Session Types for Broadcasting

Dimitrios Kouzapas
University of Glasgow
dimitrios.kouzapas@glasgow.ac.uk

Ramūnas Gutkovas
Uppsala University
ramunas.gutkovas@it.uu.se

Simon J. Gay
University of Glasgow
simon.gay@glasgow.ac.uk

Up to now session types have been used under the assumptions of point to point communication, to ensure the linearity of session endpoints, and reliable communication, to ensure send/receive duality. In this paper we define a session type theory for broadcast communication semantics that by definition do not assume point to point and reliable communication. Our session framework lies on top of the parametric framework of broadcasting ψ-calculi, giving insights on developing session types within a parametric framework. Our session type theory enjoys the properties of soundness and safety. We further believe that the solutions proposed will eventually provide a deeper understanding of how session types principles should be applied in the general case of communication semantics.

1 Introduction

Session types [5, 7, 6] allow communication protocols to be specified as types and verified by type-checking. Up to now, session type systems have assumed reliable, point to point message passing communication. Reliability is important to maintain send/receive duality, and point to point communication is required to ensure session endpoint linearity.

In this paper we propose a session type system for unreliable broadcast communication. Developing such a system was challenging for two reasons: (i) we needed to extend binary session types to handle unreliability as well as extending the notion of session endpoint linearity, and (ii) the reactive control flow of a broadcasting system drove us to consider typing patterns of communication interaction rather than communication prefixes. The key ideas are (i) to break the symmetry between the s^+ and s^- endpoints of channel s, allowing s^+ (uniquely owned) to broadcast and gather, and s^- to be shared; (ii) to implement (and type) the gather operation as an iterated receive. We retain the standard binary session type constructors.

We use ψ-calculi [1] as the underlying process framework, and specifically we use the extension of the ψ-calculi family with broadcast semantics [2]. ψ-calculi provide a parametric process calculus framework for extending the semantics of the π-calculus with arbitrary data structures and logical assertions. Expressing our work in the ψ-calculi framework allows us to avoid defining a new operational semantics, instead defining the semantics of our broadcast session calculus by translation into a broadcast ψ-calculus. Establishing a link between session types and ψ-calculi is therefore another contribution of our work.

Intuition through Demonstration. We demonstrate the overall intuition by means of an example. For the purpose of the demonstration we imply a set of semantics, which we believe are self explanatory. Assume types $S = !T; ?T; \mathtt{end}$, $\overline{S} = ?T; !T; \mathtt{end}$ for some data type T, and typings $s^+ : S$, $s^- : \overline{S}$, $a : \langle S \rangle$, $v : T$. The session type prefix $!T$ means *broadcast* when used by s^+, and *single destination send* when used by s^-. Dually, $?T$ means *gather* when used by s^+, and *single origin receive* when used by s^-.

Session Initiation through broadcast, creating an arbitrary number of receiving endpoints:

$\overline{a}s^-.P_0 \mid ax.P_1 \mid ax.P_2 \mid ax.P_3 \longrightarrow P_0 \mid P_1\{s^-/x\} \mid P_2\{s^-/x\} \mid ax.P_3$

Submitted to:

Due to unreliability, $ax.P_3$ did not initiate the session. We denote the initiating and accepting session endpoint as s^+ and s^- respectively.

Session Broadcast from the s^+ endpoint results in multiple s^- endpoints receiving:

$$s^+!\langle v\rangle;P_0 \mid s^-?(x);P_1 \mid s^-?(x);P_2 \mid s^-?(x);P_3 \longrightarrow P_0 \mid P_1\{v/x\} \mid P_2\{v/x\} \mid s^-?(x);P_3$$

Due to unreliability, a process (in the above reduction, process $s^-?(x);P_3$) might not receive a message. In this case the session endpoint that belongs to process $s^-?(x);P_3$ is considered broken, and later we will introduce a recovery mechanism.

Gather: The next challenge is to achieve the sending of values from the s^- endpoints to the s^+ endpoint. The gather prefix $s^+?(x);P_0$ is translated (in Section 4) into a process that iteratively receives messages from the s^- endpoints, non-deterministically stopping at some point and passing control to P_0.

$$s^+?(x);P_0 \mid s^-!\langle v_1\rangle;P_1 \mid s^-!\langle v_2\rangle;P_2 \mid s^-!\langle v_3\rangle;P_3 \longrightarrow^* P_0' \mid P_1 \mid s^-!\langle v_2\rangle;P_2 \mid P_3$$

with $P_0\{\{v_1,v_3\}/x\} \longrightarrow P_0'$.

