A Session Type System for Unreliable Broadcast Communication

Ramūnas Gutkovas¹, Dimitrios Kouzapas², and Simon J. Gay²

¹ Department of Information Technology, Uppsala University, Sweden
ramunas.gutkovas@it.uu.se
² School of Computing Science, University of Glasgow, UK
{dimitrios.kouzapas, simon.gay}@glasgow.ac.uk

Abstract. Session types are formal specifications of communication protocols, allowing protocol implementations to be verified by typechecking. Up to now, session type disciplines have assumed that the communication medium is reliable, with no loss of messages. However, unreliable broadcast communication is common in a wide class of distributed systems such as ad-hoc and wireless sensor networks. Often such systems have structured communication patterns that should be amenable to analysis by means of session types. We introduce a process calculus with unreliable broadcast communication, and equip it with a session type system that we show is sound. We capture two common operations, scatter and gather, inhabiting dual session types. To cope with unreliability in a session we introduce an autonomous recovery mechanism that does not require acknowledgements from session participants. Our session type formalisation is the first to consider unreliable communication.

1 Introduction

Session types [10] are formal specifications of communication protocols, allowing protocol implementations to be verified by typechecking. They provide a discipline for ensuring good communication properties in concurrent systems. Session types have been applied to a range of programming language frameworks and paradigms [2] and several session type technologies have been developed [1].

One of the basic assumptions underlying session types up to now, is the reliability of communication. It is always assumed that messages are never lost and are delivered to the receiver. This is not a realistic assumption to make when it comes to the communication that takes place in ad-hoc and wireless sensor networks. Often networks that use a shared and stateless communication medium use broadcast to deliver messages. Furthermore, messages are lost due to broadcast collisions in the shared medium, and the stateless nature of the communication medium. Nevertheless, in the presence of unreliability, these networks feature structured communication, and we would like to be able to describe it using session types.

In this paper we are the first to introduce a safe session type system for unreliable broadcast communication. The communication semantics that we propose
can capture the broadcast and gather operations in the presence of unreliability. To cope with message loss we propose a recovery mechanism that is enabled only when conditions that imply a recovery situation are met. More, specifically in our setting we assume:

**Locality:** Processes exist as network components, because we want to model real conditions in wireless and ad-hoc networks. Although we do not consider mobility in the current setting, process locality enables the possibility of adding network component mobility in the future. Without mobility we can still capture many classes of networks.

**Channel connectivity:** Communication takes place under conditions of local connectivity. An interaction between the endpoints of a channel may only take place if the two endpoints exist in connected locations. The channel connectivity assumption enables the specification of more realistic wireless network scenarios such as spatial distribution of processes, and processes with different communication ranges.

**Synchronous unreliable broadcast semantics:** In unreliable broadcast semantics, when a process sends a message, an arbitrary (possibly empty) set of receiver processes can receive the message. We assume synchronous broadcasting communication, since we require a stateless communication medium. A stateless communication medium allows us to model more realistic scenarios such as wireless sensor networks.

**Unreliable gather semantics:** Another typical communication pattern in wireless networks is the gather operation, where a process expects to receive values from an arbitrary (possibly empty) set of senders.

**Message loss:** An additional realistic assumption is that sent messages that are supposed to be involved in a gather operation might be lost. Nevertheless, it is also assumed that even if a send message is never received the sender process continues its computation, thus enabling a form of a recovery mechanism.

**Recovery:** When messages are lost it is unavoidable to observe processes that are not synchronised with the overall protocol. To overcome this problem we propose a recovery semantics that is enabled in a non-deterministic fashion only when a process is waiting to receive a message. A non-deterministic hard recovery allows us to model general recovery situations such as message receive time-out, or message collision detection.

### 1.1 Motivation

To further motivate our contribution consider the \( \pi \)-calculus process:

\[
P = s_1(v).P_0 \mid \prod_{i \in I} s_2(x).P_i
\]

where process \( \prod_{i \in I} s_2(x).P_i \) is a parallel composition of input prefixed processes that receive a message on channel \( s \). Consider also an unreliable broadcasting semantics for the above process:

\[
P \rightarrow P_0 \mid \prod_{i \in I} P_j\{v/x\} \mid \prod_{k \in K} s_2(x).P_k
\]
where $I = J \cup K$ for disjoint $J$ and $K$; and broadcast value $v$ is only received by a subset of the receiving processes.

We claim that session types cannot model the above interaction without introducing a complicated and unrealistic level of abstraction. Binary session types are inadequate for the above scenario, since they only describe two-endpoint communication.

Modelling unreliable broadcasting communication in multiparty session types needs to account for message loss in the communication medium; a receiver may or may not receive depending on whether the message was lost in the communication medium or not. The only operation that can model a choice in multiparty session types is the choice operator. For the purposes of demonstration consider the multiparty syntax presented in [7]: type $p \to q : T; G$ denotes a type where role $p$ sends a message with type $T$ to role $q$ and then continues as $G$, and type $p \to q : \{l_i : G_i\}_{i \in I}$ denotes the type where role $p$ makes a choice labelled $l_i$ on role $q$ and continues as $G_i$ for $i \in I$. A multiparty global protocol that attempts to describe an unreliable communication medium using the multiparty choice operation follows:

$$p \to m : T; m \to q : \{\text{send} : m \to q : T; \text{end}, \text{lose} : \text{end}\}.$$  

Consider that role $p$ wants to broadcast a message of type $T$ to role $q$ with role $m$ representing the communication medium. Role $p$ first sends the message to the communication medium $m$ and then role $m$ informs role $q$ of its choice whether to lose or send the message. The above specification has three deficiencies:

(i) role $q$ is informed that a message was broadcast and never received, which is unrealistic since in real applications a role does not know whether a message supposed to be received was sent or not;

(ii) there is an unnecessary loss of abstraction by modeling the communication medium in terms of a role. Often, someone who implements broadcasting applications does not need to know lower level communication details; and

(iii) extending the above scenario for more than one receiver would lead to a complicated communication protocol.

Our solution generalises the theory of binary session types to cope with the semantics of unreliable broadcast and gather. Concretely, a typed session endpoint is dedicated to the process that initiates a session, whereas multiple dually typed endpoints are used by the set of processes accepting the session. For example consider the network:

$$[a_1(s), \bar{s}(1), \bar{s}(y), 0]_l | \prod_{i \in I} [a_2(s), s_2(x), s_1(x + i) 0]_{i},$$

where $[P]_l$ denotes process $P$ in location $l$. A broadcast interaction on name $a$ will create a new session with endpoints $s$ and $\bar{s}$.

$$[a_1(s), \bar{s}(1), \bar{s}(y), 0]_l | \prod_{i \in I} [a_2(s), s_2(x), s_1(x + i) 0]_{i} \rightarrow (\nu s)([\bar{s}(1), \bar{s}(y), 0]_l | \prod_{j \in J} [s_2(x), s_1(x + j) 0]_{j}) | \prod_{k \in K} [a_2(s), s_2(x), s_1(x + k) 0]_{k},$$
Session endpoint \( \bar{s} \) is expected to be unique used only by the session initiation process, whereas the session broadcast interaction may create more than one instances instances of endpoint \( s \). A broadcast action happens from endpoint \( \bar{s} \) towards endpoints \( s \), where in the above example corresponds to the broadcasting of value 1 from endpoint \( \bar{s} \) to endpoints \( s \). Dually, a gather operation synchronises on the sending prefixes of endpoints \( s \) and the receive prefix of endpoint \( \bar{s} \), which simultaneously gathers all sent information. In the above example a gather operation corresponds to the sending of values \( x + i \) from endpoints \( s \) to endpoint \( \bar{s} \). Endpoint \( \bar{s} \) will gather all sent values modulo a pre-defined aggregation operator \( \odot \) and substitute the result for the variable \( y \).

The one to many correspondence between endpoints \( \bar{s} \) and \( s \) enables the use of standard session type duality. In the above example the type of \( \bar{s} \) is \( !\text{int} : \text{int}. end \) with \( \text{int} \) being a base type and dually, the type of \( s \) is \( ?\text{int} : \text{int}. end \).

Outline. In Section 2 we introduce our unreliable broadcast calculus. In Section 3 we define a session type system and prove the main results of this paper. In Section 4 we use our framework to correctly implement a data aggregation protocol from wireless sensor networks. Lastly, we conclude in Section 5.

2 A Broadcast Calculus

In this section, we introduce a broadcast process calculus, its syntax and semantics. Its main defining features are the unreliability of the communication medium and the aggregate broadcast communication primitives. The syntax of the calculus is defined in two levels: processes and networks with locations.

2.1 Syntax

Definition 1 (Process). We need the following set of data. Let \( C \) be a countable set of (shared) channels ranged over by \( a, b, c, \ldots ; S \) be a countable set of session channels ranged over by \( s, s', \ldots ; X \) be a countable set of variables ranged over by \( x, y, z, \ldots ; \) and \( V \) be a countable set of process variables ranged over by \( X, Y, \ldots \). We assume that these sets are disjoint.

