1$ and $\forall p_1, p_2 \in X : p_1{}^\bullet = p_2{}^\bullet \wedge |p_1{}^\bullet| > 1$, $P' = P \cup \{p'\}$ and $T' = T \cup \{t'\}$ with $p' \notin P$ and $t' \notin T$ and labelled by ε; also, chosen $p \in X$, $F' = \{(t', p')\} \cup (\{p'\} \times p^\bullet) \cup (X \times \{t'\}) \cup F \setminus \bigcup_{x \in X} \{x\} \times x^\bullet$.

Note that the definition of F' above does not depend on the choice of p.

Proposition A.2. *Transf. A.2 is computable in polynomial time in the size of* $\mathbb{N}$.

Since we are working with *safe* Petri nets, we have the following result.

Lemma A.2. *If T (resp. T') is the reachability graph of the Petri net $\mathbb{N}$ obtained after Transf. A.1 (resp. Transf. A.2), then $T \approx T'$.*

Definition A.2 (Graph composition). Let $\mathbb{N}_i = (P_i, T_i, F_i, m_{0_i})$ with $i \in \{1,2\}$ be two nets and $G_i = \langle V_i, A_i, \Lambda_i \rangle$ two graphs such that $V_i = P_i \cup T_i \cup F_i$, $i \in \{1,2\}$, the *composition of G_1 and G_2*, denoted by $G_1 \uplus G_2$ is a graph $\langle V, A, \Lambda \rangle$ defined as: $V = \{v \in V_1 \mid v \in F_1 \implies v \notin V_2\} \cup \{v \in V_2 \mid v \in F_2 \implies v \notin V_1\}$ and $A = ((A_1 \cup A_2) \cap V \times V) \cup \{(v, v') \mid \exists v'' \in F_i : (v, v'') \in A_i, (v'', v') \in A_j \wedge i \neq j \in \{1,2\}\}$ ◇

Intuitively, the composition of the graphs consists of (1) the union of the two sets of vertexes, except flow elements (p,t) and (t,p) if they appear in both V_1 and V_2; and (2) the union of the two sets of arcs between vertexes in V, and each pair of arcs of the form $(v,(x,x'))$ $((x,x'),v')$ is replaced by a single arc (v,v').

Transformation A.3 (Pre-global graph). The *pre-global graph* of $\mathbb{N} = (P,T,F,\{p_0\})$ is a tuple $\langle V, A, \Lambda \rangle$ such that $V = P \cup T \cup F$, Λ is a labelling function such that $\Lambda(v) = v$ if $v \in P \cup F$ or $v \in T$ labelled by ε, and $\Lambda(v) \in \hat{E} \cup \{◎, ○, ⊕, ⊡\}$ otherwise; and A is given by:

$$T_{\mathsf{g}}(\mathbb{N}) = \biguplus_{x \in P \cup T} T_{\mathsf{i}}(x) \uplus T_{\mathsf{o}}(x) \qquad \text{where, given } x \in P \cup T :$$

$$T_{\mathsf{i}}(x) \stackrel{\text{def}}{=} \begin{cases} ○ \to x & \text{if } {}^\bullet x = \varnothing \\ (x',x) \to x & \text{if } {}^\bullet x = \{x'\} \\ (x_1,x), \ldots, (x_i,x) \to \otimes \to x & \text{if } {}^\bullet x = \{x_1, \ldots, x_k\} \end{cases} \qquad T_{\mathsf{o}}(x) \stackrel{\text{def}}{=} \begin{cases} x \to ◎ & \text{if } x^\bullet = \varnothing \\ x \to (x,x') & \text{if } x^\bullet = \{x'\} \\ x \to \otimes \to (x,x_1), \ldots, (x,x_i) & \text{if } x^\bullet = \{x_1, \ldots, x_k\} \end{cases}$$

with $k > 1$, $\otimes = ⊕$ if $x \in P$, and $\otimes = ⊡$ if $x \in T$.

The pre-global graph for Fig. 1 is given in Fig 4 (right). Observe that all the vertexes of the form (x,x'), corresponding to an element of the flow relation, are removed as part of the graph composition (Def. A.2).

Proposition A.3. *Transf. A.3 is computable in polynomial time in the size of* $\mathbb{N}$.

We define the final transformation which cleans up a graph obtained from Transf. A.3 by removing unnecessary vertexes and arcs.

Transformation A.4. A global graph $G = \langle V, A, \Lambda \rangle$ is obtained from a pre-global graph $\langle P \cup T \cup F, A', \Lambda' \rangle$ by applying the following transformation: (1) replace each pair of transition $(x,p),(p,x') \in A'$ by $(x,x') \in A$; (2) replace each pair of transition $(x,t),(t,x') \in A'$, with t labelled by ε, by $(x,x') \in A$; and (3) label each t which is labelled by $(q_{\mathtt{s}}, q_{\mathtt{r}}, \mathtt{s} \to \mathtt{r}:a)$ in $\mathbb{N}$, by $\mathtt{s} \to \mathtt{r}:a$.

Proposition A.4. *Transf. A.4 is computable in polynomial time in the size of* $\mathbb{N}$.

B Appendix: Projections of Global Graphs

The definition of the projection of a global graph onto a participant, used in Section 4, is given below.

We first define a parallel composition of automata, which is required to project global graphs with a participant appearing in different threads. We define the $\_^\star$ function on pair of states: $q^\star = q'$ if $q = (q', q')$ and $q^\star = q$ otherwise; we overload it on sets of states, i.e., $Q^\star \stackrel{\text{def}}{=} \{q^\star \mid q \in Q\}$. We also define:

$$q \Subset q' \iff \begin{cases} q = q', \text{ or} \\ q' = (q_1, q_2) \wedge q \Subset q_i \wedge i \in \{1,2\} \end{cases}$$

We write $q \not\Subset q'$ iff $\neg(q \Subset q')$ and we overload the operator $\Subset$ on set of states such that $q \Subset Q \iff \exists q' \in Q : \; q \Subset q'$.

Definition B.1 (Parallel composition). The composition of $M_{\mathtt{i}} = (Q_{\mathtt{i}}, q_0^{\mathtt{i}}, \delta_{\mathtt{i}})$, $\mathtt{i} \in \{1,2\}$, written $M_1 \parallel M_2$, is the automaton $((Q_1 \times Q_2)^\star, (q_0^1, q_0^2)^\star, \delta)$ s.t.

$$((q_1,q_2)^\star, \ell, (q_1',q_2')^\star) \in \delta \iff \begin{cases} (q_{\mathtt{i}}, \ell, q_{\mathtt{i}}') \in \delta_{\mathtt{i}} & \text{if } q_{\mathtt{j}} = q_{\mathtt{j}}', q_{\mathtt{i}} \not\Subset Q_{\mathtt{j}} \text{ and } \mathtt{i} \neq \mathtt{j} \in \{1,2\} \\ (q_{\mathtt{i}}, \ell, q_{\mathtt{i}}') \in \delta_{\mathtt{i}} & \text{if } q_{\mathtt{i}} \Subset Q_{\mathtt{j}} \text{ and } \mathtt{i} \neq \mathtt{j} \in \{1,2\} \end{cases} \quad \diamond$$

Notice that $\parallel$ is a commutative and associative operation. Below, we give the definition of the projection function.

Definition B.2 (Projection). Given $G = \langle V', A, \Lambda \rangle$ and $v \in V'$, the *projection of* (G, v, V) *onto* $\mathtt{p}$, denoted by $(G, v, V)\!\upharpoonright_{\mathtt{p}}$, is defined as follows:

$$(G,v,V)\!\upharpoonright_{\mathtt{p}} = \begin{cases} \langle \{v\}, v, \varnothing, \varnothing \rangle & \text{if } \Lambda(v) = \circledcirc \text{ or } v \in V \\ \langle Q \cup \{v\}, v, \delta \cup \{(v,\ell,v')\} \rangle & \text{if } v' \in v^\bullet,\ \ell = \mathtt{s} \to \mathtt{r}:a\!\upharpoonright_{\mathtt{p}} \text{ and } \langle Q, v', \delta \rangle = (G, v', V \cup \{v\})\!\upharpoonright_{\mathtt{p}} \\ \langle \{v\} \cup \bigcup_{v' \in v^\bullet} Q_{v'}, v, \bigcup_{v' \in v^\bullet} \delta_{v'} \cup (v, \varepsilon, v') \rangle & \text{if } \Lambda(v) \in \{\bigcirc, \oplus\} \text{ and } \langle Q_v, v', \delta_v \rangle = (G, v', V \cup \{v\})\!\upharpoonright_{\mathtt{p}} \\ \langle Q, v, \delta \cup (v, \varepsilon, v'') \rangle & \text{if } \Lambda(v) = \square \text{ and } \langle Q, v'', \delta \rangle = \|_{v' \in v^\bullet} (G, v', V \cup \{v\})\!\upharpoonright_{\mathtt{p}} \end{cases}$$

Given a vertex $v \in V'$ such that $\Lambda(v) = \bigcirc$, the projection of G onto $\mathtt{p}$, written $G\!\upharpoonright_{\mathtt{p}}$, is the automaton, minimised wrt. language equivalence, $(Q, q_0, \delta, \mathbb{A})$ with $(G, v, \varnothing)\!\upharpoonright_{\mathtt{p}} = \langle Q, q_0, \delta \rangle$. $\diamond$

The projection of a global graph onto p uses the auxiliary function $(G, v, V)\downarrow_{\mathtt{p}}$. The function takes the following parameter, p: the identifier of the participant onto the projection is invoked, G: the global graph to be projected, v: a node in G used as an initial node for the projection, and V: a set of visited nodes.

If v has already been visited or it is a sink node, then a single state automaton is returned. If v is labelled by $\mathtt{s} \to \mathtt{r} : a$, then the projection of it successor is connected to v by a transition labelled by either $\mathtt{pr}!a$, $\mathtt{sp}?a$, or ε. If v is a choice gate or the source node, then the projection of each successor of v, connected by an ε transition from v, is returned. If v is a parallel gate, then the parallel composition of the projections of its successors are returned, connected to v by an ε transition. The parallel composition of automata uses Def. B.1, so that state identities are normalised and visited nodes are comparable with nodes produced by composing automata.

C Appendix: Equivalences between Petri Nets

We give the formal definitions of reachability graph of a Petri net and weak-bisimulation, which are used in Sections 4 and A.