After two reductions the messages from processes $s^-!\langle v_1\rangle;P_1$ and $s^-!\langle v_3\rangle;P_3$ had been received by the s^+ endpoint. On the third reduction the s^+ endpoint decided not to wait for more messages and proceeded with its session non-deterministically, resulting in a broken sending endpoint ($s^-!\langle v_2\rangle;P_2$), which is predicted by the unreliability of the broadcast semantics. The received messages, v_1 and v_2, were delivered to P_0 as a set.

Prefix Enumeration: The above semantics, although capturing broadcast session initiation and interaction, still violate session type principles due to the unreliability of communication:

$$\begin{array}{l} s^+!\langle v_1\rangle;s^+!\langle v_2\rangle;\mathbf{0} \mid s^-?(x);s^-?(y);\mathbf{0} \mid s^-?(x);s^-?(y);\mathbf{0} \quad\longrightarrow \\ \qquad\qquad s^+!\langle v_2\rangle;\mathbf{0} \mid s^-?(y);\mathbf{0} \mid s^-?(x);s^-?(y);\mathbf{0} \quad\longrightarrow\quad \mathbf{0} \mid \mathbf{0} \mid s^-?(y);\mathbf{0} \end{array}$$

The first reduction produced a broken endpoint, $s^-?(x);s^-?(y);\mathbf{0}$, while the second reduction reduces the broken endpoint. This situation is not predicted by session type principles. To solve this problem we introduce an enumeration on session prefixes:

$$\begin{array}{l} (s^+,1)!\langle v_1\rangle;(s^+,2)!\langle v_2\rangle;\mathbf{0} \mid (s^-,1)?(x);(s^-,2)?(y);\mathbf{0} \mid (s^-,1)?(x);(s^-,2)?(y);\mathbf{0} \longrightarrow \\ (s^+,2)!\langle v_2\rangle;\mathbf{0} \mid (s^-,2)?(y);\mathbf{0} \mid (s^-,1)?(x);(s^-,2)?(y);\mathbf{0} \longrightarrow \mathbf{0} \mid \mathbf{0} \mid (s^-,1)?(x);(s^-,2)?(y);\mathbf{0} \end{array}$$

The intuitive semantics described in this example are encoded in the ψ-calculi framework. From this it follows that all the operational semantics, typing system and theorems are stated using the ψ-calculus framework.

Contributions. This paper is the first to propose session types as a type meta-theory for the ψ-calculi. Applying session semantics in such a framework meets the ambition that session types can effectively describe general communication semantics. A step further is the development of a session type framework for broadcast communication semantics. It is the first time that session types escape the assumptions of point to point communication and communication reliability. We also consider as a contribution the fact that we use enumerated session prefixes in order to maintain consistency of the session communication. We believe that this technique will be applied in future session type systems that deal with unreliable and/or unpredictable communication semantics.

Related Work. Carbone *et al.* [4] extended binary session types with exceptions, allowing both parties in a session to collaboratively handle a deviation from the standard protocol. Capecchi *et al.* [3] generalized a similar approach to multi-party sessions. In contrast, our recovery processes allow a broadcast sender or receiver to autonomously handle a failure of communication. Although it might be possible to represent broadcasting in multi-party session type systems, by explicitly specifying separate messages from a single source to a number of receivers, all such systems assume reliable communication for every message.

2 Broadcast Session Calculus

We define an intuitive syntax for our calculus. The syntax below will be encoded in the ψ-calculi framework so that it will inherit the operational semantics.

$$\begin{array}{lll} P,R & ::= & \overline{a}s^-.P \mid ax.P \mid s^+!\langle v\rangle;P \mid s^-!\langle v\rangle;P \mid s^+?(x);P \mid s^-?(x);P \\ & \mid & s^+ \oplus l;P \mid s^-\&\{l_i : P_i\} \mid P \bowtie R \mid \mathbf{0} \mid \mu X.P \mid X \mid P \,|\, P \mid (\nu\, n)P \end{array}$$

Processes $\overline{a}s^-.P$, $ax.P$ are prefixed with session initiation operators that interact following the broadcast semantics. Processes $s^+!\langle v\rangle;P$, $s^-!\langle v\rangle;P$ define two different sending patterns. For the s^+ endpoint we have a broadcast send. For the s^- endpoint we have a unicast send. Processes $s^+?(x);P$, $s^-?(x);P$ assume gather (i.e. the converse of broadcast send) and unicast receive, respectively. We allow selection and branching $s^+ \oplus l;P$, $s^-\&\{l_i : P_i\}$ only for broadcast semantics from the s^+ to the s^- endpoint. Each process can carry a recovery process R with the operator $P \bowtie R$. The process can proceed nondeterministically to recovery if the session endpoint is broken due to the unreliability of the communication. Process R is carried along as process P reduces its prefixes. The rest of the processes are standard π-calculus processes.