We parameterise syntax with expressions and conditions. Let \( E \) be a non-empty set of expressions ranged over by \( e, e', \ldots \). We require parameters: a binary operation \( \odot \) on \( E \) that we call aggregation operation, and a distinguished element 1 of \( E \) called unit. Let \( F \) be a non-empty set of conditions ranged over by \( \varphi \), and a truth predicate \( \models \subseteq F \), where we write \( \models \varphi \) for \( \varphi \in \models \). Both \( E \) and \( F \) may contain variables and the set of free variables is denoted with \( \text{fn}(e) \) and \( \text{fn}(\varphi) \). There is also a substitution function \( e(e'/x) \) defined on \( E \) and \( F \) such that \( \text{fn}(e(e'/x)) \subseteq \text{fn}(e) \cup \text{fn}(e') \) and similarly for conditions. Then, the syntax
of processes is defined as follows:

\[
\begin{align*}
\kappa & ::= s \mid \bar{s} \\
P, Q, R & ::= a(s).P \quad \text{(Request)} \\
& \mid a?_1(s).P \quad \text{(Accept)} \\
& \mid \kappa?_1(e).P \quad \text{(Send/Scatter)} \\
& \mid \kappa?_2(x).P \quad \text{(Receive/Gather)} \\
& \mid \bar{s} \oplus \ell.P \quad \text{(Selection)} \\
& \mid s \& \{\ell_1 : P_1, \ldots, \ell_n : P_n\} \quad \text{(Branching)} \\
& \mid \text{if } \varphi \text{ then } P \text{ else } Q \quad \text{(Conditional)} \\
& \mid \mu X.P \quad \text{(Recursion)} \\
& \mid X \quad \text{(Process Variable)} \\
& \mid 0 \quad \text{(Inaction)}
\end{align*}
\]

(Request) and (Accept) bind \( s \) in \( P \); and (Receive) binds \( x \) in \( P \). Also, (Recursion) is a binder and binds \( X \) in \( P \). We define \( \text{fn}(P) \) to be a set of free channels, session channels, variables, and process variables of \( P \) in the standard way. We identify processes up to \( \alpha \)-equivalence.

In (Request), (Accept), (Send), (Receive), (Branching) and (Selection) forms, \( a, \kappa, \) and \( s \) are called subjects. The forms themselves are called prefixes.

We impose a well-formedness condition on the processes above: all the free occurrences of \( s \) in \( P \) of (Request) are under \( \bar{s} \), and free occurrences of \( s \) in \( P \) of (Accept) are not under \( \bar{s} \).

The (Request) and (Accept) forms are used to initiate a session \( s \) on a shared channel \( a \). They are not symmetric operations in the sense that (Request) is a broadcast operation that may initiate a session with multiple partners, while (Accept) accepts a session from a unique process. We distinguish these endpoints by annotating the process requesting a session with \( \bar{s} \) (aggregate \( s \)), and in the accepting process we make no annotations for \( s \).

There are two flavours of sending and receiving operations determined by the session channel endpoint annotation. The operation \( \bar{s}?_1(e).P \) is a broadcast (Scatter) to all session partners, while \( s?_1(e).P \) is a send operation to one partner. Analogously, \( \bar{s}?_2(x).P \) is an aggregation of messages received (Gather) from the partners, binding it to \( x \), while \( s?_2(x).P \) receives a message from the unique partner. We restrict selection only on \( \bar{s} \) in (Selection) and a choice of branch can be only made on \( s \) (Branching).

It is worth pointing out that processes are not concurrent; we introduce concurrency in the network level. There is no loss of generality, however, since we may have several processes running in a network in the same location. We use the notion of a location of distributed \( \pi \)-calculus [9], although in our formalisation the locations are fixed and the processes are not mobile, that is, they cannot migrate from one location to another.

**Definition 2 (Network).** Let \( \mathcal{N} = \mathcal{L} \cup \mathcal{S} \) ranged over by \( n \), and let \( \mathcal{L} \) be a countable set of locations ranged over by \( l \). Then, the network is defined by the
following grammar:

\[
\begin{align*}
\psi & ::= \varepsilon \mid \psi \mid s \quad \text{(Session State)} \\
N, M & ::= [P \triangleright R \mid \psi]_l \quad \text{(Node)} \\
& \mid N \mid M \quad \text{(Parallel)} \\
& \mid (\nu n)N \quad \text{(Restriction)}
\end{align*}
\]

We extend the \(\text{fn}\) function to networks. We may write \([P \triangleright Q]_l\) for \([P \triangleright Q \mid \varepsilon]_l\). We define \(\text{cnt}(s, \psi) = c\) to denote the number of occurrences \(c\) of \(s\) in \(\psi\) by structural recursion as \(\text{cnt}(s, \varepsilon) = 0\), and \(\text{cnt}(s, \psi \mid s') = \text{cnt}(s, \psi)\) if \(s \neq s'\), otherwise \(\text{cnt}(s, \psi \mid s) = 1 + \text{cnt}(s, \psi)\).

The form \((\text{Node})\) consists of a process \(P\) that is executing in a location \(l\), a recovery process \(R\) that may take over if \(P\) cannot proceed in a session, and a session state store \(\psi\) that tracks the number of session interactions the process performed thus far. A process may participate in several sessions and therefore \(\psi\) is used to track more than one session. Intuitively, a network is a parallel composition of nodes. (Restriction) binds both session and shared channels.

### 2.2 Operational Semantics

We define the operational semantics as a reduction relation on networks. In the standard way, we make use of structural congruence. We also make use of notation \(k \# P\), pronounced as \(k\) is fresh for \(P\), to mean that \(k \not\in \text{fn}(P)\) where \(k \in S \cup C \cup X \cup V\), and similarly for networks \(k \# N\). We write \((\nu n)N\) for the network \((\nu n_1)\ldots(\nu n_m)N\) where the sequence \(\tilde{n} = (n_1, \ldots, n_m)\) may be empty.

**Definition 3 (Structural Congruence).** Structural congruence on processes (resp., session state and networks) is defined to be the smallest congruence relation satisfying the following rules:

\[
\mu X.P \equiv P\{\mu X.P/X\}
\]

\[
\psi \equiv \psi' \quad \text{if } \psi \text{ is a permutation of } \psi'
\]

\[
N \equiv [0 \triangleright 0 \mid \varepsilon]_l \mid N
\]

\[
N \mid M \equiv M \mid N
\]

\[
(M \mid N) \mid S \equiv M \mid (N \mid S)
\]

\[
(\nu n)N \mid M \equiv (\nu n)(N \mid M) \quad \text{if } n \# M
\]

\[
[P \triangleright R \mid \psi]_l \equiv [P' \triangleright R' \mid \psi']_l \quad \text{if } P \equiv P' \text{ and } R \equiv R' \text{ and } \psi \equiv \psi'
\]

Structural congruence on processes includes unrolling of recursion. We identify session states up to permutation (reordering). The structural congruence on network is standard: parallel composition is commutative and associative, with \([0 \triangleright 0 \mid \varepsilon]_l\) as the unit, and the scope of a restricted channel is amenable to extrusion. The clause for nodes simply bridges the congruences.

Given a finite family of networks \(\{N_i\}_{i \in I}\), we write \(\prod_{i \in I} N_i\) for the parallel composition of \(N_1 \mid \cdots \mid N_n\) for non-empty \(I = \{1, \ldots, n\}\), otherwise \([0 \triangleright 0 \mid \varepsilon]_l\).
Definition 4 (Reduction Relation). We define the reduction relation on networks \( N \rightarrow_{G} N' \), parameterised over an arbitrary connectivity graph \( G \subseteq L \times L \), as the smallest relation satisfying the rules given in Fig. 1.

\[
\begin{align*}
\frac{}{i \in I \quad s \neq \psi, \psi_i, R, R_i \quad (l, l_i) \in G} & \quad [\text{RInit}] \\
\frac{[a(s).P \triangleright R | \psi_i] \mid \prod_{i \in I}[a(s).Q_i \triangleright R_i | \psi_i]_{l_i}}{\rightarrow_G (\nu s)[[P \triangleright R | \psi_i] \mid \prod_{i \in I}[Q_i \triangleright R_i | \psi_i]_{l_i}]} \quad [\text{RSel}] \\
\frac{i \in I \quad (l, l_i) \in G \quad \text{cnt}(s, \psi) = \text{cnt}(s, \psi_i)}{[\text{RSel}]} \\
\frac{\ell : Q_i \in B_i \quad (l, l_i) \in G \quad \text{cnt}(s, \psi) = \text{cnt}(s, \psi_i)}{[\text{RRecover}]} \\
\frac{\phi \vdash \phi}{[\text{RTrue}]} \\
\frac{\neg \phi \vdash \phi}{[\text{RFalse}]} \\
\frac{\ell : Q_i \in B_i \quad (l, l_i) \in G \quad \text{cnt}(s, \psi) = \text{cnt}(s, \psi_i)}{[\text{RRecoverSel}]} \\
\frac{N \equiv N' \quad N' \rightarrow_{G} M' \quad M' \equiv M}{N \rightarrow_{G} M} \quad [\text{RCong}] \\
\frac{N \rightarrow_{G} N'}{N | M \rightarrow_{G} N' | M} \quad [\text{RPar}] \\
\frac{N \rightarrow_{G} N'}{(\nu n)N \rightarrow_{G} (\nu n)N'} \quad [\text{RRes}] \\
\end{align*}
\]

Fig. 1. Reduction rules of the broadcast calculus.