Definition C.1 (Reachability Graph [13]). Given $\mathbb{N} = (P, T, F, m_0)$, we say that a transition $t \in T$ is enabled at marking m_1 if all its input places are marked. An enabled transition t may fire, producing a new marking m_2 with one less token in each input place and one more token in each output place. We write $m_1 \xrightarrow{t} m_2$, if m_2 is reachable from m_1 by firing t, and write $\to^*$ for the reflexive transitive closure of $\to$.

The *reachability graph* of $\mathbb{N}$ is the transition system $RG(\mathbb{N}) = (M, m_0, \hat{E}, \to)$ such that (1) $M = \{m \mid m_0 \to^* m\}$; (2) $\to = \{(m_1, lab(t), m_2) \mid m_1, m_2 \in M \wedge m_1 \xrightarrow{t} m_2\}$; and (3) $\hat{E} = \{e \mid \exists (m_1, e, m_2) \in \to \wedge e \neq \varepsilon\}$; where $lab(t)$ returns ε if the label of t is ε, and return the label e of t otherwise.

We write $m \xrightarrow{e} m'$ if $(m, e, m') \in \to$ and $m \overset{e}{\Rightarrow} m'$ if $m(\xrightarrow{\varepsilon})^* \xrightarrow{e} (\xrightarrow{\varepsilon})^* m'$, with $e \neq \varepsilon$. $\diamond$

In Def. C.2[3] we give a definition of weak bisimulation between two transition systems.

Definition C.2 (Weak bisimulation). Let $T = (M, m_0, \hat{E}, \to)$ be a transition system. A weak bisimulation on T is an equivalence relation $\mathcal{B} \subseteq M \times M$ such that for all $(m_1, m_2) \in \mathcal{B}$, the following holds

- $m_1 \xrightarrow{e} m_1'$ implies that there is m_2' such that $m_2 \overset{e}{\Rightarrow} m_2'$ and $(m_1', m_2') \in \mathcal{B}$; and
- $m_2 \xrightarrow{e} m_2'$ implies that there is m_1' such that $m_1 \overset{e}{\Rightarrow} m_1'$ and $(m_1', m_2') \in \mathcal{B}$.

Two states m_1 and m_2 are called *weakly bisimilar* on T, written $m_1 \approx_T m_2$, iff $(m_1, m_2) \in \mathcal{B}$ for some weak bisimulation $\mathcal{B}$.

Two transition systems $T_i = (M_i, m_0^i, \hat{E}_i, \to_i)$, $i \in \{1, 2\}$, such that $M_1 \cap M_2 = \varnothing$, are weakly bisimilar, written $T_1 \approx T_2$, if given $T' = (M_1 \cup M_2 \cup \{m_0\}, m_0, \hat{E}_1 \cup \hat{E}_2, \to_1 \cup \to_2 \cup \{(m_0, \varepsilon, m_0^1), (m_0, \varepsilon, m_0^2)\})$, $m_0^1 \approx_{T'} m_0^2$ holds. $\diamond$

[3] Adapted from R. Lanotte, A. Maggiolo-Schettini, and A. Troina. Weak bisimulation for probabilistic timed automata. *Theor. Comput. Sci.*, 411(50):42914322, 2010.

D Appendix: Proofs of Section 3

In this section, we first prove the key properties of the relations defined in the paper. Then we prove the main theorem (soundness), the reachability lemma and complexity results stated in Section 3.

We use the following functions in the proofs:

1. $\mathtt{snd}((q_\mathtt{s}, q_\mathtt{r}, \mathtt{s} \to \mathtt{r}:a)) = \mathtt{snd}(\mathtt{s} \to \mathtt{r}:a) \stackrel{\text{def}}{=} \mathtt{s}$;
2. $\mathtt{rcv}((q_\mathtt{s}, q_\mathtt{r}, \mathtt{s} \to \mathtt{r}:a)) = \mathtt{rcv}(\mathtt{s} \to \mathtt{r}:a) \stackrel{\text{def}}{=} \mathtt{r}$; and
3. $\mathtt{id}((q_\mathtt{s}, q_\mathtt{r}, \mathtt{s} \to \mathtt{r}:a)) = \mathtt{id}(\mathtt{s} \to \mathtt{r}:a) \stackrel{\text{def}}{=} \{\mathtt{s}, \mathtt{r}\}$.

D.1 Properties of $TS(S)$

Lemma D.1. *Let $S = (M_\mathtt{p})_{\mathtt{p} \in \mathcal{P}}$ be a GMC system and $TS(S) = (N, n_0, \hat{E}, \Rightarrow)$, for all $\mathtt{p} \in \mathcal{P}$ and for all $q \in Q_\mathtt{p}$ such that $(q, \ell_1, q_1), (q, \ell_2, q_2) \in \delta_\mathtt{p}$ for some $q_1, q_2 \in Q_\mathtt{p}$, there is $n, n_1, n_2 \in N$ such that $n[\mathtt{p}] = q$,*

- *$n \stackrel{\pi_1}{\Rightarrow} n_1 \stackrel{e_1}{\Rightarrow}$, with $\pi_1 \downarrow_\mathtt{p} = \varepsilon$, $e_1 \downarrow_\mathtt{p} = \ell_1$; and*
- *$n \stackrel{\pi_2}{\Rightarrow} n_2 \stackrel{e_2}{\Rightarrow}$, with $\pi_2 \downarrow_\mathtt{p} = \varepsilon$, $e_2 \downarrow_\mathtt{p} = \ell_2$.*

Proof. This result follows directly from the representability condition. □

Lemma D.2. *Let $S = (M_\mathtt{p})_{\mathtt{p} \in \mathcal{P}}$ be a system and $TS(S) = (N, n_0, \hat{E}, \Rightarrow)$. For all $n \neq n' \in N$, if $n \stackrel{e}{\Rightarrow} n'$, then $n' \stackrel{e}{\not\Rightarrow}$.*

Proof. By contradiction, assume there are $n \neq n' \in N$ such that $n \stackrel{e}{\Rightarrow} n'$ and $n' \stackrel{e}{\Rightarrow} n''$, where $e = (q_\mathtt{s}, q_\mathtt{r}, \mathtt{s} \to \mathtt{r} : a)$. There are two cases depending on whether the events come from distinct events in $\hat{\delta}$.

1. Assume $(n, e, n'), (n', e, n'') \in \hat{\delta}$, then it must be the case that $(q_\mathtt{s}, \mathtt{sr}!a, q_\mathtt{s}) \in \delta_\mathtt{s}$ and $(q_\mathtt{r}, \mathtt{sr}?a, q_\mathtt{r}) \in \delta_\mathtt{r}$, i.e., there is a corresponding self-loop in machines $\mathtt{s}$ and $\mathtt{r}$. Since the local source state and target states of both machines are the same, this contradicts the assumption that $n \neq n'$.
2. Assume $(n, e, n'), (n', e', n'') \in \hat{\delta}$, with $e \neq e'$, $e' = (q'_\mathtt{s}, q'_\mathtt{r}, \mathtt{s} \to \mathtt{r} : a)$, and $e \bowtie e'$. We must have that $\delta_\mathtt{s}(q_\mathtt{s}, \mathtt{sr}!a) = q'_\mathtt{s}$ and $\delta_\mathtt{r}(q_\mathtt{r}, \mathtt{sr}?a) = q'_\mathtt{r}$; and since $e \neq e'$ we must have that $q_\mathtt{s} \neq q'_\mathtt{s}$ or $q_\mathtt{r} \neq q'_\mathtt{r}$. Take $q_\mathtt{r} \neq q'_\mathtt{r}$, we must have $(q_\mathtt{r}, \mathtt{sr}?a, q'_\mathtt{r}), (q'_\mathtt{r}, \mathtt{sr}?a, q''_\mathtt{r}) \in \delta_\mathtt{r}$. This implies that we have $\neg((q_\mathtt{r}, \mathtt{sr}?a, q'_\mathtt{r}) \blacklozenge (q'_\mathtt{r}, \mathtt{sr}?a, q''_\mathtt{r}))$ since $\mathtt{sr}?a \in act(q_\mathtt{r}, q'_\mathtt{r})$. The cases where $q_\mathtt{s} \neq q'_\mathtt{s}$ and $q_\mathtt{s} \neq q'_\mathtt{s} \wedge q_\mathtt{r} \neq q'_\mathtt{r}$ are similar. Hence, by Def. 3.1, we conclude that $\neg(e \bowtie e')$ which contradicts our hypothesis.

□

D.2 Properties of $\prec$

Lemma D.3. *Let $n, n', n'' \in TS(S)$, if $n \prec n'$ and $n' \prec n''$, then $n \prec n''$.*

Proof. Assume $n \prec n'$ and $n' \prec n''$, by definition of $\prec$, we have

$$n_0 \Rightarrow^* n \Rightarrow^* n' \Rightarrow^* n''$$

which implies that there is a simple path from n to n'.

By contradiction, assume $\neg(n \prec n'')$. This implies that either (i) $n = n''$, which contradicts the fact that $n' \prec n''$; or (ii) there is a simple path from n_0 to n that includes n'', thus there is also a simple path from n_0 to n' that includes n'', which contradicts our assumption. □

Lemma D.4. *Let $n_1 \ldots n_k$ be a simple path in $TS(S)$, such that $n_k \Rightarrow n_1$. For all $1 \leqslant i, j \leqslant k$, if $n_i \prec n_j$, then $\neg(n_j \prec n_i)$.*

Proof. Take n_i and n_j such that $n_i \prec n_j$. This means that n_j never appears on a path from n_0 to n_i. Since $n_1 \ldots n_k$ forms a cycle, we must have $n_i \Rightarrow^* n_j$ and $n_j \Rightarrow^* n_i$. Thus we have

$$n_0 \Rightarrow^* n_i \Rightarrow^* n_j \Rightarrow^* n_i$$

and there is a simple path from n_0 to n_j that includes n_i (cf. simple path assumption), which implies that we cannot have $n_j \prec n_i$. □

D.3 Properties of $ln(n, e_1, e_2)$

Lemma D.5. *Let $S = (M_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$ be a system and $TS(S) = (N, n_0, \hat{E}, \Rightarrow)$. For all $n \in N$, if $n \stackrel{e_1}{\Rightarrow}$, $n \stackrel{e_2}{\Rightarrow}$, and $e_1 \neq e_2$, then $ln(n, e_1, e_2) \neq \varnothing$.*

Proof. By assumption, there is at least one node that fires both e_1 and e_2 and is reachable from n, i.e., n itself.

