Multiparty Session Nets

Luca Fossati[1], Raymond Hu[2], and Nobuko Yoshida[2]

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

Abstract. This paper introduces global session nets, an integration of multiparty session types (MPST) and Petri nets, for role-based choreographic specifications to verify distributed multiparty systems. The graphical representation of session nets enables more liberal combinations of branch, merge, fork and join patterns than the standard syntactic MPST. We use session net token dynamics to verify a flexible conformance between the graphical global net and syntactic endpoint types, and apply the conformance to ensure type-safety and progress of endpoint processes with channel mobility. We have implemented Java APIs for validating global session graph well-formedness and endpoint type conformance.

1 Introduction

Backgrounds and motivations In the early 2000s, there was an active debate on the ways in which various foundations could be applied to the description and verification of Web service standards, triggered by both researchers and developers working on Web services. Two of the major formalisms actively discussed are Petri nets and the π-calculus: the former can offer flexible graphical models of parallel workflows, while the latter can describe process interactions and mobility of channels in a textual format. A working group called Petri-and-Pi was led by Milner and van der Aalst in 2004 to seek meeting points. As a direction in a similar vein, this paper develops a new graphical formulation of multiparty session types (MPSTs) [12] based on Petri nets (PNs) that we call *session nets*. Our main motivations are (1) to offer graphical global specifications based on Petri nets that cannot be directly represented in MPST systems based on "linear" syntactic types [2, 4, 9, 12]; and (2) to apply Petri net token dynamics to a *conformance validation* which can guarantee independent endpoint processes satisfy safety and progress. We believe the resulting graphical representation, similar to notations used in BPMN [3] and UML [17], and accompanying token model will help engineers to write and understand MPST global protocols.

Session nets An MPST framework starts with global descriptions of the message passing protocols by which the participants should interact. In session nets (Figure 1), global protocols are specified by a combination of multiparty role $(\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots)$ and message $(a, b, c, \ldots)$ information over a PN control flow structure. Global session execution is modelled by standard PN token dynamics: branches and merges at places correspond to internal and external choices in the protocol flow at the specified roles; unlabelled transitions correspond to internal fork/join

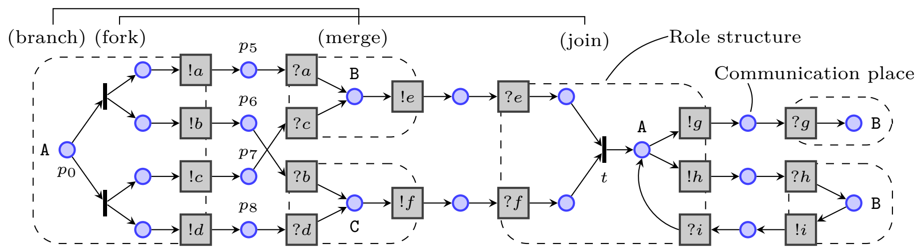


Fig. 1. Interleaved (i.e. non-nested) choice (branch-merge) and parallel (fork-join) structures with "criss-crossing" paths, leading to a recursive protocol segment.

synchronisations; and labelled transitions to observable message I/O actions (e.g. $?a$ and $!a$). Decoupling I/O transitions gives a natural asynchronous model.

The session net in Figure 1 cannot be represented by the global type syntax of [2, 4, 9, 12]. Firstly, because the "criss-crossing" of the middle two of the four paths from p_0 to t cannot be expressed in the tree structure of a linear syntax. Secondly, each of these paths flows from the initial branch through a fork, but then goes through a merge before the join. This interleaving of choice (branch-merge) and parallel (fork-join) structures is not supported by the nesting of choice and parallel constructors imposed by standard global type syntax. A technical report [19] includes session graphs of larger application protocols from [3].

Due to the flexibility of PN structures, a key design point in session nets is to characterise the nets that correspond to coherent protocols that are safely realisable as a system of distributed, asynchronous endpoints. In our framework, well-formed session graphs guarantee that net execution exhibits safety, in PN terminology (i.e. 1-boundedness), and an MPST-based form of progress. Safety states that no place is ever occupied by more than one token at a time. Progress means that every marking reachable from the initial marking enables a transition or is a terminal marking, in which tokens occupy only terminal places. § 2 defines session graphs, which are free-choice PNs by construction, their well-formedness conditions, and shows the above properties.

Conformance Unlike the typical top-down projection from global to local types in previous MPST systems [2, 4, 9, 12], we introduce a *conformance* relation between syntactic endpoint types and well-formed nets. § 3 shows our conformance allows each endpoint type to be validated against a net independently, while guaranteeing that their behaviour in composition respects the behaviour of the global net. Conformance between syntactic endpoints and global graphs is also motivated by practice: developers of Web services and other distributed applications often use expressive graphical patterns [3, 7, 24] for global specifications, but implement the endpoint programs using relatively primitive send/receive APIs, such as network socket or RPC interfaces. Our conformance accepts valid expansions of parallel specifications into a sequence of interleaved actions at the endpoint implementation level, and captures several session typing concepts, such as branch subtyping [8] and certain forms of asynchronous output permutations [6, 15].

Conformance is validated as a bidirectional I/O simulation between the (localised) net execution of a session graph and the behaviour of an individual role given by its type. It works by checking that every output specified by a local output type is accepted by the session net (acting as an environment comprising the external roles), and that every message sent in the session net by another role to the local role is handled by a local input type. For example, T_1 and T_4 are different endpoint types that each conform to the session net in Figure 1 for the `A` role.

$$\begin{array}{ll}
T_1 = !\{\mathtt{B}\langle a\rangle.!\mathtt{C}\langle b\rangle.T_2,\ \mathtt{C}\langle b\rangle.!\mathtt{B}\langle a\rangle.T_2, & T_4 = !\{\mathtt{B}\langle a\rangle.!\mathtt{C}\langle b\rangle.T_5,\ \mathtt{B}\langle c\rangle.!\mathtt{C}\langle d\rangle.T_5\} \\
\qquad \mathtt{B}\langle c\rangle.!\mathtt{C}\langle d\rangle.T_2,\ \mathtt{C}\langle d\rangle.!\mathtt{B}\langle c\rangle.T_2\} & T_5 = ?\{\mathtt{B}\langle e\rangle.?\mathtt{C}\langle f\rangle.T_6,\ \mathtt{C}\langle f\rangle.?\mathtt{B}\langle e\rangle.T_6\} \\
T_2 = ?\{\mathtt{B}\langle e\rangle.?\mathtt{C}\langle f\rangle.T_3,\ \mathtt{C}\langle f\rangle.?\mathtt{B}\langle e\rangle.T_3\} & T_6 = !\{\mathtt{B}\langle h\rangle.?\{\mathtt{B}\langle i\rangle.!\{\mathtt{B}\langle g\rangle.\mathtt{end}\}\}\} \\
T_3 = \mu\,\mathtt{t}.!\{\mathtt{B}\langle g\rangle.\mathtt{end},\ \mathtt{B}\langle h\rangle.?\{\mathtt{B}\langle i\rangle.\mathtt{t}\}\} &
\end{array}$$

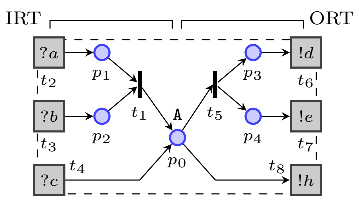
The type $!\{\mathtt{B}\langle a\rangle.T, \mathtt{C}\langle b\rangle.T', \ldots\}$ denotes a choice between outputs $\mathtt{B}\langle a\rangle$ followed by T, $\mathtt{C}\langle b\rangle$ followed by T', etc.; dually $?\{...\}$ for input choice. For singleton choices, we can omit the curly braces, e.g. $!\mathtt{C}\langle b\rangle$. A recursive type is denoted by $\mu\mathtt{t}.T$. Type T_1 corresponds most "directly" to the structure relevant to `A` in the graph. The parallel forks after p_0 to `B` and `C` are expanded into the sequential interleaved outputs (a, b and c, d) in each branch. This is followed in T_2 by the interleaved inputs (e, f) in the next part joining at t. Conformance prioritizes parallel outputs over inputs to prevent deadlocks (§ 3). T_3 conforms to the final part (after t) with a recursive type containing the branch by `A` to either enact the loop (g) or end the protocol (h). T_4 differs from T_1 by safely under-specifying a subset of the interleaved outputs (analogously to MPST output subtyping) in the first part, and performing only one specific trace of the recursive branch; replacing T_6 with $!\{\mathtt{B}\langle g\rangle.\mathtt{end}\}$ would also be conformant. T_1 and T_4 are each guaranteed compatible with any independently conformant `B` and `C` endpoints.