With a connectivity graph we can capture spatial distribution of nodes that is common in ad-hoc and wireless sensor networks. Note that connectivity graph is an arbitrary relation and is not required to be neither reflexive nor symmetric.
By not requiring symmetry, allows us to capture the asymmetry of communication often found in WSNs; where a typical situation is that a special node, base station, has more powerful antenna that allows it to broadcast to all the nodes in the network, whereas the more distant nodes with less powerful antennas cannot broadcast directly to the base station. We do not require reflexivity of a connectivity graph, that is, a node may not be able to communicate with itself, for generality reasons as we do not need reflexivity for our results to hold. We do not model mobility of the nodes (nodes are not able to change location), meaning that the connectivity graph \( G \) does not evolve during reductions. Without mobility we can still capture many classes of networks.

We have chosen communication to be synchronous in our system since it seems that kind of communication is more prevalent in broadcasting systems. This means that a sender synchronises with a subset of receivers. We believe that reformulating the system to introduce asynchrony, would require the definition of a queue process that is able to hold messages in an ordered fashion [11].

The rule [RInit] establishes a session between the requesting process node and accepting process nodes. It is a broadcast communication pattern (one-to-many) where there might not be any accepting process nodes (i.e., \( I = \emptyset \)). The session channel is chosen fresh for all session states and recovery processes of participants. Note that the session channels \( s \) in the requesting process \( P \) are annotated as \( s \) by the well-formedness condition on processes (Definition 1). Let us call with respect to the session channel \( s \) the process \( P \) prefixed with the subject \( s \) parent, and with subject \( s \) child.

The rule [RScatter] states that the parent broadcasts \( e \) to the children that continue by substituting \( e \) for \( x_i \). Both the parent and children advance their session states by emitting the corresponding session channel in their session state stores. This is the case for all reduction rules concerning interaction within the session. There is also a precondition that session partners have advanced the same number of steps in the session.

The reduction [RGather] is dual to [RScatter] in the sense that the communication is reversed (many-to-one): the parent receives from children. The rule is an abstraction of a communication pattern that consists of multiple broadcasts from each individual child. The iterated result \( e \) is collected as the product of the aggregation operation \( e_i \cap_{i \in I} e_i \) that is defined to be \( e_1 \cap \ldots \cap e_n \) for nonempty \( I = \{1, \ldots, n\} \) and \( 1 \) in case \( I = \emptyset \). We have not assumed that the \( \cap \) is commutative, thus different orderings of parallel components can lead to a different product in the reduction. Here again \( I \) might be empty. However, here the advance of parent does not correspond to a message loss, but that the node prematurely terminated the gathering operation. For example, in a real network the node could have a time limit for gathering data. When \( I \) is empty, we use the unit \( 1 \) as the product. This pattern is common in ad-hoc networks; see, for example, the RIME communication stack [8] for wireless sensor networks.

The rule [RSel] is similar to [RScatter]; however, here the parent selects the branch that children should take. [RLoss] models message loss for a child. Note that it still emits a session channel in its session stores. As noted before, message
loss for the parent is already captured by [RScatter] and [RPar]. There is no corresponding dual rule to [RSel]: the opposite communication pattern would require an assumption of a non-trivial underlying protocol where the child nodes would establish a consensus on the branch to be taken by the parent. This is unrealistic in the unreliable setting.

The communication rules [RInit], [RScatter], [RGather], and [RSel] adhere to the communication graph $G$, that is, there needs to be a connection between the parent and children locations. We don’t assume that this relation is symmetric.

[RCons], [RRes] and [RPar] are standard, but note that [RPar] is essential for capturing the unreliability of the medium since it can be used to exclude potential communication partners. [RTrue] and [RFalse] are self-explanatory.

Rules [RRecover] and [RRecoverSel] state that a child node may recover from a session, by replacing their running process with the recovery process $R$. The recovery is hard and drops all the sessions that the node was a participant of and clears the session state store. The purpose of these rules is to recover from the situation where the processes cannot continue due to a loss of messages that result in diverged session states. However, note that the rules do not have a condition that triggers recovery allowing nodes to act autonomously. This means that the recovery behaves nondeterministically and the nodes can recover even though the session state has not diverged. Having recovery nondeterministic we abstract over other possible sources of failure, e.g., a timeout of a message reception.

The semantics of [RGather] can be implemented with series of broadcasts to the parent. Thus, the calculus can be encoded in a calculus with just broadcast albeit with added level of indirection. In our previous work [13], we have such an encoding to broadcast psi-calculi [3], and we conjecture that the semantics weakly corresponds to our previous encoding.

3 Session Types

In this section, we introduce the session type system for the broadcast calculus of Section 2 and the main results of the paper: type soundness and safety. Remarkably, the types that we use are the standard binary session types as introduced by Honda et al. [10]. However, we have no session delegation.

**Definition 5 (Session Type).** Let $B$ be a set of base types ranged over by $\beta$. Then, the session types are inductively defined by the following grammar:

$$T ::= 1\beta.T \mid ?\beta.T \mid \oplus \{\ell_i : T_i\}_{i \in I} \mid \&\{\ell_i : T_i\}_{i \in I} \mid \text{end} \mid t \mid \mu t.T$$

$\mu t.T$ is a binder and binds free occurrences of $t$ in $T$. We define a capture avoiding substitution on types $\{T/t\}$ in the usual way. As usual, we identify recursive types with their expansion: $\mu t.T = T\{\mu t.T/t\}$.

We define the duality operation on types $\overline{T}$ recursively as follows:

$$\overline{\text{end}} = \text{end} \quad \overline{1\beta.T} = ?\beta.\overline{T} \quad \overline{\oplus \{\ell_i : T_i\}_{i \in I}} = \&\{\ell_i : \overline{T_i}\}_{i \in I} \quad \overline{\&\{\ell_i : T_i\}_{i \in I}} = \oplus \{\ell_i : \overline{T_i}\}_{i \in I} \quad \overline{t} = t \quad \overline{\mu t.T} = \mu t.\overline{T}$$
We say that two types $T_1$ and $T_2$ are dual if $\overline{T_1} = T_2$. Note $\overline{T} = T$ for any $T$.

We define a transition relation on types that we will use in the typing rules to obtain the session type consistent with the session state of a node.

**Definition 6 (Type Advancement).** The relation $T \rightarrow T'$ is defined inductively as follows:

$$
\begin{align*}
\overline{\beta}.T & \rightarrow \overline{\beta}.T \\
!\beta.T & \rightarrow \oplus\{l_i : T_i\}_{i \in I} \rightarrow T_i \\
& \quad \&\{l_i : T_i\}_{i \in I} \rightarrow T_i \\
\end{align*}
$$

We also define a type $n$-advancement relation $T \rightarrow^n T'$ that says $T'$ is reached in $n$ advancements from $T$, by induction, as follows:

$$
T \rightarrow^0 T' \\
T \rightarrow^n T'' \quad T'' \rightarrow T' \\
\Rightarrow T \rightarrow^{n+1} T'
$$

Type advancement for recursive types is obtained by expansion. We will typically use $n$-advancement relation to advance with regard to a session store $\psi$ as $T \rightarrow^{\text{cnt}(s,\psi)} T'$.

**Definition 7 (Typing Context).** We define shared $\Gamma$ and linear $\Delta$ typing contexts by mutual induction as follows:

$$
\begin{align*}
\gamma & ::= \ x : \beta \mid a : T \mid X : \Delta \\
\delta & ::= \ s : T \mid \bar{s} : T \\
\Gamma & ::= \epsilon \mid \Gamma, \gamma \\
\Delta & ::= \epsilon \mid \Delta, \delta
\end{align*}
$$

We denote with $\Delta, \Delta'$ the concatenation of contexts $\Delta$ and $\Delta'$, and similarly for shared contexts $\Gamma, \Gamma'$. We make use of the function $\text{subj}$ defined on $\gamma$ that extracts the subject as follows $\text{subj}(x : \beta) = x$, $\text{subj}(a : T) = a$, and $\text{subj}(X : \Delta) = X$.

We also define a typing context for the expressions and conditions: $\Gamma_E ::= \epsilon \mid \Gamma, x : \beta$.

By abuse of notation, we denote the restriction of shared typing context $\Gamma$ to expression context by $\Gamma_E$. A domain of a context $\Gamma$, and $\Delta$ are defined as follows:

$$
\text{dom}(\Gamma, \gamma) = \{\text{subj}(\gamma)\} \cup \text{dom}(\Gamma), \quad \text{dom}(\Delta, s : T) = \{s\} \cup \text{dom}(\Delta), \\
\text{dom}(\Delta, \bar{s} : T) = \{s\} \cup \text{dom}(\Delta)
$$

where $\text{dom}(\epsilon) = \emptyset$ in both cases.