If n is the only node from which e_1 and e_2 are fireable, $n \stackrel{e_1}{\Rightarrow} n_1$, and $n \stackrel{e_2}{\Rightarrow} n_2$, then $ln(n, e_1, e_2) = \{(n_1, n_2)\}$. Note that the results also holds if $n \Rightarrow n$ since we have $\neg(n \prec n)$, by definition of $\prec$. If n is not the only node from which e_1 and e_2 are fireable, then there are two cases:

1. None of these nodes is reachable from n, in which case the result is the same as above, i.e., $ln(n, e_1, e_2) = \{(n_1, n_2)\}$.
2. Let $N_0 \stackrel{\text{def}}{=} \{n' \mid n \Rightarrow^* n' \wedge n' \stackrel{e_1}{\Rightarrow} \wedge n' \stackrel{e_1}{\Rightarrow}\}$, note that $n \in N_0$, and, by contradiction, assume $ln(n, e_1, e_2) = \varnothing$. Hence, none of the nodes in N_0 satisfy the condition, i.e.,

   $$\forall n' \in N_0 : \exists n'' \in N_0 : n' \Rightarrow n'' \wedge n' \prec n''$$

   Since the number of nodes in $TS(S)$ is finite, we must have a cycle in N_0.

By Lemmas D.3 and D.4, we know that $\prec$ forms a strict partial order on the nodes of each simple cycle in N_0. Thus, for each simple cycle consisting of nodes $N_0' \subseteq N_0$, each top element $n' \in N_0'$ satisfies the condition:

$$\forall n'' \in N_0' : \; n' \Rightarrow n'' \implies \neg(n' \prec n'')$$

Let $N_0'' \subseteq N_0$ be the set of all top elements of each simple cycle in N_0, none of the nodes in N_0' satisfy the condition only if

$$\forall n' \in N_0'' : \; \exists n'' \in N_0'' : \; n' \Rightarrow n'' \wedge n' \prec n''$$

We show that given $n_i, n_j \in N_0''$ such that $n_i \Rightarrow n_j$ and $n_i \prec n_j$, we cannot have $n_j \prec n_i$, i.e., there is no $\prec$-cycle between the nodes.

(a) If $\neg(n_j \Rightarrow^* n_i)$, we cannot have $n_j \prec n_i$ (and there is no cycle between the two nodes),

(b) If there is a simple cycle between the two nodes, then either the nodes are $\prec$-incomparable (i.e., we have a contradiction) or $n_i \prec n_j \wedge \neg(n_j \prec n_i)$, cf. Lemma D.4.

(c) If there is (only) a non-simple cycle such that $n_i \Rightarrow^* n_j$, $n_j \Rightarrow^* n_i$, then, each path between n_i and n_j must go through n_3 such that $n_3 \neq n_j$ and $n_3 \neq n_i$, which contradicts $n' \Rightarrow n''$.

Given that there cannot be a $\prec$-cycle amongst the nodes in N_0'', the function must return at least one element (i.e., at least one top element satisfies the condition). □

Lemma D.6. *Let $S = (M_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$ be a GMC system and $TS(S) = (N, n_0, \hat{E}, \Rightarrow)$. For all $n \in N$, if $n \overset{e_1}{\Rightarrow} n_1$ and $n \overset{e_2}{\Rightarrow} n_2$, then either*

1. *$n_1 \overset{e_2}{\Rightarrow} n'$ and $n_2 \overset{e_1}{\Rightarrow} n'$, for some $n' \in N$, or*
2. *$\mathtt{snd}(e_1) = \mathtt{snd}(e_2)$*

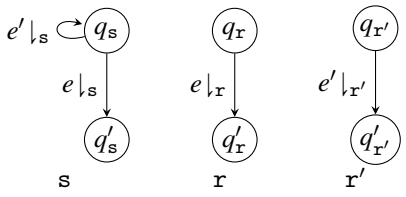
Proof. Direct from Lemma D.5 and Def. 3.5. □

Lemma D.7. *Let $S = (M_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$ be a GMC system and $TS(S) = (N, n_0, \hat{E}, \Rightarrow)$. For all $n \neq n' \in N$, if $n \overset{e}{\Rightarrow} n_1$, $n' \overset{e}{\Rightarrow} n_2$, and $n \overset{e'}{\Rightarrow} n'$ then, $n_1 \overset{e'}{\Rightarrow} n_2$.*

Proof. There are four cases depending on the machines involved in e and e', and whether the event fired from n is the same as the event fired from n' in $\hat{\delta}$.

1. If $\mathtt{id}(e) \cap \mathtt{id}(e') = \varnothing$, i.e., the events are independent, the result follows by definition of $TS(S)$: we have $n_1 \overset{e'}{\Rightarrow} n_2$.
2. If $\mathtt{id}(e) = \mathtt{id}(e')$ and $(n, e, n_1), (n', e, n_2) \in \hat{\delta}$, then we have $\forall \mathtt{p} \in \mathtt{id}(e) : \; n[\mathtt{p}] = n'[\mathtt{p}]$, which contradicts the fact that $n \neq n'$.

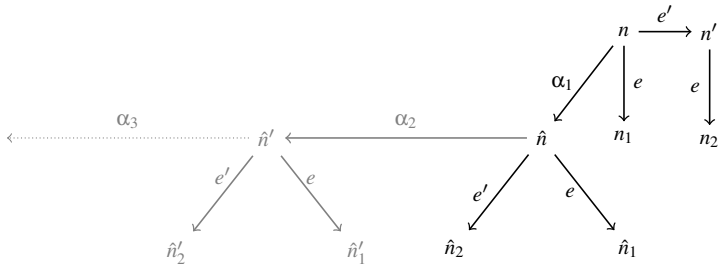
3. If $\mathtt{id}(e) \cap \mathtt{id}(e') \neq \varnothing$ and $(n, e, n_1) \notin \hat{\delta}$ or $(n', e, n_2) \notin \hat{\delta}$, then we must have, without loss of generality, $(n, e, n_1), (n', e'', n_2) \in \hat{\delta}$, such that $e \bowtie e''$ and $e \neq e''$. By definition of $\bowtie$, we must have $e = (q_{\mathtt{s}}, q_{\mathtt{r}}, \mathtt{s} \to \mathtt{r}{:}a)$ and $e'' = (q''_{\mathtt{s}}, q''_{\mathtt{r}}, \mathtt{s} \to \mathtt{r}{:}a)$, with $q_{\mathtt{s}} \neq q'_{\mathtt{s}}$ or $q_{\mathtt{r}} \neq q'_{\mathtt{r}}$. For each $\mathtt{p} \in \mathtt{id}(e) \cap \mathtt{id}(e')$, we have $(n[\mathtt{p}], e'\!\downarrow_{\mathtt{p}}, n'[\mathtt{p}]) \in \delta_{\mathtt{p}}$ by assumption and, by definition of $\blacklozenge$, we must also have $(n_1[\mathtt{p}], e'\!\downarrow_{\mathtt{p}}, n_2[\mathtt{p}]) \in \delta_{\mathtt{p}}$. Thus, if $\mathtt{id}(e) = \mathtt{id}(e')$, we have the result immediately. If $\mathtt{id}(e) \neq \mathtt{id}(e')$, then the machine not involved in e is still able to interact with $\mathtt{p}$ so to fire e', and we have the result.
4. If $\mathtt{id}(e) \cap \mathtt{id}(e') \neq \varnothing$, $\mathtt{id}(e) \neq \mathtt{id}(e')$, and $(n, e, n_1), (n', e, n_2) \in \hat{\delta}$, then, it must be the case that one of the machine has a self-loop, so that we have three machines of the form:



If there is $(q'_{\mathtt{s}}, e'\!\downarrow_{\mathtt{s}}, q'_{\mathtt{s}}) \in \delta_{\mathtt{s}}$ then we have the result immediately (as above), otherwise there are two cases, depending on whether $(n_1, n') \in ln(n, e, e')$.

(a) If $(n_1, n') \in ln(n, e, e')$, then either the two branches commute, and we have the result; or case (2) of Def. 3.5, must hold. However, in both branches, machine $\mathtt{r}$ is able to execute $e\!\downarrow_{\mathtt{r}}$ as a first action (since $\mathtt{r} \notin \mathtt{id}(e')$ by assumption). This means that the two branches must merge, i.e., once in state $q'_{\mathtt{s}}$, $\mathtt{s}$ must be able to interact with $\mathtt{r}'$ such that $\mathtt{r}'$ reaches $q'_{\mathtt{r}'}$. In addition, by representability, the self-loop at $\mathtt{s}$ must appear in $TS(S)$, which means there must be two branches in $TS(S)$, one that leads to a configuration where machines $\mathtt{s}$, $\mathtt{r}$, and $\mathtt{r}'$ are in states $q_{\mathtt{s}}$, $q_{\mathtt{r}}$, and $q_{\mathtt{r}'}$, respectively (where the self-loop is not represented); and one branch that leads to a configuration where the self-loop appears. This contradicts case (2) of Def. 3.5 since $\mathtt{r}$ would have the same first actions in both branches, while its behaviour must be different in both branches (i.e., the self-loop appears in one but not in the other).

(b) If $(n_1, n') \notin ln(n, e, e')$, we must have the following situation (by definition of the last node function):



Indeed, since $(n_1, n') \notin ln(n, e, e')$, there must be a successor of n that fires both e and e', i.e, $\hat{n}$ in the diagram above. Note that by, Lemma D.2, $\hat{n}$ cannot be a target of e or e'.

i. If $(\hat{n}_1, \hat{n}_2) \in ln(\hat{n}, e, e')$. We show that e must also be fireable from $\hat{n}_2$, by contradiction. Since $e'\downarrow_{\mathtt{s}}$ is a self-loop in $\mathtt{s}$ (and the machines are deterministic), $\mathtt{s}$ is still able to fire $e\downarrow_{\mathtt{s}}$ after $e'\downarrow_{\mathtt{s}}$; and since $\mathtt{r} \notin \mathtt{id}(e')$, the only way e would not be fireable from $\hat{n}_2$ is if $\mathtt{r} \in \mathtt{id}(\alpha_1)$ and $\forall q \in Q_{\mathtt{r}} : \ (\hat{n}[\mathtt{r}], e\downarrow_{\mathtt{r}}, q) \notin \delta_{\mathtt{r}}$), which contradicts the fact that $\hat{n} \overset{e}{\Rightarrow}$. We can now repeat the argument with $\alpha_1 = e'$, $\hat{n} = n'$, and $\hat{n}_1 = n_2$.

ii. If $(\hat{n}_1, \hat{n}_2) \notin ln(\hat{n}, e, e')$, then we can repeat the argument (cf. faded part of the picture) until we reach a pair of nodes that is in $ln(\_, e, e')$. We know that such a pair exists by Lemma D.5. □

Definition D.1. *Let $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$, $TS(S) = (N, n_0, \hat{E}, \Rightarrow)$, $\mathtt{p} \in \mathcal{P}$, $q \in Q_{\mathtt{p}}$, and $(q, \ell_1, q_1), (q, \ell_2, q_2) \in \delta_{\mathtt{p}}$, with $\ell_1 \neq \ell_2$, we write*

$$(q, \ell_1, q_1) \smile (q, \ell_2, q_2)$$

iff there exists $n \in N$ such that:

1. $n[\mathtt{p}] = q$
2. $n \overset{e_1}{\Rightarrow} n_1$, $n \overset{e_1}{\Rightarrow} n_2$,
3. $(n_1, n_2) \in ln(n, e_1, e_2)$
4. $\{\ell_i\} = hd\left(\left\{e_i\downarrow_{\mathtt{p}} \cdot \varphi \;\middle|\; \varphi \in LT(S, n_i, \mathtt{p})\right\}\right)$, *for* $i \in \{1, 2\}$.