In § 3, we use conformant endpoint types to type check endpoint session processes, including channel passing and session delegations [12]. We show that safety and progress of a well-formed session net are reflected in the MPST safety and progress of a system of conformant, well-typed endpoint processes. This approach gives a natural application for our novel notion of progress in PNs. We have implemented Java APIs for validating session graph well-formedness and endpoint type conformance to demonstrate the tractability of our framework, which are available from [18]. The technical report [19] contains use cases [3] and full proofs.

2 Session Net Graphs

2.1 Role Structures and Session Net Graphs

We first define the *labelled Petri net graphs* that we have adapted to represent message passing protocols in the manner of multiparty session types (MPST). We then introduce *role structures*, which are labelled Petri net graphs given by a few simple construction rules. Role structures are interconnected by asynchronous *communication places* to form a complete session net graph.



$$
\begin{aligned}
P &= \{p_0, p_1, p_2, p_3, p_4\}\\
T &= \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}\\
F &= \{(t_1, p_0), (t_4, p_0), (p_1, t_1), (p_2, t_1), (t_2, p_1), (t_3, p_2),\\
&\quad (p_0, t_5), (p_0, t_8), (t_5, p_3), (t_5, p_4), (p_3, t_6), (p_4, t_7)\}\\
f &= \{p_i \mapsto \texttt{A}\}_{0 \le i \le 4}\\
g &= \{t_2 \mapsto ?a, t_3 \mapsto ?b, t_4 \mapsto ?c, t_6 \mapsto !d, t_7 \mapsto !e, t_8 \mapsto !h\}
\end{aligned}
$$

Fig. 2. An example role structure

Petri net graphs We extend standard Petri net graphs with functions f and g to specify MPST roles and message labels [12]. A *labelled Petri net graph* (henceforth, *Petri net graph*) is a tuple $\boldsymbol{P} = \langle P, T, F, f, g\rangle$, where: $P = \{p_0, \dots, p_n\}$ is a finite set of *places*; $T = \{t_0, \dots, t_m\}$ is a finite set of *transitions*; $F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs* (the *flow relation*); $f : P \rightharpoonup R$ is a partial function which associates places to *role names* from the set $R = \{\texttt{A}, \texttt{B}, \texttt{C}, \dots\}$; and $g : T \rightharpoonup L$ is a partial injective function which associates transitions to *message labels* from the set $L = \{\dagger_1 a,\ \dagger_2 b,\ \dagger_3 c, \dots\}$ where $\dagger = ?\,|\,!$ is an *I/O decoration*. Places and transitions are required to be disjoint ($P \cap T = \emptyset$) and their union, denoted by X, is required to be non-empty ($X = P \cup T \neq \emptyset$). The elements $(x, y, \dots)$ of X are called *nodes*. The *pre-set* of $x \in X$ is $\bullet x = \{y \in X \mid (y, x) \in F\}$ and its *post-set* is $x\bullet = \{y \in X \mid (x, y) \in F\}$.

We represent places as circles and arcs as arrows between places and transitions, as in the standard graphical representation. We call *observable* the transitions in the domain of g and represent them as boxes. The other transitions are called *internal* and represented as narrow rectangles, as in Figure 1. Observable transitions are annotated according to the g labelling function: ?-decorated observables are referred to as *inputs*, and !-decorated observables as *outputs*. Places can be annotated according to the f function.

Role structures An *inbound role tree* (IRT) is a Petri net graph $\boldsymbol{P}{=}\langle P, T, F, f, g\rangle$ with $\mathsf{dom}(f) = P$, which forms a directed tree rooted at a place, with set of nodes X and edges F, and such that: (1) every arc is directed towards the root (the root is reachable from every node); (2) every observable transition is an input; (3) if $|X| > 1$, the set of inputs contains all and only leaves. An *outbound role tree* (ORT) is defined dually, but permits observables in non-leaf positions: (1) every arc is directed away from the root; (2) every observable transition is an output; (3) if $|X| > 1$, every leaf is an output. An IRT or ORT with $|X| = 1$ is just a single root place.

Using common terminology [3, 17], we refer to: a place in an IRT as a *merge*, and in an ORT as a *branch*; and a transition in an IRT as a *join*, and in an ORT as a *fork*. Intuitively, an IRT represents the internal synchronisations within a role after receiving control through the arrival of external messages (the input leaf nodes). An ORT represents the decisions leading to the transfer of control to other roles by dispatching external messages (the output leaf nodes). Their asymmetry reflects the I/O asymmetry of the conformance approach (see § 3).

A *role structure* consists of an IRT and an ORT, rooted at a shared place and disjoint elsewhere, for a single role. Figure 2 as a whole shows a RS for role A with core place p_0. We often annotate only the core place in each RS. Formally:

Definition 2.1 (Role structures). Let $\boldsymbol{P}_1 = \langle P_1, T_1, F_1, f_1, g_1\rangle$ be an IRT and $\boldsymbol{P}_2 = \langle P_2, T_2, F_2, f_2, g_2\rangle$ an ORT. Then $\boldsymbol{P} = \langle P_1 \cup P_2, T_1 \cup T_2, F_1 \cup F_2, f_1 \cup f_2, g_1 \cup g_2\rangle$ is a *role structure* (RS) iff: (1) $P_1 \cap P_2 = \{p\}$ and p, called the *core place*, is the root of P_1 and P_2; (2) $T_1 \cap T_2 = \emptyset$; (3) $f_1(p_1) = f_2(p_2) = \{\mathtt{A}\}$ for all $p_1 \in P_1, p_2 \in P_2$ and some $\mathtt{A} \in R$. We use $\boldsymbol{R}_1$, $\boldsymbol{R}_2$,... to denote role structures.

Session net graphs We construct *session net graphs*, using *communication places* to compose role structures by connecting their input and output transitions.

Definition 2.2 (Session net graphs). A *session net graph* (*session graph* or SG for short) $\boldsymbol{G}$ is a Petri net graph generated by the following cases:

1. $\boldsymbol{G} = \boldsymbol{R}$ is a role structure;
2. $\boldsymbol{G} = \langle P_1 \cup P_2, T_1 \cup T_2, F_1 \cup F_2, f_1 \cup f_2, g_1 \cup g_2\rangle$ is the union of disjoint session graphs $\boldsymbol{G}_1 = \langle P_1, T_1, F_1, f_1, g_1\rangle$ and $\boldsymbol{G}_2 = \langle P_2, T_2, F_2, f_2, g_2\rangle$;
3. $\boldsymbol{G} = \langle P \cup \{p\}, T, F \cup \{(t_!, p), (p, t_?)\}, f, g\rangle$ where $\langle P, T, F, f, g\rangle$ is a session graph, $p \notin P$ is a *communication place*, $t_!$ is an output and $t_?$ is an input and: (1) $\exists_{a \in L}(g(t_!) = !a \wedge g(t_?) = ?a)$; (2) $\nexists_{p' \in P \setminus \mathsf{dom}(f)}((t_!, p') \in F \vee (p', t_?) \in F)$.