In turn, we define the notion of freshness as follows: $\kappa \not\in \Delta$ is defined as $s \not\in \text{dom}(\Delta)$ where $\kappa = s$ or $\bar{s}$, and pronounced as $\kappa$ is fresh for $\Delta$. Similarly, we define $x \not\in \Gamma$, $a \not\in \Gamma$, and $X \not\in \Gamma$ as $x \not\in \text{dom}(\Gamma)$, $a \not\in \text{dom}(\Gamma)$, and $X \not\in \text{dom}(\Gamma)$, respectively.

In the linear context, we distinguish the endpoints as we do in the process syntax. The choice of representing typing contexts as lists and not as sets, as is perhaps more common, has technical convenience. It is important to have multiple copies of a session channel and its type for child nodes. This as we shall see trivialises the parallel typing rule to a simple concatenation of parallel contexts. We extend the notion of type advancement of Definition 6 to linear context advancement based on a session state assertion of Definition 2.
Definition 8 (Linear Context Advancement). Let $\psi$ be an arbitrary session state assertion of Definition 3. Then, we define linear context type advancement $\Delta \rightarrow^\psi \Delta'$ inductively as follows:

$$
\begin{align*}
\varepsilon & \rightarrow^\psi \varepsilon & \Delta \rightarrow^\psi \Delta' & \rightarrow T \rightarrow^\text{cnt}(s,\psi) T' & \rightarrow T' \Delta, s : T \rightarrow^\psi \Delta', s : T' \\
\Delta & \rightarrow^\psi \Delta' & \rightarrow T \rightarrow^\text{cnt}(s,\psi) T' & \rightarrow T' \Delta, \bar{s} : T \rightarrow^\psi \Delta', \bar{s} : T'
\end{align*}
$$

To take an example of linear context advancement, let $\psi = s_1 \mid s_1 \mid s_2$ be session state and $\Delta = \varepsilon, s_1 : !\beta_1, !\beta_1, !\beta_2, \text{end}, s_2 : !\beta_2, \text{end}, s_3 : ?\beta_2, \text{end}$ a linear context. Then, $\Delta \rightarrow^\psi s_1 : !\beta_2, \text{end}, s_2 : !\beta_2, \text{end}, s_3 : ?\beta_2, \text{end}$.

Definition 9 (Typing Judgement). We parameterise our type system on the typing judgements of expressions and conditions in the following way: let us assume the following to be a typing judgement on expressions, and typing judgement on conditions:

$$
\Gamma \vdash E \succ e : \beta \quad \text{and} \quad \Gamma \vdash E \succ \phi : \text{bool}.
$$

The above judgements are arbitrary but required to satisfy the following conditions: type is preserved by substitutions (i) if $\Gamma, x : \beta' \vdash e : \beta$ and $\Gamma \vdash E \succ e' : \beta'$, then $\Gamma \vdash E \succ e[e'/x] : \beta'$; (ii) if $\Gamma \vdash E \succ e : \text{bool}$ and $\Gamma \vdash E \succ e' : \beta$, then $\Gamma \vdash E \succ \phi(e'/x) : \text{bool}$; and that the aggregation operation does not change the type of resulting expression (iii) if $\Gamma \vdash e : \beta$ and $\Gamma \vdash E \succ e' : \beta$, then $\Gamma \vdash E \succ e \odot e' : \beta$.

Our typing judgement is of the following form

$$
\Gamma ; \Delta \vdash N.
$$

It is defined as the smallest relation satisfying the rules given in Fig. 3. We call the network $N$ or process $P$ well-typed if for some contexts $\Gamma$ and $\Delta$ it holds that $\Gamma ; \Delta \vdash N$ or $\Gamma ; \Delta \vdash P$, respectively.

Since our contexts are lists and not sets, we employ structural rules to manage contexts. The rules of note are [TNode] and [TSRes]. Other rules are fairly standard and generalise to multiple partners quite straightforwardly (cf. 10). [TSRes] rule asserts that under a shared context and a linear context with exactly one parent session type $\bar{s} : T$ for session channel $s$, and a, possibly empty, sequence of session types for the children $s : T_1, \ldots, s : T_n$, the network $N$ is typed, then the network is typed by closing of the session. The parent session type is dual to all of the children session types, and furthermore $N$ has to be an alpha-variant such that the session channel $s$ is fresh for $\Delta, \Gamma$. The number of copies of children types $s : T$ are determined by the number of parallel components that participate in the session in $N$. This is ensured by the inductive invariant that rules preserve: whenever $\Gamma ; \Delta \vdash P$ for a process $P$, then all the channels are distinct in $\Delta$.

Note that [TSRes] uses the same type $T$; however, the nodes may have diverged to a different session state. The [TNode] accounts for the divergence. It states that for an (initial) context $\Delta$ that can be advanced by the session state
Definition 10 (Linear Context Inclusion). Linear context inclusion \( \Delta \subseteq \Delta' \) relation is defined inductively as follows:

\[
\begin{align*}
\Delta, \kappa : T & \subseteq \Delta', \kappa : T \\
\Delta, \kappa : T & \subseteq \Delta', \kappa \neq \kappa' \\
\varepsilon & \subseteq \Delta
\end{align*}
\]

Lemma 1 (Congruence Invariance). If \( N \equiv N' \), then \( \Gamma ; \Delta \vdash N \) if, and only if, \( \Gamma ; \Delta \vdash N' \).

Proof. By induction on the derivation of \( N \equiv N' \). The full proof is given in Appendix A.1.
Lemma 2 (Substitution). Whenever $\Gamma \vdash e : \beta$ and $\Gamma, x : \beta; \Delta \vdash P$, then $\Gamma; \Delta \vdash P\{e/x\}$.

Proof. By induction on the depth of derivation of $\Gamma, x : \beta; \Delta \vdash P$. Since substitutions only affect variables and expressions, the result easily follows from the requirements on expression substitution of Definition 9.

Theorem 1 (Subject Reduction). If $\Gamma; \Delta \vdash N$ and $N \rightarrow_G N'$, then there exist $\Delta' \subseteq \Delta$ such that $\Gamma; \Delta' \vdash N'$.

Proof. By induction on the depth of derivation of $N \rightarrow_G N'$. The full proof is given in Appendix A.2.

Note that a linear context does not change in terms of types that it describes, but only that it may reduce in size. The reason is twofold. First, because [TNode] advances the type in accordance with the sessions state of a node, the same type from $\Delta$ may be chosen as the initial type in $\Delta'$ for each session channel. Second, a node may recover during the reduction (due to rules [RRecover] and [RRecoverSel]) and hence dropping all the sessions it participated in and in turn these sessions are discarded from the initial context $\Delta$. Also note that subject reduction property is unaffected by the structure of the connectivity graph $G$.

The subject reduction theorem still holds if we allow recovery for any process not just input prefix, but we need to modify [TSRes] rule to possibly exclude $\bar{s}$ from the context.

Definition 11 (Error Network). Let s-prefix be a network of the following form

- SCT = $[\bar{s}\langle e \rangle, P_1 \triangleright R_1 | \psi_1]_{l_1}$
- GTH = $[\bar{s}\ell, P_2 \triangleright R_2 | \psi_2]_{l_2}$
- SEL = $[\bar{s} \oplus \ell, P_3 \triangleright R_3 | \psi_3]_{l_3}$
- RCV = $[\bar{s}\ell, P_4 \triangleright R_4 | \psi_4]_{l_4}$
- SND = $[\bar{s}\langle e' \rangle, P_5 \triangleright R_5 | \psi_5]_{l_5}$
- BR = $[\bar{s} \& \{\ell_1 : Q_1, \ldots, \ell_n : Q_n\} \triangleright R_6 | \psi_6]_{l_6}$

such that all above session stores are in the same state with regard to $s$, that is, for some $m$ and for $k = 1, \ldots, 6$, we have cnt$(s, \psi_k) = m$.

An invalid s-redex is one of the following parallel compositions of s-prefixes:

- SCT | SCT
- SCT | GTH
- GTH | SCT
- GTH | SEL
- SEL | RCV
- RCV | SND
- SND | BR
- SCT | BR

A network $N$ is called an error network whenever there exists an invalid s-redex $M$ such that for some network $N'$ the following holds

$N \equiv (\nu \bar{n}, s)(N' | M)$. 
Equivalently, a valid network is a parallel composition of nodes of which all s-prefixes that are in the same session state, consists of at most one scatter prefix (resp. gather, selection) and many receive prefixes (resp. send, branch).

**Theorem 2 (Type Safety).** Let $\Gamma; \Delta \vdash N$ and $N \rightarrow^*_G N'$ then $N'$ is not an error network.

**Proof.** The proof follows easily from Theorem 1 and the fact that an error network is not well-typed.

Type safety states that a well-typed network has always a possibility of communication. A well-typed network never reduces to a state where the session stores are in the same state but there is a prefix mismatch.