Structural congruence is defined over the abelian monoid defined by the parallel operator ($|$) and the inactive process ($\mathbf{0}$) and additionally satisfies the rules:

$$(\nu\, n)\mathbf{0} \equiv \mathbf{0} \qquad P \mid (\nu\, n)Q \equiv (\nu\, n)(P \mid Q) \text{ if } n \notin \mathtt{fn}(P)$$

3 Broadcast ψ-Calculi

Here we define the parametric framework of ψ-calculi for broadcast. For a detailed description of ψ-calculi we refer the reader to [1].

We fix a countably infinite set of names $\mathcal{N}$ ranged over by a,b,x. ψ-calculi are parameterised over three nominal sets: terms ($\mathbf{T}$ ranged over by M,N,L), conditions ($\mathbf{C}$ ranged over by φ), and assertions ($\mathbf{A}$ ranged over by Ψ); and operators: channel equivalence, broadcast output and input connectivity $\dot{\leftrightarrow}, \dot{\prec}, \dot{\succ}$: $\mathbf{T} \times \mathbf{T} \to \mathbf{C}$, assertion composition $\otimes : \mathbf{A} \times \mathbf{A} \to \mathbf{A}$, unit $\mathbf{1} \in \mathbf{A}$, entailment relation $\vdash \mathbf{A} \times \mathbf{C}$, and a substitution function substituting terms for names for each set. The channel equivalence is required to be symmetric and transitive, and assertion composition forms abelian monoid with $\mathbf{1}$ as the unit element. We do not require output and input connectivity be symmetric, i.e., $\Psi \vdash M \dot{\prec} N$ is not equivalent to $\Psi \vdash N \dot{\succ} M$, however for technical reasons require that the names of L should be included in N and M whenever $\Psi \vdash N \dot{\prec} L$ or $\Psi \vdash L \dot{\succ} M$. The agents are defined as follows

$$P,Q \;::=\; M(\lambda\widetilde{a})N.P \mid \overline{M}N.P \mid \mathbf{case}\,\varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n \mid (\!|\Psi|\!) \mid (\nu a)P \mid P \,|\, Q \mid\, !P$$

where $\widetilde{a}$ bind into N and P. The assertions in the case and replicated agents are required to be guarded. We abbreviate the case agent as $\mathbf{case}\,\widetilde{\varphi} : \widetilde{P}$; we write $\mathbf{0}$ for $(\!|\mathbf{1}|\!)$, we also write $a\#X$ to intuitively mean that name a does not occur freely in X.

We give a brief intuition behind the communication parameters: Agents unicast whenever their subject of their prefixes are channel equivalent, to give an example, $\overline{M}L.P$ and $N(\lambda\widetilde{a})K.Q$ communicate whenever $\Psi \vdash M \dot{\leftrightarrow} N$. In contrast, broadcast communication is mediated by a broadcast channel, for example, the agents $\overline{M}N.P$ and $M_i(\lambda\widetilde{a_i})N_i.P_i$ (for $i > 0$) communicate if they can broadcast and receive from the same channel $\Psi \vdash M \dot{\prec} K$ and $\Psi \vdash K \dot{\succ} M_i$.

In addition to the standard structural congruence laws of pi-calculus we define the following, with the assumption that $a \# \widetilde{\varphi}, M, N, \widetilde{x}$ and π is permutation of a sequence.

$$\begin{array}{cc} (\nu a)\mathbf{case}\,\widetilde{\varphi} : \widetilde{P} \equiv_\Psi \mathbf{case}\,\widetilde{\varphi} : \widetilde{(\nu a)P} & \mathbf{case}\,\widetilde{\varphi} : \widetilde{P} \equiv_\Psi \mathbf{case}\,\pi \cdot (\widetilde{\varphi} : \widetilde{P}) \\ \overline{M}N.(\nu a)P \equiv_\Psi (\nu a)\overline{M}N.P & M(\lambda\widetilde{x})N.(\nu a)P \equiv_\Psi (\nu a)M(\lambda\widetilde{x})N.P \end{array}$$