We write $(q, \ell_1, q_1) \asymp (q, \ell_2, q_2)$ iff either (i) $(q, \ell_1, q_1) \smile (q, \ell_2, q_2)$; or (ii) there is $(q, \ell', q') \in \delta_{\mathtt{p}}$ such that $(q, \ell_1, q_1) \smile (q, \ell', q')$ and $(q, \ell', q') \asymp (q, \ell_2, q_2)$.

Lemma D.8 (Verified branches). *Let $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ be a GMC system and $TS(S) = (N, n_0, \hat{E}, \Rightarrow)$. For all $\mathtt{p} \in \mathcal{P}$ and for all $q \in Q_{\mathtt{p}}$, if $(q, \ell_1, q_1), (q, \ell_2, q_2) \in \delta_{\mathtt{p}}$ (with $\ell_1 \neq \ell_2$), then $(q, \ell_1, q_1) \asymp (q, \ell_2, q_2)$.*

Proof. First, we show the following:

$$\forall (q, \ell_1, q_1) \in \delta_{\mathtt{p}}, \ \exists (q, \ell_2, q_2) \in \delta_{\mathtt{p}} : \ (q, \ell_1, q_1) \smile (q, \ell_2, q_2) \tag{1}$$

By Lemma D.1, for each $(q, \ell_2, q_2) \in \delta_{\mathtt{p}}$ there is n such that $n[\mathtt{p}] = q$,

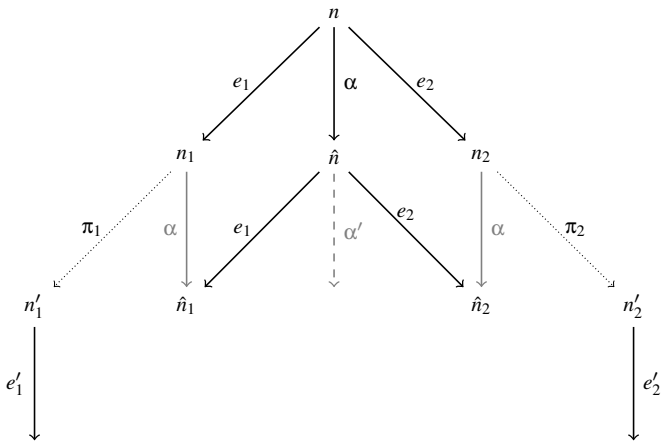
$$n \overset{\pi_1}{\Rightarrow} n_1 \overset{e_1}{\Rightarrow} \text{ with } \pi_1\downarrow_{\mathtt{p}} = \varepsilon \wedge e_1\downarrow_{\mathtt{p}} = \ell_1 \quad \text{and} \quad n \overset{\pi_2}{\Rightarrow} n_2 \overset{e_2}{\Rightarrow} \text{ with } \pi_2\downarrow_{\mathtt{p}} = \varepsilon \wedge e_2\downarrow_{\mathtt{p}} = \ell_2 \tag{2}$$

Choose n, ℓ_2, π_1, and π_2 such that π_1 is the smallest, i.e., there is no other node n' such (2) holds and π_1 is strictly smaller, for some ℓ_2. In other words, n is the last node from which $\mathtt{p}$ can "choose" to fire either ℓ_1 or another action (ℓ_2). Let $k = |hd(LT(S, n, \mathtt{p}))|$ and recall that $\ell_1, \ell_2 \in hd(LT(S, n, \mathtt{p}))$ by assumption.

We first show the result for $k = 2$.

1. If n is such that $n \overset{e_1}{\Rightarrow}$ and $n \overset{e_2}{\Rightarrow}$, with $e_i\downarrow_{\mathtt{p}} = \ell_i$ ($i \in \{1, 2\}$), then, by Lemma D.5, there is $\hat{n}, \hat{n}_1, \hat{n}_2 \in N$ such that $\hat{n} \overset{e_1}{\Rightarrow} \hat{n}_1$ and $\hat{n} \overset{e_2}{\Rightarrow} \hat{n}_2$, with $(\hat{n}_1, \hat{n}_2) \in ln(\hat{n}, e_1, e_2)$. Thus, we have $(q, \ell_1, q_1) \smile (q, \ell_2, q_2)$.

2. If n is such that $n \stackrel{e_1}{\Rightarrow}$, $n \stackrel{e_2}{\Rightarrow}$, with $e_i\!\downarrow_{\mathtt{p}} \neq \ell_i$ ($i \in \{1,2\}$), and $(n_1,n_2) \in ln(n,e_1,e_2)$, then we have $(q,\ell_1,q_1) \smile (q,\ell_2,q_2)$, since n is the latest node and $k=2$.
3. If n is such that $n \stackrel{e_1}{\Rightarrow}$, $n \stackrel{e_2}{\Rightarrow}$, with $e_i\!\downarrow_{\mathtt{p}} \neq \ell_i$ ($i \in \{1,2\}$), and $(n_1,n_2) \notin ln(n,e_1,e_2)$, we must have the following situation, where $e_i'\!\downarrow_{\mathtt{p}} = \ell_i$,



Since, $(n_1,n_2) \notin ln(n,e_1,e_2)$, it must be the case that there is a successor of n that is able to fire both e_1 and e_2, i.e., $\hat{n}$ here. Note that by, Lemma D.2, $\hat{n}$ cannot be a target of e_1 or e_2. In addition, by Lemma D.7, there must be an event α between both n_i and $\hat{n}_i$ ($i \in \{1,2\}$), cf. faded α-labelled edges in the diagram above.
By assumption ($k=2$), we have $\{\ell_i\} = hd\,(LT(S,n_i,\mathtt{p}))$ (otherwise, n would not be the latest node); thus $\ell_2 \neq \alpha\!\downarrow_{\mathtt{p}} \neq \ell_1$.
For $i \in \{1,2\}$, we reason as follows:
   - If case (1) of Def. 3.5 applies to the branching at n_i, then e_i' can also be fired from $\hat{n}_i$, and we have the result.
   - If case (2) of Def. 3.5 applies to the branching at n_i, then either (i) machine $\mathtt{p}$ must execute an action different from ℓ_i in the branch corresponding to π_i. However, this contradicts the fact that ℓ_2 cannot appear in $\pi_1'\!\downarrow_{\mathtt{p}}$. Or, ($ii$) there is merging node n'', reachable from both n_1 and n_2, and two paths (from n'' to n_i) such that machine $\mathtt{p}$ behaves identically on both; which contradicts our assumption that n is the last node such that ℓ_1 and ℓ_2 are fireable.

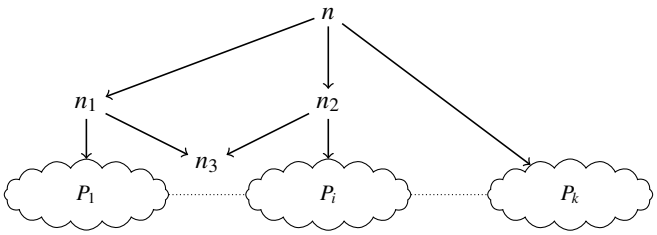
   Since ℓ_1 and ℓ_2 are the only actions fireable from q, we conclude that each action is executed from $\hat{n}_1$ and $\hat{n}_2$, respectively, and we obtain the result.
4. If $(\hat{n}_1,\hat{n}_2) \notin ln(\hat{n},e_1,e_2)$, then we repeat the same argument until we find a pair that is included in $ln(\_,e_1,e_2)$, and we know there is such a pair by Lemma D.5; cf. faded α' edge in the diagram above (or leads to a contradiction)

If $k > 2$, then we reason similarly, with k events e_i and nodes n_i, where each path cannot execute any actions from $\mathtt{p}$ without leading to a contradiction. We then conclude that case (1) of Def. 3.5 must apply for each path and we finally reach a pair of nodes in $ln(\hat{n},e_1,e_2)$, as above.

Second, having established (1), we show that

$$\forall (q, \ell_1, q_1) \neq (q, \ell_2, q_2) \in \delta_{\mathtt{p}} : \ (q, \ell_1, q_1) \asymp (q, \ell_2, q_2) \tag{3}$$

We show this by contradiction, if (3) does not hold, we must have partitions of outgoing transitions from q such that the result hold within each partition, but not across two partitions, i.e., we have a situation of the form below, where n is the last node such that all transitions of $\mathtt{p}$ from q are taken.



However, between two partitions either (*i*) case (2) of Def. 3.5 applies, i.e., the branches do not commute nor merge (cf. right-hand side of the diagram above) and thus, there must be a node n and two events e_1 and e_2 which will be checked for, by Lemma D.5; or (*ii*) case (1) of Def. 3.5 applies (cf. left-hand side of the diagram), in which case there is two other nodes n_1 and n_2, which commute (or merge, cf., item (*ii*) in (2a) of Def. 3.5) to a third node, n_3, and we can inductively repeat the argument in (*i*) where n_1 and n_2 replace n. Note that it must be the case that $\mathtt{p}$ executes some actions from nodes n_3, thus linking two partitions together. □

D.4 Proof of Theorem 3.1 (Soundness)

Theorem 3.1 (Soundness). *If S is GMC, then it is* safe*, i.e., free from orphan message, deadlock and unspecified reception configurations.*

Proof. Absence of orphan message configuration and unspecified reception configurations follow directly from Lemma D.12.