### 4 Data Aggregation in a Wireless Sensor Network

We express an aggregation protocol in a wireless sensor network using our framework. A wireless sensor network consists of spatially distributed nodes that sense environment data. In this example, we use the aggregation function $\text{max}$ that computes the maximum of two integers, e.g., temperature of the environment.

The node called sink initiates a data collection algorithm in the network by requesting the aggregated values of its neighbouring nodes. It then disseminates the maximum of these values to the network by sending the aggregate again to its neighbours. A node that receives a request then: (1) proceeds in turn to initiate the same process as the sink by requesting aggregated values from its neighbourhood; (2) aggregates the received values with its own value and sending it to the request node; (3) receives the maximum value originating from the sink; and (4) forwards the maximum value to its neighbours.

The above interaction can be described abstractly from the initiator node perspective by the session type

$$T = \gamma \text{int},! \text{int}.end.$$

That is, first it requests and receives an aggregate value set from its neighbours, and then proceeds by sending the maximum value to its neighbours. Nodes then participate in two sessions: a session that they accept from the initiator, and a session that they initiate themselves to start the aggregation. The former is described by the dual type $\overline{T}$ and the latter by $T$.

Let a set of expressions $E$ be defined by the following:

$$e, e' ::= x \mid \text{max}(e, e') \mid n \quad (n \in \mathbb{N}, x \in \mathcal{X}).$$

Let the set of base types $\mathcal{B}$ be $\{\text{int}\}$. Then, define the expected typing judgements for expressions $\Gamma, x : \text{int} \vdash x : \text{int}$, $\Gamma \vdash n : \text{int}$ for any $n \in \mathbb{N}$, and if $\Gamma \vdash e : \text{int}$ and $\Gamma \vdash e' : \text{int}$, then $\Gamma \vdash \text{max}(e, e') : \text{int}$. The set of conditions $\mathcal{F}$ is empty. We define the aggregation operation $e \odot e'$ as $\text{max}(e, e')$, and the unit $1$ as $0 \in \mathbb{N}$.
We implement sink and nodes as the following processes according to the above description:

\[
\text{SINK} = a_1(s).\bar{s}_?x.\bar{s}_!(x).0 \\
\text{NODE}_i = a_i(s).a_i(s_c).\bar{s}_?x.\bar{s}_!(\max(x,v_i)).\bar{s}_y(y).\bar{s}_c!(y).0
\]

where \(v_i \in \mathbb{N}\) for \(i = 2, 3\). For simplicity, let us model a three node network. Let the set of locations be \(L = \{1, 2, 3\}\), and the connectivity graph \(G\) be \(\{(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)\}\) that we visually depict as

\[
\begin{array}{c}
2 \\
\downarrow \\
\text{1} \\
\uparrow \\
\text{3}
\end{array}
\]

The network \(N\) implementing the aggregation protocol is then the following

\[
[SINK \triangleright SINK | \varepsilon]_1 \mid [\text{NODE}_2 \triangleright \text{NODE}_2 | \varepsilon]_2 \mid [\text{NODE}_3 \triangleright \text{NODE}_3 | \varepsilon]_3
\]

where the recovery is simply a restart of the process. The network is well-typed:

\[
a : T \triangleright \varepsilon \vdash N.
\]

It is fairly easy to see that \(\text{SINK}\) is well-typed after applying [TPar] with [TReq]

\[
a : T ; s : \text{int} \triangleright \text{int.end} \vdash \bar{s}_?x.\bar{s}_!(x).0
\]

and then type of the session channel \(\bar{s}\) matches the process. Each node can be typed similarly, however, we have two sessions by applying [TAcc] and [TReq]

\[
a : T ; s : \text{int} \triangleright \text{int.end}, \bar{s}_c : \text{int} \triangleright \text{int.end} \vdash \bar{s}_?x.\bar{s}_!(\max(x,v_i)).\bar{s}_y(y).\bar{s}_c!(y).0
\]

note that the type for \(s\) is dual to that of \(\bar{s}\) of sink above and that the interactions on two sessions match both of the types. This holds by easy applications of [TSnd] and [TRcv].

Thus, the operation of the network \(N\) is safe. Let us take an example of a run of the protocol from \(N\) over \(G\) where \(N\) first reduces to the following by establishing a shared session between the sink and the nodes 2 and 3

\[
(\nu s)([\bar{s}_?x.\bar{s}_!(x).0 \triangleright SINK | \varepsilon]_1 \mid N_2 \mid N_3)
\]

where, for \(k \in 2, 3\), \(N_k = [a_i(s_c).\bar{s}_?x.\bar{s}_!(\max(x,v_k)).\bar{s}_y(y).\bar{s}_c!(y).0 \triangleright \text{NODE}_k | \varepsilon]_k\)

Then, the nodes 2 and 3 in turn request a session of their own [RInit], but since there are no other process accepting a session (neither the sink, nor the nodes themselves) they establish a session with just themselves:

\[
N'_k = (\nu s_k)([\bar{s}_?x.\bar{s}_!(\max(x,v_k)).\bar{s}_y(y).\bar{s}_c!(y).0 \triangleright \text{NODE}_k | \varepsilon]_k
\]

for \(k = 1, 2\). The nodes then again in turn proceed to gather [RGather]. However, as there are no partners they simply use the unit 0 as the result and emit \(s_k\) to their session state:

\[
N'_k = (\nu s_k)([s_(\max(0,v_k)).\bar{s}_y(y).\bar{s}_c!(y).0 \triangleright \text{NODE}_k | s_k]_k
\]
for \( k = 1, 2 \). The sink may then proceed the gather process and we obtain the network \( N' \) where the sink is ready to disseminate the maximum value

\[
(v, s, s_2, s_3) ([\hat{s}_i(\max(\max(0, v_2), \max(0, v_3))), \circledast]_0 \triangleright \text{Sink} \mid s)_1 \mid [s_2(y).\hat{s}_2'(y), \circledast]_0 \triangleright \text{Node}_2 \mid s_2 \mid s)_2 \mid [s_3'(y).\hat{s}_3'(y), \circledast]_0 \triangleright \text{Node}_3 \mid s_3 \mid s)_3
\]

At this point the network is still well-typed \( a : \mathcal{T} \varepsilon \vdash N' \). After applications of [TSRes] we can type with the context \( \Delta = \bar{s} : \mathcal{T}, s : \mathcal{T}, \bar{s}_2 : \mathcal{T}, \bar{s}_3 : \mathcal{T} \). And for example the sink types with \([\text{TN} \circ \text{Node}]\), after splitting the context \( \Delta \) with \([\text{TPAR}]\), by advancing the context with the session state \( \psi = s \) as \( \bar{s} : \text{int.} \text{int. end} \). Now, suppose only node 2 receives the scatter from the sink.

\[
(v, s, s_2, s_3) ([\circledast]_0 \triangleright \text{Sink} \mid s)_1 \mid [s_2((\max(\max(0, v_2), \max(0, v_3))), \circledast]_0 \triangleright \text{Node}_2 \mid s_2 \mid s)_2 \mid [\circledast]_0 \triangleright \text{Node}_3 \mid s_3 \mid s)_3
\]

The node 2 can proceed by broadcasting its received result to ether, and after that node 3 recovers giving us one of the final states of the protocol:

\[
(v, s, s_2, s_3) ([\circledast]_0 \triangleright \text{Sink} \mid s)_1 \mid [\circledast]_0 \triangleright \text{Node}_2 \mid s_2 \mid s)_2 \mid \text{Node}_3 \triangleright \text{Node}_3 \mid \varepsilon)_3.
\]

The configuration of the network is well-typed; the session stores are consistent with the initial type \( \mathcal{T} \). Even with failure to receive a message the runtime of the protocol is successful, there are no communication errors and the communication is best effort as intended.

This simple example has a terminal state. In order to implement continuous aggregation we can simply redefine the network with recursion as usual and is typed by the same context \( a : \mathcal{T} \varepsilon \). For example, we can redefine \( \text{Sink} \) to be \( \text{Sink} \circ \text{Rec} = \mu X. a(s, \text{\hat{s}}_i(x), \text{\hat{s}}_i(x), X) \).