Communication places represent asynchronous message dependencies between the roles of the connected RSs. Condition 3 prevents connecting any observable transition to more than one communication place ($P \setminus \mathsf{dom}(f)$ gives the set of communication places). In Figure 1, communication places p_5 and p_7 connect the leftmost A and B RSs, while p_6 and p_8 connect the leftmost A and C RSs.

The behaviour of a role in an SG protocol is given by all the RSs for that role and the message causalities with other RSs. Each RS represents a control point in the protocol where an internal decision by the role is activated by incoming messages, leading to the dispatch of subsequent messages. This decision may then be handled as an external choice distributed over multiple RSs downstream. In Figure 1, A's internal choice to send g or h is handled by B over the two right-most B-RSs. Recursive protocols are also formed from the composition of RSs.

Free-choice graphs [10] are a well-known class of Petri net graphs, where complexity is limited by structurally preventing conflicts. A Petri net graph is *free-choice* if, for any arc from a place p to a transition t, either $\bullet t = \{p\}$ or $p\bullet = \{t\}$. The following states that every SG is *free-choice*.

Proposition 2.1. *If $\boldsymbol{G}$ is an SG, then $\boldsymbol{G}$ is a free-choice Petri net graph.*

2.2 Well-formedness of Session Net Graphs

Paths, cycles and diamonds Let $\boldsymbol{G} = \langle P, T, F, f, g\rangle$ and $X = P \cup T$. A node $x \in X$ is *initial* if $\bullet x = \emptyset$ and *terminal* if $x\bullet = \emptyset$. We write $Term(\boldsymbol{G})$ for the set of terminal nodes in $\boldsymbol{G}$. $F_{\text{-}x} = \{(x', x'') \mid x' \in X \setminus \{x\}, x'' \in X \setminus \{x\}, (x', x'') \in F\}$ denotes the restriction of F to $X \setminus \{x\}$. We extend this definition to a set of nodes

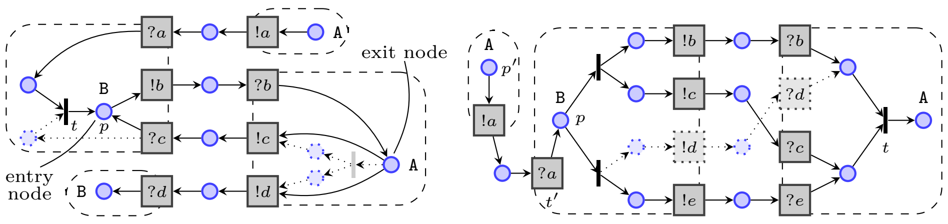


Fig. 3. Illustration of cycles and diamonds

in the natural way, where we omit set parenthesis, e.g. we write $F_{-x,y} = F_{-\{x,y\}}$. The reflexive and transitive closure of a relation $\Re$ is denoted $\Re^*$.

A *path* in $\boldsymbol{P}$ is a finite, non-empty sequence of nodes $x_0..x_n$ such that $(x_i, x_{i+1})_{0 \leq i \leq n-1} \in F$. We let $\sigma, \sigma', \ldots$ range over the set of paths augmented by the empty sequence ϵ; $\sigma\sigma'$ denotes the concatenation of σ and σ'. We sometimes treat σ as the set of nodes occurring in it, e.g. we write $\sigma \cup \sigma'$. We say σ is a *simple path* iff every $x \in \sigma$ occurs exactly once; σ *contains* a node x if $x \in \sigma$.

A *cycle* φ is a path $x\, x_1..x_n\, x$ where $x\, x_1..x_n$ is a simple path. A node x is: an *entry node of* φ iff there is a path σ from an initial node to x and $\sigma \cap \varphi = \{x\}$; an *exit node of* φ iff there is a path σ' from x to a terminal node and $\sigma' \cap \varphi = \{x\}$. Figure 3 (left) shows a cycle, along with its entry and exit nodes.

A *diamond* δ from *start* x to *end* y, $x \neq y$, is a pair of paths $\delta = \langle x\, \sigma_1\, y,\ x\, \sigma_2\, y \rangle$, where $\sigma_1 \cap \sigma_2 = \emptyset$, $\sigma_1 \cup \sigma_2 \neq \emptyset$ and $x, y \notin \sigma_1 \cup \sigma_2$. δ is *pre-cross-free* if for all $z' \in \sigma_1$ and $z'' \in \sigma_2$, $(z', z'') \notin F^*_{-x,y}$ or $(z'', z') \notin F^*_{-x,y}$. Informally, δ is pre-cross-free if it does not feature a pair of criss-crossing paths between its two sides. In Figure 3 (right), the diamond with start p and end t is pre-cross-free when the dotted part is ignored. Finally, δ is *cross-free* if it is pre-cross-free in the graph obtained by removing the nodes of a path, if any, from an initial node to each $z \in \bullet x$. That is, a cross-free diamond has an entry path via each $z \in \bullet x$ that does not overlap the diamond. The p–t diamond in Figure 3 (right) is cross-free due to the path from p' to t'.

The conditions for an SG to be well-formed are as follows.

Definition 2.3 (Well-formed session graph). An SG $\boldsymbol{G} = \langle P, T, F, f, g \rangle$ is *well-formed* if it is a connected graph that respects the following conditions:

(Reachability)	(R1)	There is exactly one initial node: place $p_I \in P$
	(R2)	All terminal nodes are core places
	(R3)	$\forall_{x \in X}\ ((p_I, x) \in F^* \wedge \exists_{y \in Term(\boldsymbol{G})}\ ((x, y) \in F^*))$
(Labels)	(L1)	$\forall_{p \in P}, \forall_{t,t' \in p\bullet}\ (\{f(p') \mid (t, p') \in F^*\} = \{f(p') \mid (t', p') \in F^*\})$
	(L2)	$\forall_{t \in \mathsf{dom}(g)}, \exists_{p \in P \setminus \mathsf{dom}(f)} ((p, t) \in F \vee (t, p) \in F)$
(Cycles)	(C1)	If x is an entry node for some cycle φ, $x \in P$
	(C2)	If x is an exit node for some cycle φ, $x \in P$
(Diamonds)	(D1)	If $\langle x\sigma_1 y, x\sigma_2 y \rangle$ is a diamond, then $x \in T \Rightarrow y \in T$
	(D2)	If $\langle x\sigma_1 y, x\sigma_2 y \rangle$ is cross-free, then $x \in P \Rightarrow y \in P$
	(D3)	If $\langle t\sigma_1 y, t\sigma_2 y \rangle$ is cross-free, then for all $p \in \sigma_1$ and $t' \in p\bullet$, there is a σ'_1 such that $t' \in \sigma'_1 y$ and $\langle t\sigma'_1 y, t\sigma_2 y \rangle$ is cross-free

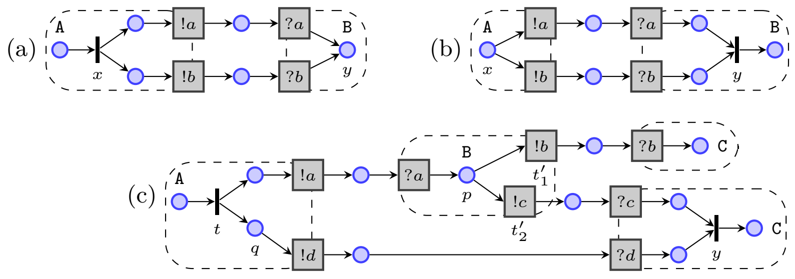


Fig. 4. Badly-formed session graphs illustrating conditions (D1)–(D3)

The first five conditions correspond to basic properties of MPST global types. (R1)–(R3) ensure that every node is reachable from the initial place and a terminal place is reachable from them. (L1) checks that the sets of roles involved in each case of a branch are equal (branch mergeability [4, 9]). (L2) ensures that the SG construction has connected every input and output to a communication place.