We show absence of deadlock configuration, by contradiction, using Lemma D.14. Assume there is a deadlock configuration $s \in RS(S)$ such that $s = (\vec{q}; \vec{\varepsilon})$, $n = \vec{q}$, and $(q_{\mathtt{r}}, \mathtt{sr}?a, q'_{\mathtt{r}}) \in \delta_{\mathtt{r}}$ for some $\mathtt{r} \in \mathcal{P}$. By Lemma D.14, we have $n_0 \Rightarrow^* n$, and since S is representable, there must be an execution in which $\mathtt{sr}?a$ is taken. Moreover, by Lemma D.12, there is no unspecified reception configuration, so it must be the case that there is a branching in $M_{\mathtt{r}}$ such that the machine is expecting a message a in one branch, but not in the other (reasoning in a similar fashion to the proof of Lemma D.12). The branching in $M_{\mathtt{r}}$ must be reflected in $TS(S)$ such that there is a branch going to n and another branch where the interaction on a takes place. However, in the n branch, the behaviour of $\mathtt{r}$ is empty (deadlock) and thus we obtain a contradiction with Def. 3.5 since $\mathtt{r}$ executes $\mathtt{sr}?a$ in one branch and does nothing (i.e., ε) in the other branch. □

Lemma D.9. *Let $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ be a GMC system and $TS(S) = (N, n_0, \hat{E}, \Rightarrow)$. For all $\mathtt{p} \in \mathcal{P}$ and for all $q \in Q_{\mathtt{p}}$, if $(q,\ell_1,q_1),(q,\ell_2,q_2) \in \delta_{\mathtt{p}}$ (with $\ell_1 \neq \ell_2$), and there is $n \in N$, $n[\mathtt{p}] = q$, such that $n \stackrel{e_i}{\Rightarrow}$ with $e_i\restriction_{\mathtt{p}} = \ell_i$, then $(q,\ell_1,q_1) \smile (q,\ell_2,q_2)$.*

Proof. Direct from Def. D.1 and Lemma D.5. □

Lemma D.10. *Let $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ be a GMC system and $TS(S) = (N, n_0, \hat{E}, \Rightarrow)$. For all $\mathtt{p} \in \mathcal{P}$ and for all $q \in Q_{\mathtt{p}}$, if $(q,\ell_1,q_1),(q,\ell_2,q_2) \in \delta_{\mathtt{p}}$ (with $\ell_1 \neq \ell_2$), such that ℓ_1 (resp. ℓ_2) is a send action (resp. receive) action and $(q,\ell_1,q_1) \smile (q,\ell_2,q_2)$, then there is $n \in N$, $n[\mathtt{p}] = q$, such that $n \stackrel{e_i}{\Rightarrow}$ with $e_i\restriction_{\mathtt{p}} = \ell_i$.*

Proof. By contradiction, assume there is no such $n \in N$. Then for all $n \stackrel{\pi_i}{\Rightarrow} n_i \stackrel{e_i}{\Rightarrow}$, with $\pi_i\restriction_{\mathtt{p}} = \varepsilon$ and $e_i\restriction_{\mathtt{p}} = \ell_i$, $|\pi_i| > 0$. Observe that such nodes exists since $(q,\ell_1,q_1) \smile (q,\ell_2,q_2)$. This implies that case (1) of Def. 3.5 cannot apply, and case (2) of Def. 3.5 cannot apply either (since there is no unique sender); which contradicts the fact that S is GMC. □

Lemma D.11 (Mixed Choice). *Let $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ be a GMC system, for all $\mathtt{p} \in \mathcal{P}$ and for all $q \in Q_{\mathtt{p}}$ such that $(q,\ell,q'),(q,\ell',q'') \in \delta_{\mathtt{p}}$ for some $q',q'' \in Q_{\mathtt{p}}$, if ℓ is a* send *action and ℓ' is a* receive *action, then there is $\hat{q} \in Q_{\mathtt{p}}$ such that $(q',\ell',\hat{q}) \in \delta_{\mathtt{p}}$ and $(q'',\ell,\hat{q}) \in \delta_{\mathtt{p}}$.*

Proof. By Lemma D.8, we must have $(q,\ell,q') \asymp (q,\ell',q'')$.

If $(q,\ell,q') \smile (q,\ell',q'')$, the result follows directly by Lemma D.10.

If $\neg((q,\ell,q') \smile (q,\ell',q''))$, then there must be a chain of transitions (q,ℓ_i,q_i) such that

$$(q,\ell_1,q_1) \smile \ldots \smile (q,\ell_k,q_k) \qquad k > 2 \tag{4}$$

where $\ell_1 = \ell$, $\ell_k = \ell'$, $q_1 = q'$, and $q_k = q''$. We show that for all $1 \leqslant i \neq j \leqslant k$: ℓ_i is a *send* action and ℓ_j is a *receive* action, then there is $\hat{q} \in Q_{\mathtt{p}}$ such that $(q_i,\ell_j,\hat{q}) \in \delta_{\mathtt{p}}$ and $(q_j,\ell_i,\hat{q}) \in \delta_{\mathtt{p}}$. We show the result by induction on the length k of the smallest chain of transitions for which (4) holds. Assume $k = 3$, then there are two cases

- If ℓ_2 is a send action, then it must commute with ℓ_3, by Lemma D.10 and Def. 3.5. By Lemma D.9, we must also have $(q,\ell_1,q_1) \smile (q,\ell_3,q_3)$, and we are done.
- If ℓ_2 is a receive action, then it must commute with ℓ_1, by Lemma D.10, and we reason as above.

The inductive case follows straightforwardly. Assume the result holds for $k > 3$ and let us show it holds for $k+1$. If ℓ_k is a send action, it must commute with ℓ_{k+1}, which implies that there is a smaller chain $\asymp$-linking ℓ_{k+1} with ℓ_1 (as above). If ℓ_k is a receive action, then it must commute with all the send actions ℓ_i such that $i < k$, thus there is a smaller chain $\asymp$-linking ℓ_{k+1} with a send action, and we are done by induction hypothesis. □

Lemma D.12. *Let $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ be a GMC system for all $s \in RS(S)$, if $s \xrightarrow{\mathtt{sr}!a} s_1 \xrightarrow{\varphi} s_2$, with $\mathtt{sr}?a \notin \varphi$, then $s_2 \to^* s_3 \xrightarrow{\mathtt{sr}?a} s_4$.*

Proof. By contradiction, assume there is $s \in RS(S)$ such that $s \xrightarrow{\mathtt{sr}!a} s'$ and for all φ such that $s' \xrightarrow{\varphi}$, we have $\mathtt{sr}?a \notin \varphi$. In addition, assume that s is the first configuration (from s_0) such that a sent message cannot be received.

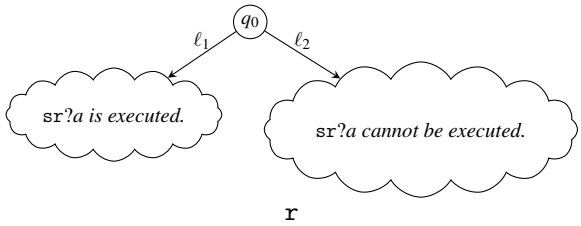
Assume the state of $\mathtt{s}$ in s is $q_{\mathtt{s}}$. Since S is representable, there must be $s'' \in RS(S)$, where the state of $\mathtt{s}$ in s'' is $q_{\mathtt{s}}$, such that $s' \xrightarrow{\mathtt{sr}!a} \xrightarrow{\mathtt{sr}?a}$, i.e., there is an execution where the message a sent by $\mathtt{s}$ is received by $\mathtt{r}$. If the message cannot be received by $\mathtt{r}$ from s, it means that either

1. $\mathtt{r}$ is unable to fire the action $\mathtt{sr}?a$ because it is expecting another message, i.e., it is blocked by a transition $\mathtt{pr}?b$, or
2. there is a branching in $M_{\mathtt{r}}$ such that in one of the branch a is received but not in the other.

Case (1) implies that there is another configuration *before* s where a message is not received, thus we only consider case (2).

Let us consider the last state $q \in Q_{\mathtt{r}}$ such that $(q,\ell_1,q_1),(q,\ell_2,q_2) \in \delta_{\mathtt{r}}$ $(\ell_1 \neq \ell_2)$ and there is no $\mathtt{sr}?a$ transitions fireable from q_1 (before any other transition receiving from $\mathtt{s}$) and the first transition receiving from $\mathtt{s}$ after q_2 is $\mathtt{sr}?a$. Observe that since q is the *last* state where such a branching occur, there cannot be $q' \in Q_{\mathtt{r}}$ such that $(q_1,\ell_2,q'),(q_2,\ell_1,q') \in \delta_{\mathtt{r}}$.

Essentially, we have the following situation:



where $\mathtt{r}$ *cannot execute* $\mathtt{sr}?a$ *once it has taken the transition* ℓ_2.

We have the following cases:

1. ℓ_1 is a send action and ℓ_2 is a receive action,
2. both ℓ_1 and ℓ_2 are send actions, or
3. both ℓ_1 and ℓ_2 are receive actions.

Following Lemma D.11, case (1) cannot happen since it would imply that the branches commute.

Observe that if there are more than two transitions outgoing q, we can partition them into two sets: (*i*) the transitions after which a is a received and (*ii*) the transitions after which a is not received. By Lemma D.8, there must be, at least, one transition (q,ℓ_2,q_2) in (*i*) and one transition (q,ℓ_1,q_1) in (*ii*) such that $(q,\ell_1,q_1) \smile (q,\ell_2,q_2)$. Hereafter, we only consider these two transitions. Case (2) of Def. 3.5 must apply to a branching node

n such that $n[\mathtt{p}] = q$ and the first action of $\mathtt{r}$ in one branch is ℓ_1 and the first action of $\mathtt{r}$ in the other branch is ℓ_2, by definition of $\smile$. Note that case (*ii*) of (2a) cannot apply for $\mathtt{r}$ since it would otherwise mean that there is a later branching in machine $\mathtt{r}$ such that it can "choose" whether to receive a or not; i.e., $\mathtt{r}$ must be *involved* in the choice.

Case (2). Then, $\mathtt{r}$ is the selector at node n. Following Def. 3.5, there are two subcases: Machine $\mathtt{s}$ executes a *receive* action before sending another action. This implies that there must be a branching in $\mathtt{s}$ *before* the transition $\mathtt{sr}!a$, which means that $\mathtt{sr}!a$ can only be executed once the ℓ_2-branch has been executed. Thus, $\mathtt{r}$ cannot make a "bad choice": the two branches are mutually exclusive in all possible executions.