Suppose that the sink is defined instead with session prefixes swapped as \( \text{Sink}' = a(s, \text{\hat{s}}_i(\nu_0), \text{\hat{s}}_i(x)). \circledast \). That is, it is not well-typed \( a : \mathcal{T} \varepsilon \not\vdash \text{Sink}' \). Then, the network still reduces, but no communication takes place; furthermore, communication is not even possible between nodes and the sink. Let us take a minimal but sufficient example to illustrate. The network \([\text{Sink}' \triangleright \text{Sink}' \mid \varepsilon)_1 \mid [\text{Node} \triangleright \text{Node} \mid \varepsilon)_2\) reduces to the following, by first establishing a session between the sink and node, and node requests a session and vacuously gathers:

\[
(v, s, s') ([\hat{s}_i(\nu_0), \hat{s}_i(x)], \circledast) \triangleright \text{Sink}' \mid \varepsilon)_1 \mid [s_1(\max(0, v_i), s_2(y), \text{\hat{s}}'_1(y)], \circledast) \triangleright \text{Node} \mid s'_2)
\]

Even though the two nodes are in the same session state with regard to session \( s \), there is communication mismatch; this is an error network in the sense of Definition [11]. The sink will scatter its value without any possibility of the node receiving it, the node will then lose its message by [Rloss]

\[
(v, s, s') ([\text{\hat{s}}_i(x)], \circledast) \triangleright \text{Sink}' \mid \varepsilon)_1 \mid [s_1(y), \text{\hat{s}}'_1(y)], \circledast) \triangleright \text{Node} \mid s'_2)
\]

and then will need to recover as both nodes are expecting a receive:

\[
(v, s, s') ([\text{\hat{s}}_i(x)], \circledast) \triangleright \text{Sink}' \mid \varepsilon)_1 \mid [\text{Node} \triangleright \text{Node} \mid \varepsilon)_2\).
Thus, sink proceeds successfully while node recovered and no communication occurs:

\[(\nu s, s')([0 \triangleright \text{Sink}\' | \varepsilon]_1 | [\text{Node} \triangleright \text{Node} | \varepsilon]_2)\].

This illustrates the point that our type system does not only guarantee progress, which can always be achieved by assuming that messages are lost and using the recovery processes. It guarantees that progress occurs through communication.

5 Conclusions

We have introduced a process calculus with unreliable broadcast communication and a session type system that guarantees safe communication. We have captured two common operations in broadcasting systems with unreliable communication: scatter and gather. Our calculus can tolerate message loss and has the ability to recover. We believe that this sufficiently captures the requirements of systems that are based on unreliable broadcast communication, such as wireless sensor networks.

Reliable broadcasting semantics were proposed in the form of multicasting in [7]. Types for reliable gather semantics were proposed in [6] but an implementation was never proposed. This is the first time broadcasting and gather are presented in the context of unreliability and message loss. A form of recovery as exception handling was introduced in binary [5] and multiparty [4] session types, that defines a complicated procedure for informing and coordinating a set of session endpoints about an exception. The recovery procedure in the above work assumes strong global synchronisation requirements, among the session endpoints. In contrast, our semantics allows for each process to autonomously recover from failure of communication. We believe that our recovery semantics are more natural and general because they can account for local session failures as well. Hüttel and Pratas [12] investigate expressiveness of a similar process calculus with scatter and gather operations.

We are the first to consider a session type system for unreliable communication. Furthermore, we are the first to introduce notions of locality and channel connectivity in session types, putting forward the requirements for developing a session type system in the presence of node distribution.

The system presented here is based on a form of binary session types. As seen in Section 4, in order to implement wireless network algorithms we need to interleave session channels. A future research direction is to develop a more robust multiparty session type system where network roles can interact with multiple roles inside a session. Furthermore, we would like to investigate more elaborate autonomous recovery mechanisms that would allow a node to continue in a session instead of restarting. We also plan to extend the system with node mobility that is important in wireless sensor networks. Node mobility would allow nodes to migrate between locations, introduce and disable nodes in a network. Because of autonomous recovery mechanism, we believe that adding node mobility to our system should not pose difficulties.
Acknowledgements Kouzapas and Gay are supported by the UK EPSRC project “From Data Types to Session Types: A Basis for Concurrency and Distribution” (EP/K034413/1). This research was supported by a Short-Term Scientific Mission grant from COST Action IC1201 (Behavioural Types for Reliable Large-Scale Software Systems).

References

A Proofs

A.1 Congruence invariance proof

Proof (of Lemma 1). We proceed by induction on the derivation of $N \equiv N'$:

case $N \mid M \equiv M \mid N$. Let us first show the $\Rightarrow$ direction. Assume $\Gamma; \Delta \vdash N \mid M$.

\[
\frac{\Gamma; \Delta_1 \vdash N \quad \Gamma; \Delta_2 \vdash M}{\Gamma; \Delta_1, \Delta_2 \vdash N \mid M}[\text{TPAR}]
\]

where $\Delta_1, \Delta_2$ is a permutation of $\Delta$ obtained by the exchange rule. We use $\Gamma; \Delta_1 \vdash N$ and $\Gamma; \Delta_2 \vdash M$ to construct

\[
\frac{\Gamma; \Delta_2 \vdash M \quad \Gamma; \Delta_1 \vdash N}{\Gamma; \Delta_2, \Delta_1 \vdash M \mid N}[\text{TPAR}]
\]

Obviously, $\Delta_2, \Delta_1$ is a permutation of $\Delta$. We are done with this case. The other direction is analogous.

case $(M \mid N) \mid S \equiv M \mid (N \mid S)$. Assume $\Gamma; \Delta \vdash (M \mid N) \mid S$. Then the derivation tree of the assumption is

\[
\frac{\Gamma; \Delta_1 \vdash M \quad \Gamma; \Delta_2 \vdash N \quad \Gamma; \Delta_3 \vdash S}{\Gamma; \Delta_1, \Delta_2, \Delta_3 \vdash (M \mid N) \mid S}[\text{TPAR}]
\]

where $\Delta_1, \Delta_2, \Delta$ is a permutation of $\Delta$. Then, using the leafs of the above tree we obtain

\[
\frac{\Gamma; \Delta_1 \vdash M \quad \Gamma; \Delta_2 \vdash N \quad \Gamma; \Delta_3 \vdash S}{\Gamma; \Delta_2, \Delta_3 \vdash M \mid N}[\text{TPAR}]
\]

\[
\frac{\Gamma; \Delta_1 \vdash M \quad \Gamma; \Delta_2 \vdash N \quad \Gamma; \Delta_3 \vdash S}{\Gamma; \Delta_1, \Delta_2, \Delta_3 \vdash (M \mid N) \mid S}[\text{TPAR}]
\]

We are done with this case. The opposite direction is analogous.

case $(\nu a)N \mid M \equiv (\nu a)(N \mid M)$. Assume $\Gamma; \Delta \vdash (\nu a)N \mid M$. From assumption we obtain the following tree

\[
\frac{\Gamma, a : T; \Delta_1 \vdash N \quad a \# \Gamma}{\Gamma, a : T; \Delta_1, \Delta_2 \vdash (\nu a)N \mid M}[\text{TCRes}]
\]

\[
\frac{\Gamma, a : T; \Delta_1 \vdash N \quad a \# \Gamma \quad \Gamma; \Delta_2 \vdash M}{\Gamma, a : T; \Delta_1, \Delta_2 \vdash (\nu a)(N \mid M)}[\text{TPAR}]
\]

We then derive the tree where we apply the weakening rule $[\text{TShWeak}]$ since by assumption $a \# M$:

\[
\frac{\Gamma; \Delta_1 \vdash N \quad \Gamma; \Delta_2 \vdash M}{\Gamma, a : T; \Delta_1, \Delta_2 \vdash N \mid M}[\text{TPAR}]\quad a \# \Gamma
\]

\[
\frac{\Gamma, a : T; \Delta_1 \vdash N \quad \Gamma; \Delta_2 \vdash M}{\Gamma, a : T; \Delta_1, \Delta_2 \vdash N \mid M}[\text{TPAR}]\quad \Gamma; \Delta_1, \Delta_2 \vdash (\nu a)(N \mid M)[\text{TCRes}]
\]

The other direction is similar because again by assumption we may use the weakening rule to obtain the right premise for the $[\text{TCRes}]$ rule.
case $(\nu s)N \mid M \equiv (\nu s)(N \mid M)$. Assume $\Gamma; \Delta \vdash (\nu s)N \mid M$. From the assumption we obtain the following tree
\[
\frac{\Gamma; \Delta_1, \hat{s} : T, s : T, \ldots, s : T \vdash N \quad s \not\in \Gamma, \Delta_1}{\Gamma; \Delta_1 \vdash (\nu s)N} \quad \text{[TSRES]} \quad \frac{\Gamma; \Delta_2 \vdash M}{\Gamma; \Delta_1, \Delta_2 \vdash (\nu s)N \mid M} \quad \text{[TPAR]}
\]
We construct the following by noting that $s \not\in \Gamma, \Delta_1, \Delta_2$ implies $s \not\in \Gamma, \Delta_1$.
Note that in the tree before applying [TPAR] we use the exchange rule.
\[
\frac{\Gamma; \Delta_1, \hat{s} : T, s : T, \ldots, s : T \vdash N \quad \Gamma; \Delta_2 \vdash M}{\Gamma; \Delta_1, \Delta_2, \hat{s} : T, s : T, \ldots, s : T \vdash N \mid M} \quad \text{[TECH]} \quad \frac{s \not\in \Gamma, \Delta_1, \Delta_2}{\Gamma; \Delta_1, \Delta_2 \vdash (\nu s)(N \mid M)} \quad \text{[TSRES]}
\]
The opposite direction is similar: as $s \not\in M$ there is nothing to consume $s$ on $M$ and we can only split the context as above when applying the [TPAR] rule.