The remaining conditions ensure safety and progress of token dynamics, by constraining the composition of branch-merge, fork-join and recursive structures to be a realisable MPST protocol. (C1) and (C2) state that an entry or exit node of any cycle is a place. (D1) requires a diamond starting at a transition to also end at a transition. (D2) imposes a dual condition only on cross-free diamonds. (D3) checks that branches along a cross-free transition-start diamond are re-merged before the diamond ends. Cross-free diamonds represent the "minimal" diamond structures for which these latter constraints need hold. Checking these conditions on cross-free diamonds only (i.e. not all diamonds) permits a larger set of well-formed SGs, e.g. the p_0–t diamond in Figure 1 is not checked for (D2).

We illustrate conditions (C1), (C2) and (D1)–(D3) by examples. In Figure 3 (left), if the dotted structure in the top-left RS for B is added, the transition t would be the entry node of a cycle, violating (C1): the net execution from the initial place would be immediately stuck. If instead the dotted structure in the bottom-right RS for A is added, the greyed-out internal transition would be an exit node, violating (C2): the net execution would be unsafe, allowing an unbounded number of tokens to accumulate within the cycle. Figure 4 (a)–(c) give badly-formed SGs that violate conditions (D1)–(D3), respectively. In (a), the diamond opened by a fork but closed by a place is unsafe (not 1-bounded). In (b), the (cross-free) diamond opened by a branch but closed by a transition will be stuck. Note that it is not necessary to apply this condition to non-cross-free diamonds, e.g. the p_0–t diamond in Figure 1. In (c), the branch at p along the upper side of the t–y diamond will prevent the net from terminating if t'_1 is chosen by B.

Proposition 2.2. *For any SG* $\boldsymbol{G}$*, well-formedness is decidable.*

Deciding well-formedness conditions (R1) to (C2) is straightforward from their definitions. For (D1) and (D2), we can show that if the properties hold for any SG diamond comprised of simple paths, they hold for all general diamonds that may be derived from the "simple diamond" by performing some number of cycles along its sides. The case of (D3) is similarly decided by checking only the diamonds restricted to simple paths from the start to p and from p to the end.

2.3 Session Nets

A *Petri net* $\langle \boldsymbol{P}, M\rangle$ is a Petri net graph $\boldsymbol{P} = \langle P, T, F, f, g\rangle$ with a *marking* $M : P \to \mathbb{N}_0$. The following is standard terminology. A place $p \in P$ contains n *tokens* in M, if $M(p) = n$. A transition $t \in T$ is *enabled* at M (written $\langle \boldsymbol{P}, M\rangle \xrightarrow{t}$) when $M(p) > 0$ for every $p \in \bullet t$. When t is enabled it may *fire*, yielding a new marking M' (written $\langle \boldsymbol{P}, M\rangle \xrightarrow{t} \langle \boldsymbol{P}, M'\rangle$) such that: $M'(p) = M(p) - 1$, for all $p \in (\bullet t \setminus t\bullet)$; $M'(p) = M(p)+1$, for all $p \in (t\bullet \setminus \bullet t)$; $M'(p) = M(p)$, otherwise. We may omit $\boldsymbol{P}$ if it is clear from the context. A *firing sequence* $M_0 \xrightarrow{t_1} M_1 \ldots \xrightarrow{t_n} M_n$ can also be written $\phi : \langle \boldsymbol{P}, M_0\rangle \xrightarrow{s} \langle \boldsymbol{P}, M_n\rangle$, where $s = t_1 \ldots t_n$. A marking M' is *reachable* from M in $\boldsymbol{P}$ if there is a firing sequence ϕ from $\langle \boldsymbol{P}, M\rangle$ to $\langle \boldsymbol{P}, M'\rangle$.

Definition 2.4 (Session nets). *Let $\boldsymbol{G} = \langle P, T, F, f, g\rangle$ be a well-formed SG with initial place $p_I \in P$. A Petri net $\boldsymbol{N} = \langle \boldsymbol{G}, M\rangle$ is: 1) an* initial session net *and M is the* initial marking *for $\boldsymbol{G}$ iff $M(p_I) = 1$, and $M(p) = 0$ for all $p \in P \setminus \{p_I\}$; 2) a* session net *iff M is reachable from the initial marking M_0 for $\boldsymbol{G}$.*

Session nets satisfy the standard safety of Petri nets [10, 16], i.e. no place contains more than one token in any marking reachable from the initial marking. Formally: a Petri net $\langle \boldsymbol{P}, M\rangle$ is *safe* iff $M'(p) \le 1$, for all p and M' reachable from M in $\boldsymbol{P}$.

Theorem 2.1 (Safety). *Every initial session net is safe.*

We want to ensure that sessions can always terminate successfully [2, 12, 20]. Standard Petri nets liveness [10, 16] asks for continuous execution in a system such that no part ever becomes redundant, which is not practical for general sessions. Deadlock-freedom instead requires that every reachable marking enables some transition, which does not ensure the progress of all session participants.

Let $\langle \boldsymbol{G}, M\rangle$ be a session net for $\boldsymbol{G} = \langle P, T, F, f, g\rangle$. The marking M is *terminal* in $\boldsymbol{G}$ just when, for all $p \in P$, $M(p) > 0$ implies $p \in \mathit{Term}(\boldsymbol{G})$, i.e. only terminal places contain tokens. Progress asks for some terminal marking to be reachable:

Theorem 2.2 (Progress). *Let $\boldsymbol{N} = \langle \boldsymbol{G}, M\rangle$ be a session net. Then there is a terminal marking M' which is reachable from M in $\boldsymbol{G}$.*

The proofs of decidability of well-formedness (Proposition 2.2), safety (Theorem 2.1) and progress (Theorem 2.2) are based on a conspicuous set of basic properties of diamonds and cycles in well-formed SGs.

3 Endpoint Types and Conformance

Endpoint types represent the local view of a global protocol from the perspective of a role. This section defines *conformance* between well-formed SGs and syntactic endpoint types. Using the results of § 2, we show the key property of our framework: executing a system of independently conformant endpoints preserves conformance to the corresponding global net execution, thereby ensuring safety and progress.

Endpoint multiparty session types Syntactic endpoint types provide a more programmatic specification for implementation, to be verified by type checking (as shown in § 3) or type inference (along the line of [23]). We define their syntax and LTS with message buffers for asynchronous FIFO communication.

Endpoint types are defined as follows:

$$T ::= \ ?\{\mathtt{r}_i\langle a_i\rangle.T_i\}_{i\in I} \ \mid \ !\{\mathtt{r}_i\langle a_i\rangle.T_i\}_{i\in I} \ \mid \ \mu\,\mathtt{t}.T \ \mid \ \mathtt{t} \ \mid \ \mathsf{end}$$

Input choice $(?\{\mathtt{r}_i\langle a_i\rangle.T_i\}_{i\in I})$ is an external choice, receiving one of the I-indexed messages labelled a_i from role $\mathtt{r}_i$ $(\mathtt{A}, \mathtt{B}, \ldots)$. Dually, *output choice* $(!\{\mathtt{r}_i\langle a_i\rangle.T_i\}_{i\in I})$ internally chooses one of the a_i messages to send to $\mathtt{r}_i$. $\mathtt{t}$ is a recursion variable, $\mu\,\mathtt{t}.T$ is a recursive type that binds $\mathtt{t}$ in T, and end is the terminated type. We assume that all labels in types are distinct and recursive types are guarded, taking an equi-recursive view of types [2, 12]. Let R be a set of roles, then:

$$C ::= (\vec{T}, \vec{w}) \qquad \vec{T} = (T_{\mathtt{r}})_{\mathtt{r}\in R} \qquad \vec{w} = (w_{\mathtt{rr}'})_{\mathtt{r}\neq\mathtt{r}'\in R} \qquad w ::= \vec{a}$$

where C denotes *configurations* and w denotes *buffers*. Let m denote the *actions* $m ::= \mathtt{r}!\mathtt{r}'\langle a\rangle \mid \mathtt{r}?\mathtt{r}'\langle a\rangle$. We write $!m$ to stand for $\mathtt{r}!\mathtt{r}'\langle a\rangle$ for some $\mathtt{r}$, $\mathtt{r}'$ and a; similarly for $?m$. The relation $T \xrightarrow{m} T'$, on endpoint types for role $\mathtt{r}$, is given by:

$$!\{\mathtt{r}_i'\langle a_i\rangle.T_i\}_{i\in I} \xrightarrow{\mathtt{r}!\mathtt{r}_i'\langle a_i\rangle} T_i \qquad ?\{\mathtt{r}_i'\langle a_i\rangle.T_i\}_{i\in I} \xrightarrow{\mathtt{r}?\mathtt{r}_i'\langle a_i\rangle} T_i \qquad \frac{T[\mu\,\mathtt{t}.T/\mathtt{t}] \xrightarrow{m} T'}{\mu\,\mathtt{t}.T \xrightarrow{m} T'}$$