Indeed, if $\mathtt{r}$ was able to make another choice, it would mean that it could choose between receiving ℓ_1 and ℓ_2, i.e., there is no $\lhd$-dependency between ℓ_1 and ℓ_2. More formally, assume we have $(q, \mathtt{s}_1\mathtt{r}?\ell_1, q_1), (q, \mathtt{s}_2\mathtt{r}?\ell_2, q_2) \in \delta_{\mathtt{r}}$. By representability, there must be a node $n \in N$ such that, for $i \in \{1,2\}$, $n \stackrel{\pi_i}{\Rightarrow} n_i \stackrel{e_i}{\Rightarrow}$ with $e_i\!\upharpoonright_{\mathtt{r}} = \mathtt{s}_i\mathtt{r}?\ell_i$, $\pi_i\!\upharpoonright_{\mathtt{r}} = \varepsilon$, and $n[\mathtt{r}] = q$. Also, by GMC and the fact that we assumed that the non-reception at $\mathtt{r}$ is the first one in the system, $\mathtt{r}$ must be able to receive both messages from $\mathtt{s}_1$ and $\mathtt{s}_2$ in each branch, i.e., we must have, for $i \neq j \in \{1,2\}$, $n_i \stackrel{e_i}{\Rightarrow}\stackrel{\pi_i}{\Rightarrow}\stackrel{e_j}{\Rightarrow}$ with $e_j\!\upharpoonright_{\mathtt{r}} = \mathtt{s}_j\mathtt{r}?\ell_j$. If $\mathtt{s}_1 = \mathtt{s}_2$, the two actions are always mutually exclusive (since the channels are message order preserving) and thus there is always a trivial $\lhd$-dependency. If $\mathtt{s}_1 \neq \mathtt{s}_2$, we must have a $\lhd$-chain between the two corresponding interactions in both branches. The shortest of such chains are as follows, with $i \neq j \in \{1,2\}$,

$$\mathtt{s}_i \to \mathtt{r} : \ell_i \lhd \mathtt{r} \to \mathtt{s}_j : a \lhd \mathtt{s}_j \to \mathtt{r} : \ell_j$$

which implies that $\mathtt{r}$ cannot choose between receiving either message in any asynchronous execution.

Case (3). This means that another machine selects the branch:

- If $\mathtt{s}$ is the selector, then there must be a branching in machine $\mathtt{s}$ such that $\mathtt{s}$ sends a different message to all participants involved in the choice (possibly through other participants), including $\mathtt{r}$. This implies that $\mathtt{r}$ must be aware of which branch $\mathtt{s}$ has chosen and thus cannot make a "bad choice" either.
- If $\mathtt{s}$ is not the selector, there must be another participant making the decision and we reason as in case (2) above. □

D.5 Proof of Reachability

The following lemma states the reachability property, which is important to apply Cortadella's algorithm.

Lemma D.13. *If S is GMC and $TS(S) = (N, n_0, \hat{E}, \Rightarrow)$, then $\forall n \in N :\ n_0 \Rightarrow^* n$.*

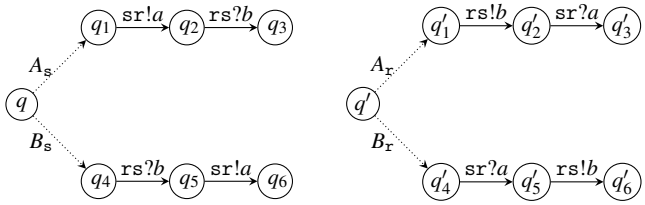
Proof. By contradiction, assume there is $(\vec{q}, \vec{\varepsilon}) \in RS_1(S)$ such that $n_0 \Rightarrow^* \vec{q}$ and $(\vec{q}', \vec{\varepsilon}) \in RS_1(S)$ such that $(\vec{q}', \vec{\varepsilon})$ is reachable by 1-bounded execution from $(\vec{q}, \vec{\varepsilon})$, but not by a synchronous execution. This implies that there is a dependency chain between the two configurations. The smallest such chain is:

$$(\vec{q}, \vec{\varepsilon}) \xrightarrow{\mathtt{sr}!a} \xrightarrow{\mathtt{rs}!b} \xrightarrow{\mathtt{rs}?b} \xrightarrow{\mathtt{sr}?a} (\vec{q}', \vec{\varepsilon})$$

i.e., it is not possible to swap actions $\mathtt{rs}!b$ and $\mathtt{sr}?a$, thus there is no synchronous counterpart for this execution. Such an execution implies that we have machines of the form:

$$\mathtt{s}:\ q_1 \xrightarrow{\mathtt{sr}!a} q_2 \xrightarrow{\mathtt{rs}?b} q_3 \qquad \mathtt{r}:\ q_1' \xrightarrow{\mathtt{rs}!b} q_2' \xrightarrow{\mathtt{sr}?a} q_3'$$

By representability, each execution must be represented in the $TS(S)$, thus, there must be two branches in each machine, so that both branches appear in $TS(S)$:



By assumption, branches $A_\mathtt{s}$ and $A_\mathtt{r}$ can be executed synchronously, since $n_0 \Rightarrow^* \vec{q}$, while, by representability, $A_\mathtt{s}$ and $B_\mathtt{r}$ (resp. $B_\mathtt{s}$ and $A_\mathtt{r}$) must be executable synchronously.

- If $q_1 = q_4$ or $q_1' = q_4'$, we obtain a contradiction with the assumption that $(\vec{q}', \vec{\varepsilon})$ is not reachable by a synchronous execution.
- Hence, if $q_1 \neq q_4$ and $q_1' \neq q_4'$, there are three branches in $TS(S)$: $(A_\mathtt{s}, A_\mathtt{r})$, $(A_\mathtt{s}, B_\mathtt{r})$, and $(B_\mathtt{s}, A_\mathtt{r})$. This implies that machine $\mathtt{s}$ (resp. $\mathtt{r}$) has the same first actions in branches $(A_\mathtt{s}, A_\mathtt{r})$ and $(A_\mathtt{s}, B_\mathtt{r})$ (resp. $(A_\mathtt{s}, A_\mathtt{r})$ and $(B_\mathtt{s}, A_\mathtt{r})$). Thus, either the branches commute, i.e., $q_1 = q_4$, $q_3 = q_6$, $q_1' = q_4'$, $q_3' = q_6'$, or there is a path in $\mathtt{s}$ (resp $\mathtt{r}$) such that both branches merge, either way, this contradicts the fact that $(\vec{q}', \vec{\varepsilon})$ is not reachable by a synchronous execution. If the branches do not commute nor merge, then the system is not GMC, which contradicts our assumptions.

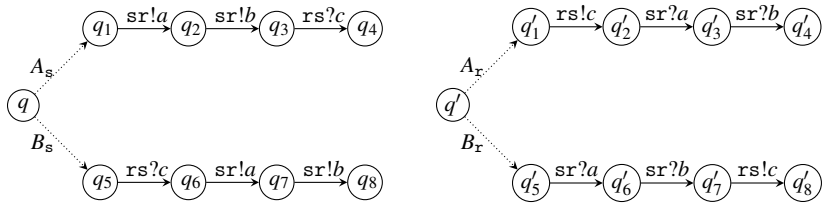
□

Lemma D.14. *If S is GMC and $TS(S) = (N, n_0, \hat{E}, \Rightarrow)$, then $\forall (\vec{q}, \vec{\varepsilon}) \in RS(S)\ :\ n_0 \Rightarrow^* \vec{q}$.*

Proof. The proof is similar to the proof of Lemma D.13. By contradiction, assume there is $s \in RS_2(S) \backslash RS_1(S)$ such that $s = (\vec{q}, \vec{\varepsilon})$ and $\neg(n_0 \Rightarrow^* \vec{q})$. Then, there should be an execution of the form

$$s_0 \xrightarrow{\varphi_1} s_1 \xrightarrow{\mathtt{sr}!a} s_2 \xrightarrow{\varphi_2} s_3 \xrightarrow{\mathtt{sr}!b} s_4 \xrightarrow{\varphi_3} s_5 \xrightarrow{\mathtt{sr}?a} s$$

with $s_1 \in RS_1(S)$, stable, $s_3 \in RS_1(S)$ and $s_4 \in RS_2(S) \backslash RS_1(S)$, such that there is a dependency chain in φ_3 between $\mathtt{sr}!b$ and $\mathtt{sr}?a$ (otherwise s would be reachable by a 1-bounded execution). The smallest such chain is of the form $\mathtt{rs}!c \cdot \mathtt{rs}?c$, thus let $\varphi_2 = \mathtt{rs}!c \cdot \mathtt{rs}?c$. By representability, as in the proof of the lemma above, there must be two branches in each machine so that each trace can be executed synchronously, i.e.,



Branches $A_{\mathtt{s}}$ and $A_{\mathtt{r}}$ can be executed synchronously, since s_1 is stable and $s_1 \in RS_1(S)$ (cf. Lemma D.13) By representability, $A_{\mathtt{s}}$ and $B_{\mathtt{r}}$ (resp. $B_{\mathtt{s}}$ and $A_{\mathtt{r}}$) must be executable synchronously. Hence, if $q_1 \neq q_5$ and $q'_1 \neq q'_5$, there are three branches in $TS(S)$: $(A_{\mathtt{s}},A_{\mathtt{r}})$, $(A_{\mathtt{s}},B_{\mathtt{r}})$, and $(B_{\mathtt{s}},A_{\mathtt{r}})$. This implies that machine $\mathtt{s}$ (resp. $\mathtt{r}$) has the same first actions in branches $(A_{\mathtt{s}},A_{\mathtt{r}})$ and $(A_{\mathtt{s}},B_{\mathtt{r}})$ (resp. $(A_{\mathtt{s}},B_{\mathtt{r}})$ and $(B_{\mathtt{s}},A_{\mathtt{r}})$). Thus, either the branches commute, i.e., $q_1 = q_5$, $q_4 = q_8$, $q'_1 = q'_5$, $q'_4 = q'_8$, or they merge, which contradicts the fact that $(\vec{q}',\vec{\varepsilon})$ is not reachable by a synchronous execution; or the system is not GMC, which contradicts our assumptions. □

D.6 Complexity Analysis: Proofs of Propositions 3.1 and 3.2

In this section, we fix $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$, $TS(S) = (N, n_0, \hat{E}, \Rightarrow)$, and we let $\delta_{\mathtt{M}}$ be the biggest set of transitions of all the machines.

We first describe the maximum size of the basic constructions.