\underline{case} $N \equiv [0 \triangleright 0 \mid \varepsilon] \mid N$. Assume $\Gamma; \Delta \vdash N$. Then, the following is a derivation tree
\[
\frac{\Gamma; \varepsilon \vdash 0 \quad \text{[TINACT]}}{\Gamma; \varepsilon \vdash 0} \quad \frac{\Gamma; \varepsilon \vdash 0 \quad \text{[TINACT]}}{\Gamma; \varepsilon \vdash 0} \quad \frac{\Gamma; \varepsilon \vdash 0 \quad \text{[TINODE]}}{\Gamma; \Delta \vdash N} \quad \frac{\Gamma; \Delta \vdash N \mid N}{\Gamma; \Delta \vdash [0 \triangleright 0 \mid \varepsilon] \mid N} \quad \text{[TPAR]}
\]
The other direction is analogous.

\underline{case} $[P \triangleright Q \mid \psi] \equiv [P' \triangleright Q' \mid \psi]$. This follows from the fact that the structural congruence preserves the free names.

\underline{case} $\mu X.P \equiv P\{\mu X.P / X\}$. Assume $\Gamma; \Delta \vdash \mu X.P$. From assumption we obtain the following tree
\[
\frac{\Gamma; X : \Delta; \Delta \vdash X \quad \text{[TVAR]}}{\Gamma; X : \Delta; \Delta \vdash X} \quad \frac{\Gamma; X : \Delta; \Delta \vdash \mu X.P \quad X \not\in \Gamma}{\Gamma; \Delta \vdash \mu X.P} \quad \text{[TREC]}
\]
We replace the leafs of the form [TVAR] with the above tree and also substitute in the derivation tree $X$ for $\mu X.P$ as follows
\[
\frac{\Gamma; \Delta \vdash \mu X.P}{\Gamma; X : \Delta; \Delta \vdash \mu X.P} \quad \text{[TSHWMEAK]} \quad \frac{\Delta \vdash \mu X.P / X \quad \text{[TSHWMEAK]}}{\Gamma; X : \Delta; \Delta \vdash P\{\mu X.P / X\}} \quad \frac{\Gamma; \Delta \vdash P\{\mu X.P / X\}}{\Gamma; \Delta \vdash P\{\mu X.P / X\}}
\]
In the last derivation step, we use a straightforward lemma: $\Gamma; X : \Delta; \Delta \vdash P$ and $X \not\in \Gamma$, then $\Gamma; \Delta \vdash X$. It is easy to see that the above tree satisfies the rules of Fig. 2 as needed. We also used the fact that $X \not\in \Gamma$ to add $X : \Delta$ to the context. The opposite direction is similar where we collapse the tree.
A.2 Subject reduction proof

**Definition 12.** Define the distinct predicate inductively as follows on linear contexts:

\[
\frac{\#(\Delta) \neq \#(\Delta, \kappa : T)} {\#(\Delta, \kappa : T) \neq \#(\varepsilon)}
\]

We make use of the following lemma that says whenever a process (NB: not a network) is well-typed, then all the channels are distinct in the linear context.

**Lemma 3.** If \( \Gamma; \Delta \vdash P \), then \( \#(\Delta) \).

**Proof.** By induction on the derivation of \( \Gamma; \Delta \vdash P \).

Now we are ready to prove the subject reduction property of our system.

**Proof (of Theorem 1).** By induction on the depth of derivation of \( N \rightarrow^* N' \).

- **case [RLoss].** Assume \([s_1(\varepsilon), Q \triangleright R | \psi | _l] \rightarrow_G [Q \triangleright R | \psi | _l] \) and \( \Gamma; \Delta \vdash [s_1(\varepsilon), Q \triangleright R | \psi | _l] \). From the assumption we derive the following tree:

\[
\Delta \rightsquigarrow \psi \Delta', s : \beta.T \quad \Gamma; \Delta', s : \beta.T \vdash s_1(\varepsilon).Q \\
\Gamma \vdash R
\]

By Lemma 3 we know that \( s \not\# \Delta' \), thus \( \Delta \rightsquigarrow^* \Delta', s : T \). Then, we can type the reduct as follows:

\[
\Delta \rightsquigarrow^* \Delta', s : T \quad \Gamma; \Delta', s : T \vdash Q \\
\Gamma; \Delta \vdash [Q \triangleright R | \psi | _l]
\]

- **case [RSscatter].** Assume \([s_1(\varepsilon), P \triangleright R | \psi | _l] \rightarrow_G [P \triangleright R | \psi | _l] \) and \( \bigwedge_{i \in I} [s_i(x_i), Q_i \triangleright R_i | \psi_i | _l] \). Without loss of generality, for convenience, suppose \( I = \{1, \ldots, n\} \).

We obtain the following derivation tree:

\[
\Delta_0 \rightsquigarrow^* \Delta'_0, \tilde{s} : \beta.T \quad \Gamma; \Delta'_0, \tilde{s} : \beta.T \vdash \tilde{s}_1(\varepsilon).P \\
\Gamma \vdash R
\]

\[
\Delta_0, \Delta_1, \ldots, \Delta_n \vdash [\tilde{s}_1(\varepsilon).P \triangleright R | \psi | _l] \bigwedge_{i \in I} [s_i(x_i), Q_i \triangleright R_i | \psi_i | _l]
\]

where \( \Delta_0, \Delta_1, \ldots, \Delta_n \) is a permutation of \( \Delta \), and \( \Delta'_0, \tilde{s} : T \) is \( \Delta_0 \), and \( D \) is as follows where we use [TPAR] to split the linear contexts.

\[
\Delta_1 \rightsquigarrow^* \Delta'_1, s : \beta.T' \\
\Gamma; \Delta_1, s : \beta.T' \vdash Q_1 \\
\Gamma \vdash R_1
\]

\[
\bigwedge_{i \in I} [s_i(x_i), Q_i \triangleright R_i | \psi_i | _l]
\]

\[
\Delta_1, ..., \Delta_n \bigwedge_{i \in I} [s_i(x_i), Q_i \triangleright R_i | \psi_i | _l]
\]

\[
\Gamma; \Delta_1, ..., \Delta_n \bigwedge_{i \in I} [s_i(x_i), Q_i \triangleright R_i | \psi_i | _l]
\]

\[
\Gamma \vdash D'
\]
where $D''$ is the following

$$
D' \vdash \Delta_2, \ldots, \Delta_n \vdash \prod_{i \in I \setminus \{1\}} \beta_i(x_i).Q_i \triangleright R_i \mid \psi_i |_{l_i}
$$

We iterate $[\text{TPar}]$ and $[\text{TNode}]$ rules on $D'$ to obtain the premises: $\Gamma; x_i : \beta; \Delta'_0, \bar{s} : T \vdash Q_i$ for $i \in I$.

Observe that $\bar{s} \neq \Delta'_0$ by Lemma 3. Thus, type advancement only affects $s$, i.e., $\Delta_0 \rightarrow^\psi \Delta'_0, \bar{s} : T$.

Then, the following is a typing derivation

$$
\Delta, \beta, \Delta'_0, \bar{s} : T \vdash \Gamma, x : \beta; \Delta'_0, \bar{s} : T \vdash P \quad \Gamma, \bar{s} : T \vdash R \quad \Gamma \vdash \Delta_1, \ldots, \Delta_n \vdash \prod_{i \in I \setminus \{1\}} \beta_i(x_i).Q_i \triangleright R_i \mid \psi_i |_{l_i},
$$

$[\text{TPar}]$

The cases for $D''$ are similar but we use the substitution Lemma 2 where appropriate.