We write $T \xrightarrow{m}$ iff $T \xrightarrow{m} T'$ for some T'. Lastly, $(\vec{T}, \vec{w}) \xrightarrow{\mathtt{r}\dagger\mathtt{r}'\langle a\rangle} (\vec{T}', \vec{w}')$ iff:

$$\dagger =! \implies (T_{\mathtt{r}} \xrightarrow{\mathtt{r}!\mathtt{r}'\langle a\rangle} T_{\mathtt{r}}' \wedge (i \neq \mathtt{r} \Rightarrow T_i' = T_i) \wedge w_{\mathtt{rr}'} \cdot a = w_{\mathtt{rr}'}' \wedge (ij \neq \mathtt{rr}' \Rightarrow w_{ij} = w_{ij}'))$$

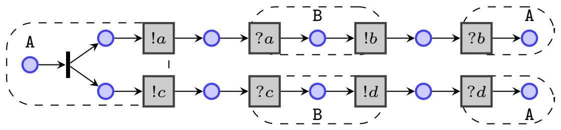
$$\dagger =? \implies (T_{\mathtt{r}} \xrightarrow{\mathtt{r}?\mathtt{r}'\langle a\rangle} T_{\mathtt{r}}' \wedge (i \neq \mathtt{r} \Rightarrow T_i' = T_i) \wedge w_{\mathtt{r}'\mathtt{r}} = a \cdot w_{\mathtt{r}'\mathtt{r}}' \wedge (ij \neq \mathtt{r}'\mathtt{r} \Rightarrow w_{ij} = w_{ij}'))$$

Output by $\mathtt{r}$ to $\mathtt{r}'$ enqueues a message in the buffer. Input by $\mathtt{r}'$ consumes messages in the same order, checking that the label matches one of those expected.

Conformance Conformance replaces the usual projection found in MPST systems [12]. Similarly to safe projections and session type subtyping [8], conformance relates the local protocol behaviour of a role to the global specification. Unlike projection, it uses the global behavioural model (i.e. net dynamics) to validate each local behaviour at endpoint level.

The functions $\mathtt{local}(t)$ and $\mathtt{remote}(t)$ lookup the *local* and the *remote role* of an observable transition t, respectively. Given $\boldsymbol{G} = \langle P, T, F, f, g\rangle$, let $t \in \mathsf{dom}(g)$. Then $\mathtt{local}(t) = f(p)$, for $p \in \mathsf{dom}(f)$ such that $((p,t) \in F \vee (t,p) \in F)$. Similarly, $\mathtt{remote}(t) = f(p)$, for $p \in \mathsf{dom}(f)$ such that there are $p' \notin \mathsf{dom}(f)$ and t', where either: $\{(p,t'), (t',p'), (p',t)\} \subseteq F$ if $g(t) = ?a$; or $\{(t,p'), (p',t'), (t',p)\} \subseteq F$ if $g(t) = !a$. We define the *projected LTS* on session nets for a role $\mathtt{r}$ as follows:

1. $\langle \boldsymbol{G}, M\rangle \xrightarrow{\mathtt{r}\dagger\mathtt{r}'\langle a\rangle} \langle \boldsymbol{G}, M'\rangle$ if $M \xrightarrow{t} M'$, $g(t) = \dagger a$, $\mathtt{local}(t) = \mathtt{r}$ and $\mathtt{remote}(t) = \mathtt{r}'$
2. $\langle \boldsymbol{G}, M\rangle \xrightarrow{\tau_{\mathtt{r}}} \langle \boldsymbol{G}, M'\rangle$ if $M \xrightarrow{t} M'$ and $\mathtt{local}(t) \neq \mathtt{r}$.



$T_{\mathtt{A}}^{bad} = !\mathtt{B}\langle a\rangle.?\mathtt{B}\langle b\rangle.!\mathtt{B}\langle c\rangle.?\mathtt{B}\langle d\rangle.\mathsf{end}$
$T_{\mathtt{B}}^{bad} = ?\{\mathtt{A}\langle a\rangle.?\mathtt{A}\langle c\rangle.!\mathtt{A}\langle b\rangle.!\mathtt{A}\langle d\rangle.\mathsf{end},$
$\mathtt{A}\langle c\rangle.?\mathtt{A}\langle a\rangle.!\mathtt{A}\langle b\rangle.!\mathtt{A}\langle d\rangle.\mathsf{end}\}$

Fig. 5. Motivation for output priority in independent conformance to parallel SG flows

3. $\langle \boldsymbol{G}, M\rangle \xrightarrow{\tau} \langle \boldsymbol{G}, M'\rangle$ if $M \xrightarrow{t} M'$ and $t \notin \mathsf{dom}(g)$.

We write $\xrightarrow{\tau^*}$ for the reflexive and transitive closure of $\xrightarrow{\tau}$, and $\Rightarrow$ for the reflexive and transitive closure of $\xrightarrow{\tau} \cup \xrightarrow{\tau_{\mathtt{r}}}$. Conformance is defined as follows.

Definition 3.1 (Conformance). An endpoint type $T_{\mathtt{r}}$ *conforms to a session net* $\langle \boldsymbol{G}, M\rangle$, written $T_{\mathtt{r}} \asymp \langle \boldsymbol{G}, M\rangle$, if the following conditions are satisfied:

1. (a) if $T_{\mathtt{r}} \xrightarrow{\mathtt{r}!\mathtt{r}'\langle a\rangle} T'_{\mathtt{r}}$, then $\langle \boldsymbol{G}, M\rangle \Rightarrow\xrightarrow{\mathtt{r}!\mathtt{r}'\langle a\rangle} \langle \boldsymbol{G}, M'\rangle$ and $T'_{\mathtt{r}} \asymp \langle \boldsymbol{G}, M'\rangle$
   (b) if $T_{\mathtt{r}} \xrightarrow{?m}$, then $\langle \boldsymbol{G}, M\rangle \Rightarrow\xrightarrow{?m'}$ for some $?m'$
2. (a) if $\langle \boldsymbol{G}, M\rangle \Rightarrow\xrightarrow{\mathtt{r}!\mathtt{r}'\langle a\rangle}$, then $T_{\mathtt{r}} \xrightarrow{!m}$ for some m
   (b) if $\langle \boldsymbol{G}, M\rangle \Rightarrow\xrightarrow{\mathtt{r}?\mathtt{r}'\langle a\rangle} \langle \boldsymbol{G}, M'\rangle$, then:
   – $T_{\mathtt{r}} \xrightarrow{\vec{m}}\xrightarrow{?m}$, for some $?m$ and sequence of output actions $\vec{m}$
   – if $T_{\mathtt{r}} \xrightarrow{?m}$ for some $?m$, then $T_{\mathtt{r}} \xrightarrow{\mathtt{r}?\mathtt{r}'\langle a\rangle} T'_{\mathtt{r}}$ and $T'_{\mathtt{r}} \asymp \langle \boldsymbol{G}, M'\rangle$
3. if $\langle \boldsymbol{G}, M\rangle \xrightarrow{\tau_{\mathtt{r}}} \langle \boldsymbol{G}, M'\rangle$, then $T_{\mathtt{r}} \asymp \langle \boldsymbol{G}, M'\rangle$

$T_{\mathtt{r}} \asymp \boldsymbol{G}$ ($T_{\mathtt{r}}$ *conforms to* $\boldsymbol{G}$) if $T_{\mathtt{r}} \asymp \langle \boldsymbol{G}, M_0\rangle$ where M_0 is the initial marking.