- $|N| = \prod_{\mathtt{p}\in\mathcal{P}} \left|Q_{\mathtt{p}}\right|$ since each node in N consists of a $|\mathcal{P}|$ vector of states, each in $Q_{\mathtt{p}}$.
- $\left|\hat{\delta}\right| = |E| = \left|2\mathbb{A} \times |N|^2\right|$ since there can be at most $2\mathbb{A}$ transitions between two nodes in N, indeed:
  - the local state components of each event is a determined by the node it is fired from,
  - by construction, given two nodes, all the transitions between these nodes must involve the same machines $\mathtt{s}$ and $\mathtt{r}$,
  - the machines are deterministic, thus they can only send a same message once per state, and
  - each message may be sent by each one of the two machines.
- $|TS(S)| = |N| + \left|\hat{\delta}\right|$
- $\left|\hat{E}\right| = |E|$, since, in the worst case, $\bowtie$ is the identity relation.
- Relation $\prec$ can be a computed via a breadth-first traversal of $TS(S)$, i.e., in $O(|N| + |\Rightarrow|)$ time, and its maximal size if $|N|^2$. Note that it is not necessary to compute the transitive closure of $\prec$ since, to check whether a node is a "last node", it suffices to check its (direct) neighbours.

Proposition D.1. *The $\bowtie$-relation is computable in $O\left(\sum_{\mathtt{p}\in\mathcal{P}}\left(\left|\delta_{\mathtt{p}}\right|^6\right) + \left(|E|^2 \times (2 \times |\delta_{\mathtt{M}}|^2)\right)\right)$ time.*

Proof. First, we observe that computing the $\blacklozenge$-relation for each machine can be done in time $\sum_{\mathtt{p}\in\mathcal{P}}\left(\left|\delta_{\mathtt{p}}\right|^6\right)$ since the size of the non-transitive version of $\blacklozenge$ is $\left|\delta_{\mathtt{p}}^2\right|$ and computing the transitive closure of a binary relation has a cubic complexity. Then, to compute the $\bowtie$-relation, for each pair of events in E, we have to look up in two sets of $\blacklozenge$ relations. □

Proposition 3.1. *Given a system $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$, checking whether S satisfies the representability condition is computable in $O(\sum_{\mathtt{p}\in\mathcal{P}} 2^{|N|+|Q_{\mathtt{p}}|})$ time, with $|N| = \prod_{\mathtt{p}\in\mathcal{P}}\left|Q_{\mathtt{p}}\right|$.*

Proof. The first part of the definition requires, for each machine, to (*i*) compute its projection from $TS(S)$ (which depends on the size of N) and (*ii*) to check that this projection is *language equivalent* with the original machine. The second part requires for each machine (sum over $\mathcal{P}$), for each state ($Q_{\mathtt{p}}$ factor), and for each transition from that state ($\delta_{\mathtt{p}}$ factor) to look for a node in $TS(S)$ (N factor) such that this transition is reflected in $TS(S)$ ($\hat{\delta}$ factor). Since for each local state, we are only interested in the nodes where this local state appears, we divide by $\left|Q_{\mathtt{p}}\right|$. Hence, we have that the time complexity of the representability condition is:

$$
\begin{aligned}
&O\left(\sum_{\mathtt{p}\in\mathcal{P}}\left(\left|\hat{\delta}\right| + 2^{|N|+|Q_{\mathtt{p}}|}\right) + \sum_{\mathtt{p}\in\mathcal{P}}\left(\left|Q_{\mathtt{p}}\times\delta_{\mathtt{p}}\times N\right| \times \frac{\left|\hat{\delta}\right|}{|Q_{\mathtt{p}}|}\right)\right)\\
=&O\left(\sum_{\mathtt{p}\in\mathcal{P}}\left(\left|\hat{\delta}\right| + 2^{|N|+|Q_{\mathtt{p}}|}\right) + \sum_{\mathtt{p}\in\mathcal{P}}\left(\left|\delta_{\mathtt{p}}\times N\times\hat{\delta}\right|\right)\right)\\
=&O\left(\sum_{\mathtt{p}\in\mathcal{P}}\left(2^{|N|+|Q_{\mathtt{p}}|}\right)\right)
\end{aligned}
$$

□

Proposition 3.2. *Given a system $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$, checking whether S satisfies the branching property is computable in time $O\left(|\Rightarrow|^2 \times |\Rightarrow|! \times \sum_{\mathtt{r}\in\mathcal{P}}\left(|\delta_{\mathtt{r}}|^2\right)\right)$.*

Proof. The check of part (1) of Def. 3.5 is $O\left(|\Rightarrow|^2 \times |N|\right)$, so we consider the most complex case, namely part (2) of Def. 3.5, which is the sum of the complexity of conditions conditions (2a), (2b), and (2c) of Def. 3.5.

For each branch with events $e_1 \neq e_2$ and each participant $\mathtt{p}\in\mathcal{P}$, we have to compute the sets $L^i_{\mathtt{p}}$. This also allows to compute the last nodes and can be done considering simple paths only, and is easily computed with a breadth-first visit of $TS(S)$, and therefore this is done in polynomial time in $|TS(S)|$.

Once this is done it is easy to see that the check of part (2b) of Def. 3.5 is polynomial in $|\mathcal{P}\cup Act|$.

The most complex part of the computation is to check conditions (2b)(*ii*) and (2c) because it requires to check the paths in $TS(S)$ and the enumeration of all paths is computed in $O(|\Rightarrow|!)$.

Finally, the check of condition (2b)(*ii*) can be done in $O(\mathsf{size}(TS(S)) \times |\Rightarrow|!)$ while (2c) is checked in $O\left(|\Rightarrow|^2 \times |\Rightarrow|! \times \sum_{\mathtt{r}\in\mathcal{P}}\left(|\delta_{\mathtt{r}}|^2\right)\right)$ time. □

E Appendix: Proofs of Section 4

E.1 Equivalences

Lemma A.1. *If T is the reachability graph of the Petri net $\mathbb{N}$ obtained from $TS(S)$ via the algorithm in [13], and T' is the reachability graph of the Petri net obtained after applying Transf. A.1, then $T \approx T'$.*

Proof. Trivial since $\mathbb{N}$ is safe and the only added transition is labelled by ε. □

Lemma A.2. *If T (resp. T') is the reachability graph of the Petri net $\mathbb{N}$ obtained after Transf. A.1 (resp. Transf. A.2), then $T \approx T'$.*

Proof. Follows from the fact that $\mathbb{N}$ is safe and the only added transitions are labelled by ε. □

Theorem 4.1 (Completeness). *Let $S = (M_\mathtt{p})_{\mathtt{p}\in\mathcal{P}}$ be GMC and $TS(S) = (N, n_0, \hat{E}, \Rightarrow)$. If $TS(S)$ is such that $\forall n,n' \in N : \ n \Rightarrow n' \implies n \neq n'$, then the projection of the global graph G synthesised from S is such that S is isomorphic to $(G\downarrow_\mathtt{p})_{\mathtt{p}\in\mathcal{P}}$.*

Proof. The proof follows from (1) the assumption that $TS(S)$ is self-loop free, (2) Lemmas A.1 and A.2, and (3) the fact that transformation to (pre-) global graph preserves the structure and labels of the joined net. □

E.2 Complexity Analysis of the Transformations

Proposition A.1. *Transf A.1 is computable in linear time in the size of m_0.*

Proof. Trivial. □

Proposition A.2. *Transf. A.2 is computable in polynomial time in the size of $\mathbb{N}$.*

Proof. The algorithm for part (1) of the transformation works as follows: (1) compute the preset of each place ($|F|$), (2) sort a table of pairs (preset, place) by preset, e.g., a lexicographic order on sets of transitions ($|P|\,(\log|P|)$), (3) go through the table and apply the transformation on each set of places which have the same preset ($|P|$). Observe that, once a set of places with the same preset has been identified, the transformation can be done in linear time. The algorithm for part (2) works similarly. □

Proposition A.3. *Transf. A.3 is computable in polynomial time in the size of $\mathbb{N}$.*

Proof. One has to iterate on each element in $P \cup T \cup F$ and the composition of graphs is polynomial in the size of the two graphs (number of vertices). □

Proposition A.4. *Transf. A.4 is computable in polynomial time in the size of $\mathbb{N}$.*

Proof. The transformation can be done by iterating on the arcs of the pre-global graph, so to find matching arc ($|A|^2$). □

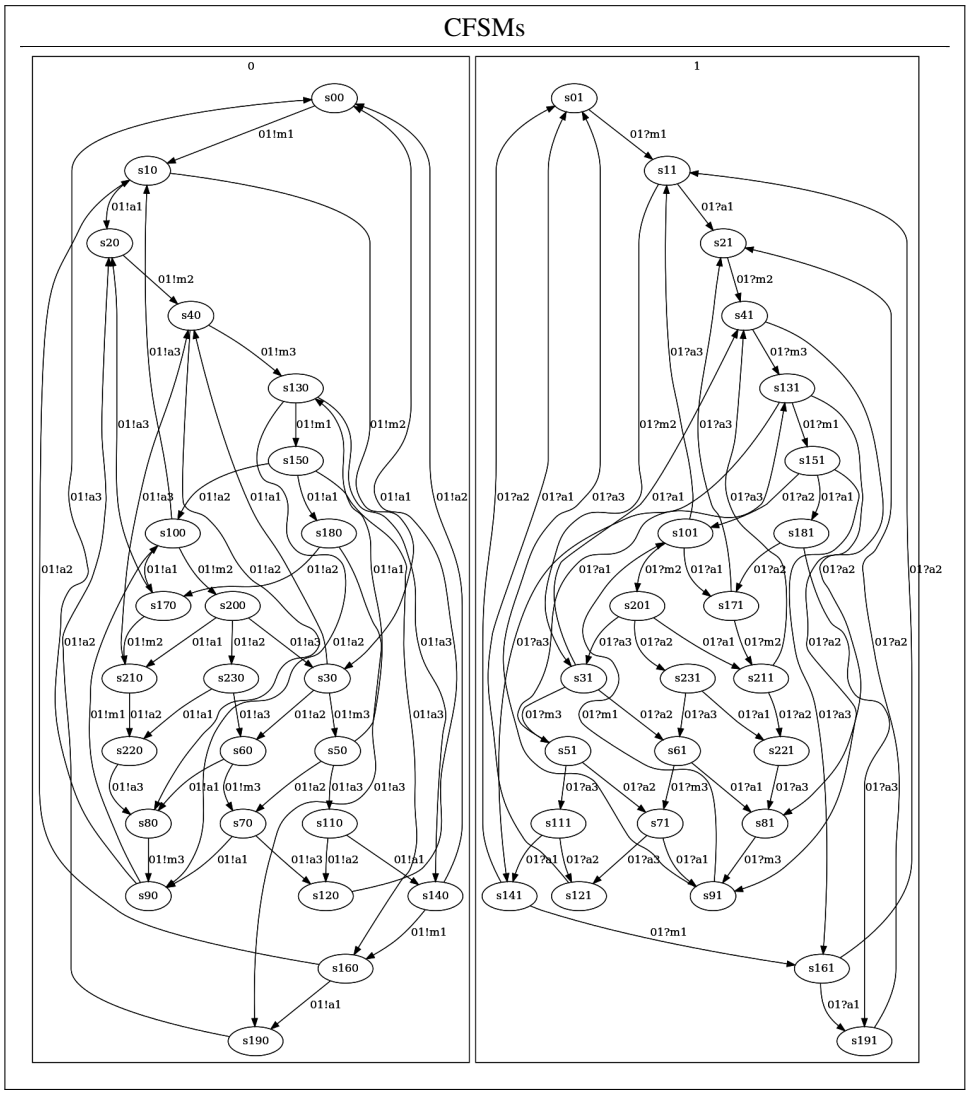
F Appendix: Benchmark Examples from Section 5