**Case $[\text{RGather}]$.** For convenience let $I = \{1, \ldots, n\}$. From an assumption we obtain the following tree:

$$
\Delta_0 \rightarrow^\psi \Delta'_0, \bar{s} : \gamma, \beta, T
$$

$$
\Delta_0, \Delta_1, \ldots, \Delta_n \vdash \prod_{i \in I \setminus \{1\}} \beta_i(x_i).Q_i \triangleright R_i \mid \psi_i |_{l_i},
$$

$[\text{TNode}]$

$D$ is the following

$$
\Delta_1 \rightarrow^\psi \Delta'_1, s : \gamma, \beta, T
$$

$$
\Delta_1, \ldots, \Delta_n \vdash \prod_{i \in I \setminus \{1\}} \beta_i(x_i).Q_i \triangleright R_i \mid \psi_i |_{l_i},
$$

By Lemma 3 from $\Gamma; \Delta'_0, \bar{s} : T$ we get that $\bar{s} \neq \Delta'_0$, thus $\Delta \rightarrow^\psi \Delta'_0, \bar{s} : T$ since the advancement has no affect on $\Delta'_0$. From above we have that $\Gamma \vdash \Delta'_0, \bar{s} : T \vdash P$ and previous fact, we get $\Gamma; \Delta'_0, \bar{s} : T \vdash P[e/x]$. By iterating $[\text{TPar}]$ and $[\text{TNode}]$ rules on $D'$, we get assumptions: $\Delta_0 \rightarrow^\psi \Delta'_0, \bar{s} : \gamma, \beta, T$, and $\Gamma; \Delta'_1, s : T' \vdash Q_i$, and $\Gamma \vdash \Delta'_1, s : T' \vdash Q_i$, and $\Gamma \vdash \Delta'_1, s : T' \vdash Q_i$, and $\Gamma \vdash \Delta'_1, s : T' \vdash Q_i$, and $\Gamma \vdash \Delta'_1, s : T' \vdash Q_i$, and $\Gamma \vdash \Delta'_1, s : T' \vdash Q_i$, and $\Gamma \vdash \Delta'_1, s : T' \vdash Q_i$, and $\Gamma \vdash \Delta'_1, s : T' \vdash Q_i$, and $\Gamma \vdash \Delta'_1, \ldots, \Delta_n \vdash \prod_{i \in I \setminus \{1\}} \beta_i(x_i).Q_i \triangleright R_i \mid \psi_i |_{l_i}$;

Hence, we can derive the first half:

$$
\Delta_0 \rightarrow^\psi \Delta'_0, \bar{s} : T
$$

$$
\Gamma; \Delta'_0, \bar{s} : T \vdash P[e/x] \quad \Gamma; \bar{s} : T \vdash R \quad \Gamma \vdash \Delta_1, \ldots, \Delta_n \vdash \prod_{i \in I \setminus \{1\}} \beta_i(x_i).Q_i \triangleright R_i \mid \psi_i |_{l_i},
$$

$[\text{TPar}]$

$D''$ is

$$
\Delta_1 \rightarrow^\psi \Delta'_1, s : T
$$

$$
\Delta_1, \ldots, \Delta_n \vdash \prod_{i \in I \setminus \{1\}} \beta_i(x_i).Q_i \triangleright R_i \mid \psi_i |_{l_i},
$$

$[\text{TPar}]$
and $D''$ is simply an iteration of the above. We obtain $\Delta_1 \rightarrow^s \Delta'_1$, $s : T'$ by a similar consideration as above with Lemma 3. Other premises are straightforward.

case [RRes] We first consider the case when the restricted name is a session channel. Thus, we have

$$
\frac{\Gamma; \Delta, \bar{s} : T, s : T, \ldots, s : T \vdash N \quad s \# \Gamma; \Delta}{\Gamma; \Delta \vdash (\nu s)N} \text{[TSRes]}
$$

By induction hypothesis, we have that for any $\Delta''$ there is $\Delta''' \subseteq \Delta''$ such that $\Gamma; \Delta''' \vdash N'$. In particular, $\Delta'' = \Delta, \bar{s} : T, s : T, \ldots, s : \bar{T}$. Thus,

$$
\frac{\Gamma; \Delta'' \vdash N'}{\Gamma; \Delta \vdash (\nu s)N'} \text{[TSRes]}
$$

The case when the restricted name is a shared channel, follows immediately from the induction hypothesis by using the [TCRes] rule.

case [RPar] We have the tree

$$
\frac{\Gamma; \Delta_1 \vdash N_1 \quad \Gamma; \Delta_2 \vdash N_2}{\Gamma; \Delta_1, \Delta_2 \vdash N_1 | N_2} \text{[TPar]}
$$

By induction hypothesis, for any $\Gamma$, we have $\Gamma; \Delta_1 \vdash N_1$ and $N_1 \rightarrow_G N'_1$, and, for some $\Delta'_1$, s.t. $\Delta'_1 \subseteq \Delta_1$ and $\Gamma; \Delta'_1 \vdash N'_1$. Hence, we derive, as required, the following:

$$
\frac{\Gamma; \Delta'_1 \vdash N'_1 \quad \Gamma; \Delta_2 \vdash N_2}{\Gamma; \Delta'_1, \Delta_2 \vdash N'_1 | N_2} \text{[TPar]}
$$

since $\Delta'_1, \Delta_2 \subseteq \Delta_1, \Delta_2$.

case [RRecover] We have

$$
\frac{\Delta \rightarrow^\psi \Delta' \quad \Gamma; \Delta' \vdash s_\ell(x).P \quad \Gamma; \varepsilon \vdash R}{\Gamma; \Delta \vdash [s_\ell(x).P \triangleright R | \psi]_l} \text{[TNode]}
$$

Note $\varepsilon \subseteq \Delta$, thus

$$
\frac{\varepsilon \rightarrow^\psi \varepsilon \quad \Gamma; \varepsilon \vdash R \quad \Gamma; \varepsilon \vdash R}{\Gamma; \varepsilon \vdash [R \triangleright R | \varepsilon]_l} \text{[TNode]}
$$

case [RRecoverSel] We have

$$
\frac{\Delta \rightarrow^\psi \Delta' \quad \Gamma; \Delta' \vdash s \& \{\ell_i : Q_i\}_{i \in I} \quad \Gamma; \varepsilon \vdash R}{[s \& \{\ell_i : Q_i\}_{i \in I} \triangleright R | \psi]_l} \text{[TNode]}
$$

Note $\varepsilon \subseteq \Delta$, thus

$$
\frac{\varepsilon \rightarrow^\psi \varepsilon \quad \Gamma; \varepsilon \vdash R \quad \Gamma; \varepsilon \vdash R}{\Gamma; \varepsilon \vdash [R \triangleright R | \varepsilon]_l} \text{[TNode]}
$$
case [RTrue] and [RFalse]. Straightforward.

case [RCong] Follows from Lemma [1]

case [RInit]

\[ \Delta_0 \rightarrow^{\psi} \Delta', \Gamma, a : T ; \Delta', s : T \vdash P \]
\[ \Gamma, a : T ; \Delta ; P \vdash R | \psi | ]_i \]
\[ \hat{D} \]

\[ \hat{D}' \]

\[ \hat{D}'' \]

Note that \( s \neq \Delta_0, \Delta_1, \ldots, \Delta_n \) and \( s \neq \psi, \psi \). In the above we chose \( s \) accordingly. Thus, we can derive:

\[ \Delta_0, s : T \rightarrow^{\psi} \Delta', s : T \]
\[ \Gamma, a : T ; \Delta', s : T \vdash P \]
\[ \Gamma, a : T ; \Delta ; P \vdash R | \psi | ]_i \]
\[ \hat{D}'" \]

\[ \hat{D} \]

\[ \hat{D}'' \]

The rest of the tree \( \hat{D}'' \) is derived analogously.

case [RSelect] We have the following tree, for \( j \in J \):

\[ \Delta_0 \rightarrow^{\psi} \Delta_0, s : \{ \ell_j : T_j \}_{j \in J} \]
\[ \Gamma, \Delta_0', s : \{ \ell_j : T_j \}_{j \in J} \vdash P \]
\[ \Gamma, s : \ell_j, P \vdash R | \psi | ]_i \]
\[ \hat{D} \]

\[ \hat{D}' \]

\[ \hat{D}'' \]

where \( \hat{D} \) is as follows where \( J_1 = \{ 1, \ldots, n \} \) for some \( n \).

\[ \hat{D}'" \]

\[ \hat{D} \]

\[ \hat{D}'' \]

where \( \hat{D}'' \) is the following

\[ \Delta_1 \rightarrow^{\psi} \Delta_1, s : \{ \ell_j : T_j \}_{j \in J_1} \]

We iterate [TPAR], [TNode] and [TBr] to obtain the rest of assumptions in \( \hat{D}' \), namely, for all \( i \in I, \) that \( \Delta_i \rightarrow^{\psi} \Delta_i', s : \{ \ell_j : T_j \}_{j \in J_i} \), and \( \Gamma, \Delta_j', s : T_j \vdash Q_j, \) for all \( j \in J_i \), and \( \Gamma, \varepsilon \vdash R_i \).
By Lemma 3, we have $s \not\equiv \Delta'_{0}$, thus

$$
\frac{\Delta_0 \rightarrow \psi | s \Delta'_0, \bar{s} : T_j \quad \Delta'_0, \bar{s} : T_j \vdash P \quad \Gamma ; \varepsilon \vdash R}{\Gamma ; \Delta_0 \vdash [P \triangleright R \mid \psi | s]_l} \quad D''$

where $(\ell : Q_i) \in B_i$; we use the same idea to build the rest of the tree $D''$. Let us demonstrate this for the second parallel component $Q_1$:

$$
\frac{\Delta_1 \rightarrow \psi_1 | s \Delta'_1, s : T_k \quad \Delta'_1, s : T_k \vdash Q_1 \quad \Gamma ; \varepsilon \vdash R_1}{\Gamma ; \Delta_1 \vdash [Q_1 \triangleright R_1 \mid \psi_1 | s]_l} \quad [\text{TNode}]
$$

where $(\ell : T_k) \in \{\ell_j : T_j\}_{j \in J_1}$ and $(\ell : Q_1) \in \{\ell_j : Q_j\}_{j \in J_1}$. The other cases are similar.