The asymmetry between cases 1 and 2 is due to choice subtyping [8], and the omission of parallel endpoint types. In 1(a), every endpoint output must be simulated by the session net. In 2(a), an endpoint only has to perform *some* output when the net outputs. Thus endpoint outputs may safely underspecify the global model. Dually, endpoint inputs may be overspecified. In 2(b) and 1(b), the endpoint simulates every input by the net, but not vice versa. In 2(b), we allow the endpoint to output before simulating an input: this is sound because the net can do the same outputs without disabling the original input. Note that the subtyping [8] is included in the conformance: if $T_{\mathtt{r}} \asymp \langle \boldsymbol{G}, M\rangle$ and $T'_{\mathtt{r}} \leqslant T_{\mathtt{r}}$ where $\leqslant$ is defined as in [8, Definition 8], then $T'_{\mathtt{r}} \asymp \langle \boldsymbol{G}, M\rangle$ (see [19]).

Conformant endpoint types for the SG in Figure 1 were explained in § 1. Figure 5 shows a SG between roles $\mathtt{A}$ and $\mathtt{B}$, and endpoint types $T_{\mathtt{A}}^{bad}$ and $T_{\mathtt{B}}^{bad}$ for $\mathtt{A}$ and $\mathtt{B}$, respectively (using the abbreviated notation described in § 1). Note that these types do not independently conform to the SG: $T_{\mathtt{A}}^{bad}$ refines the global protocol by forcing a process to wait for an acknowledgement to a (message b), before sending c; similarly, $T_{\mathtt{B}}^{bad}$ mandates to wait for both a and c before doing any output. When composed together, they get stuck in a deadlock. Conformance is designed to prioritise outputs over inputs, thus ruling out incorrect protocols as $T_{\mathtt{A}}^{bad}$ and $T_{\mathtt{B}}^{bad}$. If output priority was to be relaxed, both $T_{\mathtt{A}}^{bad}$ and $T_{\mathtt{B}}^{bad}$ would be conformant and deadlocks would not be prevented.

Weak transition sequences of a net are finite; hence we have:

$$\begin{array}{lll}
P ::= & \overline{u}[\mathtt{r}_1,..,\mathtt{r}_n](c).P & \text{Request} \\
\mid & u[\mathtt{r}](c).P & \text{Accept} \\
\mid & c!\,\mathtt{r} : l\langle v\rangle; P & \text{Select} \\
\mid & c?\{\mathtt{r}_i : l_i(z_i).P_i\}_{i\in I} & \text{Branch} \\
\mid & P \mid Q \ \mid 0 & \text{Parallel, Nil} \\
\mid & \mu X.P \ \mid X & \text{Recursion} \\
\mid & (\boldsymbol{\nu} a)P \ \mid (\boldsymbol{\nu} s)P & \text{Hiding} \\
\mid & s[\mathtt{r},\mathtt{r}'] : h & \text{Queue}
\end{array}
\qquad
\begin{array}{ll}
h \ ::= \epsilon \mid h\cdot l\langle v\rangle \mid h\cdot s[\mathtt{r}] & \\
v \ ::= a \mid s[\mathtt{r}] \mid x & \text{(values)} \\
u \ ::= a \mid x & \text{(identifiers)} \\
c \ ::= x \mid s[\mathtt{r}] & \text{(sessions)} \\
s, s', ... & \text{(session names)} \\
a, b, ... & \text{(shared names)} \\
x, y, z, ... & \text{(variables)}
\end{array}$$

Fig. 6. Syntax of processes

Proposition 3.1. *For any endpoint type $T_{\mathtt{r}}$ and SG $\boldsymbol{\mathsf{G}}$, conformance is decidable.*

Theorem 3.1 (Soundness). *Let $\boldsymbol{\mathsf{G}} = \langle P, T, F, f, g\rangle$ have initial marking M_0 and f have range R. Let $C_0 = (\vec{T_0}, \vec{\epsilon})$ be an* initial *configuration such that $T_{0\mathtt{r}} \asymp \langle \boldsymbol{\mathsf{G}}, M_0\rangle$, for all $\mathtt{r} \in R$. Let also $C_0 \xrightarrow{m_1} C_1 \ldots \xrightarrow{m_n} C_n$ be such that $C_i = (\vec{T_i}, \vec{w_i})$, for all $i \in \{1, \ldots, n\}$. Then $\langle \boldsymbol{\mathsf{G}}, M_0\rangle \xrightarrow{\tau^*}\xrightarrow{m_1} \langle \boldsymbol{\mathsf{G}}, M_1\rangle \ldots \xrightarrow{\tau^*}\xrightarrow{m_n} \langle \boldsymbol{\mathsf{G}}, M_n\rangle$, for some $M_1 \ldots M_n$; such that $T_{i\mathtt{r}} \asymp \langle \boldsymbol{\mathsf{G}}, M_i\rangle$, for all $i \in \{1, \ldots, n\}$ and $\mathtt{r} \in R$.*

We now define the safety properties of a configuration C, following those in communicating automata [9, § 3]. We say C is terminal if $C = (\vec{\mathsf{end}}, \vec{\epsilon})$.

1. C is a *deadlock configuration* if $\vec{w} = \vec{\epsilon}$, while C is not terminal and no $T_{\mathtt{r}}$ is an output type, i.e. some types are blocked, waiting for messages.
2. C is an *orphan message configuration* if all $T_{\mathtt{r}} \in \vec{T}$ are end but $\vec{w} \neq \emptyset$, i.e. there is at least an orphan message in a buffer.
3. C is an *unspecified reception configuration* if there is $\mathtt{r} \in R$ such that $T_{\mathtt{r}}$ is an input and, for all $\mathtt{r}' \in R$ and a, $T_{\mathtt{r}} \xrightarrow{\mathtt{r}?\mathtt{r}'\langle a\rangle} T'_{\mathtt{r}}$ implies that $|w_{\mathtt{r}'\mathtt{r}}| > 0$ and $w_{\mathtt{r}'\mathtt{r}} \neq a \cdot w$, i.e $T_{\mathtt{r}}$ is prevented from receiving any message from buffer $\mathtt{r}'\mathtt{r}$.

We say C is *deadlock-free* (resp. *orphan message-free*, *reception error-free*) if no C' such that $C \xrightarrow{\vec{m}} C'$ is a deadlock (resp. orphan message, unspecified reception) configuration. C is *safe* if it is deadlock-free, orphan-free and reception error-free.