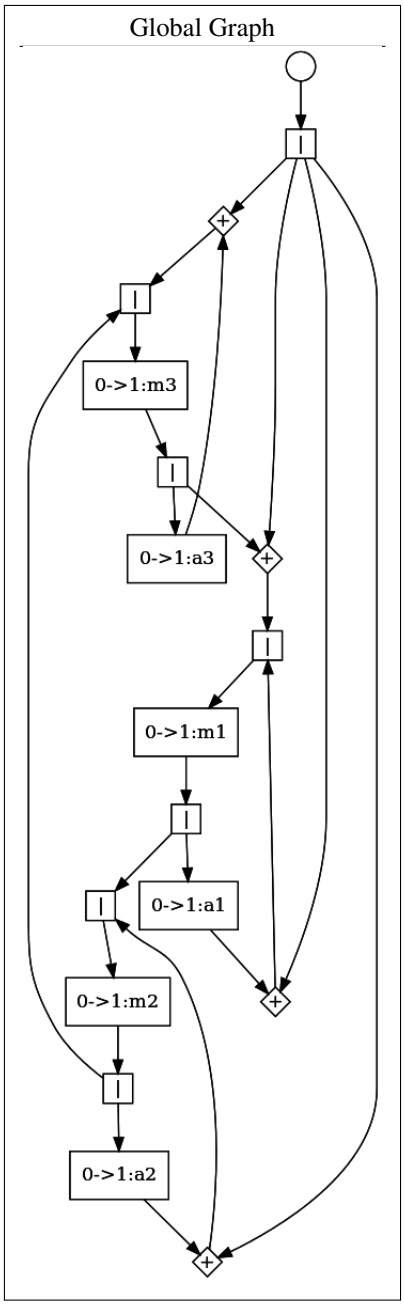
This section lists the most interesting synthesis examples selected from the benchmark given in Section 5. All benchmark examples are available in the online appendix[4] and textual representations of these are also available in [17] (`gmc-synthesis/tests/benchmark/gmc` directory). Other GMC and non-GMC protocols are also available online.[5]

F.1 Alternating 3-bit protocol

This protocol, adapted from [14], models a protocol where machine `0` repeatedly sends to machine `1` alternating messages m_1, m_2, and m_3 but will always concurrently wait for the acknowledgement a_i before sending m_i.

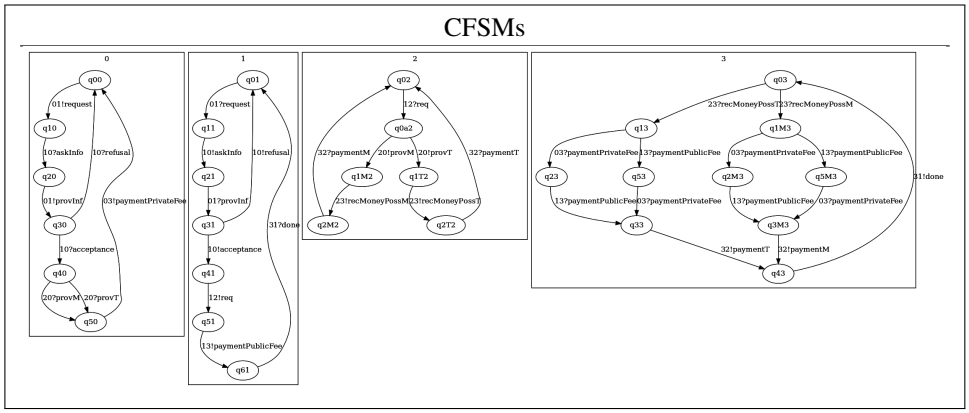
[4] `http://www.doc.ic.ac.uk/~jlange/benchprotocols.pdf`
[5] `http://www.doc.ic.ac.uk/~jlange/demo.tar.gz`

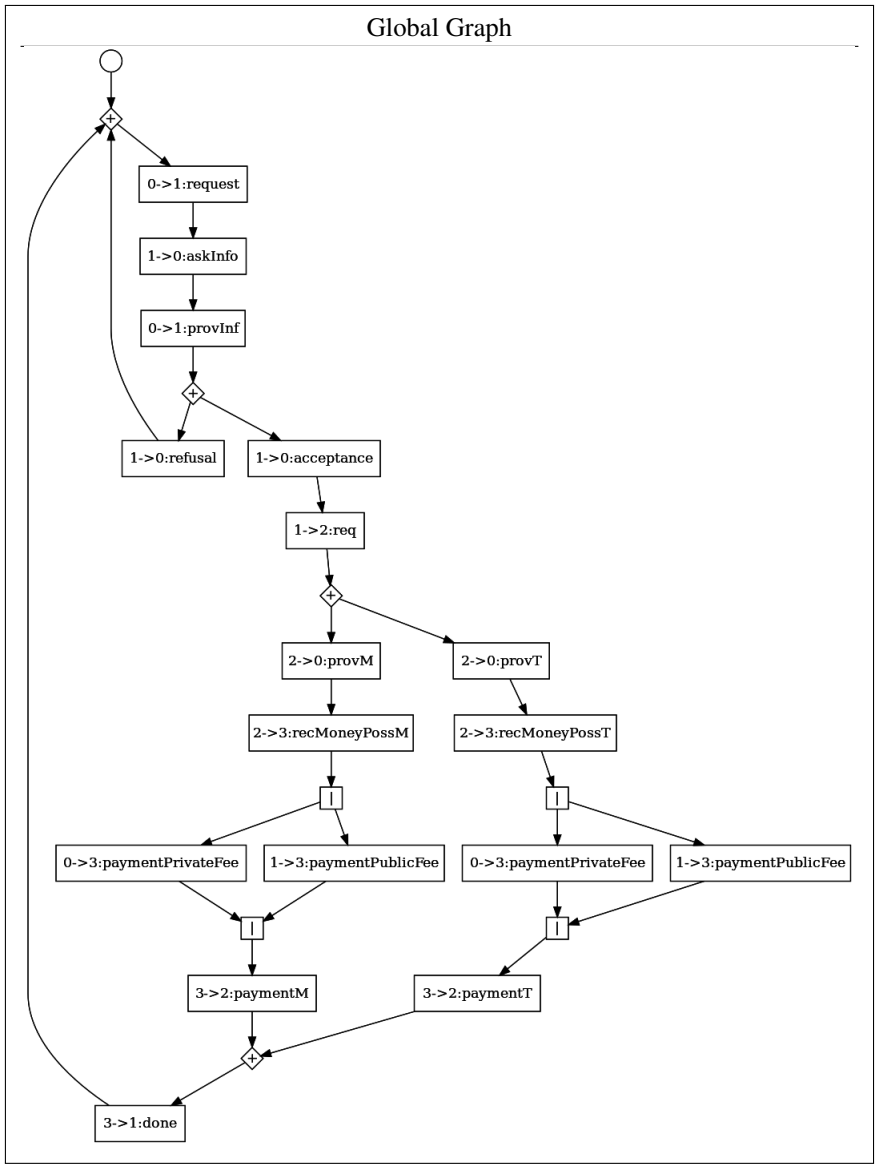

CFSMs
0
1
s00
s10
s20
s40
s130
s150
s100
s180
s170
s200
s210
s230
s30
s220
s60
s50
s80
s70
s110
s90
s120
s140
s160
s190
s01
s11
s21
s41
s131
s151
s101
s181
s201
s171
s31
s231
s211
s51
s61
s221
s111
s71
s81
s141
s121
s91
s161
s191
01!m1
01!a1
01!m2
01!m3
01!a3
01!m1
01!m2
01!a3
01!a2
01!a1
01!a1
01!a1
01!a2
01!a2
01!a2
01!a1
01!m2
01!a1
01!a2
01!a2
01!a2
01!a1
01!m2
01!a1
01!a2
01!a3
01!a3
01!a2
01!a3
01!a2
01!m1
01!a2
01!a1
01!a3
01!a1
01!m3
01!a1
01!a3
01!a2
01!a1
01!m3
01!a1
01!a3
01!a2
01!a1
01!m1
01!a1
01?m1
01?a1
01?m2
01?m3
01?a3
01?m2
01?a3
01?m1
01?a2
01?a1
01?a3
01?a1
01?a3
01?a2
01?a1
01?a1
01?m2
01?a1
01?a2
01?a2
01?a2
01?a3
01?a1
01?a2
01?a3
01?a3
01?a2
01?a1
01?m2
01?a2
01?a2
01?m3
01?m1
01?a2
01?a3
01?a1
01?a2
01?a3
01?a3
01?a3
01?a2
01?m3
01?a1
01?a3
01?a3
01?a2
01?a3
01?a1
01?m3
01?a1
01?a2
01?a1
01?a2
01?a3
01?a1
01?m3
01?m1


Global Graph
0->1:m3
0->1:a3
0->1:m1
0->1:a1
0->1:m2
0->1:a2

F.2 Sanitary Agency

This protocol, adapted from [27], models a software system that aims at "supporting elderly citizens in receiving sanitary assistance from the public administration". In our formalisation, machine `0` is the *Citizen*, machine `1` is the *Sanitary Agency*, machine `2` is the *Coop*, and machine `3` is the *Bank*.




Global Graph
0->1:request
1->0:askInfo
0->1:provInf
1->0:refusal
1->0:acceptance
1->2:req
2->0:provM
2->0:provT
2->3:recMoneyPossM
2->3:recMoneyPossT
0->3:paymentPrivateFee
1->3:paymentPublicFee
0->3:paymentPrivateFee
1->3:paymentPublicFee
3->2:paymentM
3->2:paymentT
3->1:done