Theorem 3.2 (Safety and Progress). *Let $\boldsymbol{\mathsf{G}} = \langle P, T, F, f, g\rangle$ be a well-formed SG, where the range of f is R. Let $C_0 = (\vec{T_0}, \vec{\epsilon})$ be an initial configuration such that $T_{0\mathtt{r}} \asymp \boldsymbol{\mathsf{G}}$, for all $\mathtt{r} \in R$. Then* (1) *C_0 is safe; and* (2) *for all C such that $C_0 \xrightarrow{\vec{m}} C$, either C is terminal or $C \xrightarrow{m'} C'$, for some action m'.*

4 Multiparty Asynchronous Session Calculus

Safety and progress are reflected from session graphs onto processes through type conformance. The syntax (Figure 6) is extended from [2], allowing communication with different roles within a single branch. It supports channel mobility and session delegation (i.e. passing and hiding shared/session channels). We

$$[\textsc{Req}]\,\frac{T_{\mathtt{r}_1} \asymp \boldsymbol{G} = \langle P, T, F, f, g\rangle \quad \mathit{\Gamma} \vdash u : \boldsymbol{G} \qquad \mathtt{range}(f) = \{\mathtt{r}_1, .., \mathtt{r}_n\} \quad \mathit{\Gamma} \vdash Q \triangleright \mathit{\Delta}, x : T_{\mathtt{r}_1}}{\mathit{\Gamma} \vdash \overline{u}[\mathtt{r}_1, .., \mathtt{r}_n](x).Q \triangleright \mathit{\Delta}} \qquad [\textsc{Acc}]\,\frac{T_{\mathtt{r}_i} \asymp \boldsymbol{G} \quad \mathit{\Gamma} \vdash u : \boldsymbol{G} \qquad \mathit{\Gamma} \vdash Q \triangleright \mathit{\Delta}, x : T_{\mathtt{r}_i} \quad i \neq 1}{\mathit{\Gamma} \vdash u[\mathtt{r}_i](x).Q \triangleright \mathit{\Delta}}$$

$$[\textsc{Sel}]\,\frac{j \in I \quad \mathit{\Gamma} \vdash P \triangleright \mathit{\Delta}, c : T_j \quad \mathit{\Gamma} \vdash u : \boldsymbol{G}_j}{\mathit{\Gamma} \vdash c!\,\mathtt{r}_j : l_j\langle u\rangle; P \triangleright \mathit{\Delta}, c :!\{\mathtt{r}_i\langle l_i\langle \boldsymbol{G}_i\rangle\rangle.T_i\}_{i\in I}}$$

$$[\textsc{Bra}]\,\frac{\forall i \in I \quad \mathit{\Gamma}, z_i : \boldsymbol{G}_i \vdash P_i \triangleright \mathit{\Delta}, c : T_i}{\mathit{\Gamma} \vdash c?\{\mathtt{r}_i : l_i(z_i).P_i\}_{i\in I} \triangleright \mathit{\Delta}, c :?\{\mathtt{r}_i\langle l_i\langle \boldsymbol{G}_i\rangle\rangle.T_i\}_{i\in I}}$$

$$[\textsc{SSel}]\,\frac{j \in I \quad \mathit{\Gamma} \vdash P \triangleright \mathit{\Delta}, c : T_j}{\mathit{\Gamma} \vdash c!\,\mathtt{r}_j : l_j\langle c'\rangle; P \triangleright \mathit{\Delta}, c :!\{\mathtt{r}_i\langle l_i\langle T_i'\rangle\rangle.T_i\}_{i\in I}, c' : T_j'}$$

$$[\textsc{SBra}]\,\frac{\forall i \in I \quad \mathit{\Gamma} \vdash P_i \triangleright \mathit{\Delta}, c : T_i, z_i : T_i'}{\mathit{\Gamma} \vdash c?\{\mathtt{r}_i : l_i(z_i).P_i\}_{i\in I} \triangleright \mathit{\Delta}, c :?\{\mathtt{r}_i\langle l_i\langle T_i'\rangle\rangle.T_i\}_{i\in I}}$$

Fig. 7. Process typing for conformant endpoints

summarise the semantics adapted from [2]. $\lfloor\textsc{Link}\rfloor$ creates a new session s with bidirectional queues, where $\mathsf{fn}(P)$ is the set of free names of P; $\lfloor\textsc{Sel}\rfloor$ enqueues and $\lfloor\textsc{Bra}\rfloor$ dequeues a message. Other rules give the closure under $|$, ν and structural equivalence $\equiv$ (including $(\boldsymbol{\nu} s)(s[\mathtt{r}_1, \mathtt{r}_1'] : \epsilon \mid .. \mid s[\mathtt{r}_n, \mathtt{r}_n'] : \epsilon) \equiv 0$).

$$\begin{array}{ll}
\lfloor\textsc{Link}\rfloor & \overline{a}[\mathtt{r}_1, .., \mathtt{r}_n](x).P_1 \mid a[\mathtt{r}_2](x).P_2 \mid \cdots \mid a[\mathtt{r}_n](x).P_n \\
& \quad \longrightarrow (\boldsymbol{\nu} s)(\mathit{\Pi}_{i\in\{1,..,n\}}(P_i[s[\mathtt{r}_i]/x] \mid \mathit{\Pi}_{j\in\{1,..,n\}\setminus i}\, s[\mathtt{r}_i, \mathtt{r}_j] : \epsilon)) \quad s \notin \mathsf{fn}(P_i) \\
\lfloor\textsc{Sel}\rfloor & s[\mathtt{r}]!\,\mathtt{r}' : l\langle v\rangle; P \mid s[\mathtt{r}, \mathtt{r}'] : h \longrightarrow P \mid s[\mathtt{r}, \mathtt{r}'] : h \cdot l\langle v\rangle \\
\lfloor\textsc{Bra}\rfloor & s[\mathtt{r}]?\{\mathtt{r}_i' : l_i(z_i).P_i\}_{i\in J} \mid s[\mathtt{r}_j', \mathtt{r}] : l_j\langle v\rangle \cdot h \longrightarrow P_j[v/z_j] \mid s[\mathtt{r}_j', \mathtt{r}] : h
\end{array}$$

Conformance replaces the usual endpoint type projection [12]. Type environments use well-formed SGs $\boldsymbol{G}$ and endpoint types T from the previous sections:

$$\mathit{\Gamma} \ ::= \ \emptyset \ \mid \ \mathit{\Gamma} \cdot u{:}\boldsymbol{G} \ \mid \ \mathit{\Gamma} \cdot X{:}\mathit{\Delta} \qquad\qquad \mathit{\Delta} \ ::= \ \emptyset \ \mid \ \mathit{\Delta} \cdot c{:}T$$

SG/endpoint type messages are injectively mapped to pairs of process labels $l_1, l_2, \ldots$ and $\boldsymbol{G}$ or T, e.g. $?\{\mathtt{r}_i\langle l_i\langle S_i\rangle\rangle.T_i\}_{i\in I}$ where each S is either $\boldsymbol{G}$ (shared channel passing) or T (session delegation). $X : \mathit{\Delta}$ types a recursive process. $\mathit{\Gamma} \vdash P \triangleright \mathit{\Delta}$ is a typing judgement.

Figure 7 lists the key rules, adapted from [2], for typing conformant endpoint processes in the session net setting; the omitted rules are as in [2]. Rule [Req] types a session initiation request by checking that the endpoint type for the session body conforms to the $\boldsymbol{G}$ associated to the shared channel for role $\mathtt{r}_1$; [Acc] types an initiation accept in the dual manner. Rules [Sel] and [Bra] type selection and branching with shared channel passing (i.e. passing SG-typed messages). Rules [SSel] and [SBra] similarly type selection and branching with session delegation (i.e. linear communication of endpoint-typed messages).

Without explicit subsumption typing rules, conformance still enables the typing of processes with branch/select and recursive subtype behaviours [8] and permutation of selections [15], via parallel expansion. By Theorem 3.1, we have the following subject reduction theorem, from which the safety properties for processes are derived as a corollary [12].

Theorem 4.1. *Suppose $\Gamma \vdash P \rhd \emptyset$ and $P \longrightarrow^* P'$. Then $\Gamma \vdash P' \rhd \emptyset$.*

Session net progress (Theorem 2.2) corresponds to the following progress property for processes within a single session [12] (a session net, as any individual global type, models a single protocol). We say $P_0 = \overline{a}[\mathtt{r}_1, .., \mathtt{r}_n](x).P_1 \mid a[\mathtt{r}_2](x).P_2 \mid \cdots \mid a[\mathtt{r}_n](x).P_n$ is *simple* if $a : \boldsymbol{G} \vdash P_0 \rhd \emptyset$, P_i does not contain session delegation, accept, request and hiding, and $\boldsymbol{G} = \langle P, T, F, f, g\rangle$ where $\mathtt{range}(f) = \{\mathtt{r}_1, .., \mathtt{r}_n\}$.

Theorem 4.2 (Progress). *Let $a : \boldsymbol{G} \vdash P_0 \rhd \emptyset$ and let P_0 be simple. Then for all P such that $P_0 \longrightarrow^* P$, either $P \equiv 0$ or $P \longrightarrow P'$, for some P'.*

Thus safety and progress of a well-formed net ensure those of the conforming, well-typed processes. Progress across separate sessions can be obtained by using advanced typing systems, e.g. [2], at the top of the typing systems in Figure 7.

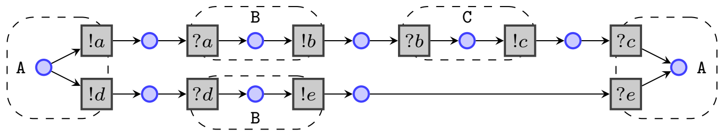
5 Implementation and Related Work

Implementation We have implemented Java APIs for validating session graph well-formedness and endpoint type conformance to demonstrate the tractability of our framework. The code and implementations of all the examples in this paper and [19]. are available at [18]. We plan to integrate this framework into an extension of Session Java [13], using these well-formedness and conformance APIs to extend the type system following § 3.

Related Work Workflow nets [20] (WFNs) are a class of Petri nets originally introduced to describe the operation of business processes. A WFN is an abstraction of a global system on which Petri net techniques are used to verify properties such as dead-lock freedom and proper termination. Session nets differ firstly by specifying multiparty role and message details that WFNs are not concerned with. A sound WFN is a good single, self-contained system, whereas a well-formed session net further ensures that the global protocol, given by the configuration of roles and messages on the structure of the net, is safely realisable as a set of independent, distributed endpoints. Secondly, as an MPST framework, session nets bridge from the global graph to syntactic endpoint specifications (via conformance), that are then used to type-check endpoint code.

Open WF-nets (oWFNs) [14, 22] are an endpoint-oriented adaptation of WF-nets to distributed systems, that starts from constructing a separate net for each endpoint. In contrast, session nets start from the global-oriented SG model of a protocol against which each endpoint is checked for conformance. In oWFNs, the final system properties depend on the specific endpoint composition (effectively treating the complete system as a standalone WF-net), whereas in session nets, any endpoints that are independently conformant to an SG are guaranteed to give a good composition. Like basic WFNs, oWFNs do not explicitly specify or validate multiparty protocol details.

Although Petri nets classes such as WFNs can be interpreted in a communications setting (e.g. in [21], the validation of a sequence of I/O action sequences is subsumed under the general task of accepting traces of fired transitions), they do not explicitly describe communication protocols. The multiparty protocol information captured by an SG and their associated well-formedness is crucial in the design of session nets, allowing us to validate the safe decomposition of the global system into distributed endpoints. Without these concerns, it is not necessary to consider as many structural constraints for WFNs as for well-formed SGs. (A basic WFN requires only (1) one initial and one terminal place, and (2) that any transition is contained in a path from the initial to the terminal place; an SG with a single terminal node is thus a WFN.) As an example, the following shows a SG whose underlying Petri net satisfies safety and progress, but not the conditions on role labelling (specifically, Def. 2.3 (L1)).



This global protocol cannot be safely realised between the distributed endpoints at the implementation level. If an A endpoint chooses to send d in an instance of this protocol, the C endpoint will not receive any message. However, this means C cannot locally determine whether A has indeed selected $!d$, or whether A actually selected $!a$ and C should wait (indefinitely) to do $?b$. A simple way to amend this SG is to ensure that C is also present along the lower branch (not necessarily in the same order), so that the initial internal choice by A is explicitly communicated to C in all eventualities. Other cases of incoherent message labelling, but otherwise safe in terms of the underlying Petri net, are similarly ruled out by well-formedness, e.g. race conditions in parallel protocol flows.

In [21], WF-nets are used to implement tools for checking the conformance of an executed process to a BPEL specification. Their conformance checking, however, is done at run-time and is used to verify the execution trace of a process, via e.g. a logging service or runtime monitor. Our notion of conformance is different, as it is used to statically check the local correctness of each endpoint type by relating them all to an agreed SG. A well-typed system of endpoint processes is guaranteed to behave safely for all executions.

Session nets, as in [2, 4, 9, 12] and other type structures for Web services (e.g. [1, 5, 11]), abstract from specific data types so that data typing can be integrated orthogonally. Recently there have been several works to bridge communicating automata with choreographies or session types [1, 9]. The main focus of [1, 4, 9] are projectability conditions for more general forms of global specifications. The unit of their specifications is an input-output relation between two roles (i.e. $\mathtt{A} \to \mathtt{B}$), whereas a main new feature of session nets is the explicit representation of the internal decision structures of participants to produce outputs in response to inputs. This enables more flexible well-formed global types than those in [1, 4, 9]. None of these works proposed conformance as we have developed for session nets.

Acknowledgements. We thank Luca Aceto and Kohei Honda for their comments and encouragements. The work has been partially sponsored by EPSRC EP/K034413/1 and EP/K011715/1.

References

1. S. Basu, T. Bultan, and M. Ouederni. Deciding choreography realizability. In *POPL*, pages 191–202. ACM, 2012.
2. L. Bettini, M. Coppo, L. D'Antoni, M. D. Luca, M. Dezani-Ciancaglini, and N. Yoshida. Global progress in dynamically interleaved multiparty sessions. In *CONCUR*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008.
3. Business Process Model and Notation. `http://www.bpmn.org`.
4. G. Castagna, M. Dezani-Ciancaglini, and L. Padovani. On global types and multiparty sessions. *Logical Methods in Computer Science*, 8(1), 2012.
5. G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for web services. In *POPL*, pages 261–272. ACM, 2008.
6. T. Chen and K. Honda. Specifying stateful asynchronous properties for distributed programs. In *CONCUR*, volume 7454 of *LNCS*, pages 209–224. Springer, 2012.
7. CPN Homepage. `http://cpntools.org/`.
8. R. Demangeon and K. Honda. Full abstraction in a subtyped pi-calculus with linear types. In *CONCUR*, volume 6901 of *LNCS*, pages 280–296, 2011.
9. P.-M. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *ESOP*, volume 7211 of *LNCS*, pages 194–213, 2012.
10. J. Desel and J. Esparza. *Free Choice Petri Nets (Cambridge Tracts in Theoretical Computer Science)*. Cambridge University Press, 1995.
11. S. Hallé, T. Bultan, G. Hughes, M. Alkhalaf, and R. Villemaire. Runtime verification of web service interface contracts. *Computer*, 43(3):59–66, Mar. 2010.
12. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284. ACM, 2008.
13. R. Hu, N. Yoshida, and K. Honda. Session-Based Distributed Programming in Java. In *ECOOP'08*, volume 5142 of *LNCS*, pages 516–541. Springer, 2008.
14. N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg. Analyzing interacting bpel processes. In *Business Process Management*, volume 4102 of *LNCS*, pages 17–32. Springer, 2006.
15. 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.
16. T. Murata. Petri nets: properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
17. OMG. Unified Modelling Language, Version 2.0, 2004.
18. Java APIs for session nets. `http://www.doc.ic.ac.uk/~rhu/session_nets.html`.
19. Technical Report, Department of Computing, Imperial College London, May 2014. 2014/5.
20. W. M. P. van der Aalst. Verification of workflow nets. In *ICATPN*, volume 1248 of *LNCS*, pages 407–426. Springer, 1997.
21. W. M. P. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and H. M. W. Verbeek. Choreography conformance checking: An approach based on bpel and petri nets. In *The Role of Business Processes in Service Oriented Architectures*. IBFI, 2006.
22. W. M. P. van der Aalst et al. Multiparty contracts: Agreeing and implementing interorganizational processes. *Comput. J.*, 53(1):90–106, 2010.
23. V. T. Vasconcelos and K. Honda. Principal typing scheme for polyadic π-calculus. In *Proc. CONCUR'93*, 1993.
24. Workflow Patterns homepage. `http://www.workflowpatterns.com/`.