The following is a reduction context with two types of numbered holes (condition hole $\hat{[]}$ and process hole $[]$) such that no two holes of the same type have the same number.

$$C \ ::= \ (\mathbf{case}\,\hat{[]}_j : C \,[]\, \widetilde{\varphi} : \widetilde{P}) \mid C \quad | \quad \textstyle\prod_{k>0}[]_{i_k}$$

The filling of the holes is defined in the following way: filling a process (resp. condition) hole with a assertion guarded process (resp. condition) taken from the number position of a given sequence. We denote filling of holes as $C[(\varphi_i)_{i\in I};(P_j)_{j\in J};(Q_k)_{k\in K}]$ where the first component is for filling the condition holes and the other two are for filling process holes.

We require that I is equal to the numbering set of condition holes and furthermore J and K are disjoint and their union is equal to the numbering set of context for the process holes. We also require that every J numbered hole is either in parallel with any of the K holes or is parallel to **case** where recursively a K numbered hole can be found. When the numbering is understood we simply write $C[\widetilde{\varphi};\widetilde{P};\widetilde{Q}]$.

In the following we define reduction semantics of ψ-calculi, in addition to the standard labelled transition semantics [1]. The two rules describe unicast and broadcast semantics. We identify agents up to structural congruence, that is, we also assume the rule such that two agents reduce if their congruent versions reduce. In the broadcast rule, if for some $a \in \widetilde{a}$, $a \in n(K)$, then $\widetilde{b} = \widetilde{a}$, otherwise $\widetilde{b} = \widetilde{a} \setminus n(N)$. To simplify the presentation we abbreviate $\prod \widetilde{(\!|\Psi|\!)}$ as $(\!|\hat{\Psi}|\!)$ and $\otimes_i \Psi_i$ as $\hat{\Psi}$. We prove that reductions correspond to silent and broadcast transitions.

$$\frac{N' = N[\widetilde{x} := \widetilde{L}] \text{ and } \hat{\Psi} \vdash M \leftrightarrow M' \text{ and } \forall i.\hat{\Psi} \vdash \varphi_i}{(\nu\widetilde{a})(C[\widetilde{\varphi};\ \widetilde{R};\ M(\lambda\widetilde{x})N.P, \overline{M'}N'.Q] \mid (\!|\hat{\Psi}|\!)) \quad \to \quad (\nu\widetilde{a})(P[\widetilde{x} := \widetilde{L}] \mid Q \mid \prod \widetilde{R} \mid (\!|\hat{\Psi}|\!))}$$

$$\frac{\hat{\Psi} \vdash M \mathrel{\dot{\prec}} K \text{ and } \forall i.\hat{\Psi} \vdash K \mathrel{\dot{\succ}} M'_i \text{ and } N'_i[\widetilde{x}_i := \widetilde{L}_i] = N \text{ and } \forall j.\hat{\Psi} \vdash \varphi_j}{(\nu\widetilde{a})(C[\widetilde{\varphi};\ \widetilde{R};\ \overline{M}N.P, \widetilde{(M'(\lambda\widetilde{x})N'.Q)}] \mid (\!|\hat{\Psi}|\!)) \quad \to \quad (\nu\widetilde{b})(P \mid \prod_i Q_i[\widetilde{x}_i := \widetilde{L}_i] \mid \prod \widetilde{R} \mid (\!|\hat{\Psi}|\!))}$$

Theorem 3.1. *Let α be either a silent or broadcast output action. Then, $\mathbf{1} \triangleright P \xrightarrow{\alpha} P'$ iff $P \to P'$*

Proof Sketch. The complicated direction is $\Longrightarrow$. One needs to prove similar results for the other actions, and then demonstrate that they in parallel have the right form. □

4 Translation of Broadcast Calculus to Broadcast ψ-Calculus

The semantics for the broadcast session calculus are given as an instance of the ψ-calculi with broadcast [2]. To achieve this effect we define a translation between the syntax of § 2 and a particular instance of the ψ-calculi. Operational semantics are then inherited by the ψ-calculi framework.

We fix the set of labels $\mathscr{L}$ and ranged over by $l, l_1, l_2 \ldots$. The following are the nominal sets

$$\begin{array}{lcl}
\mathbf{T} & = & \mathscr{N} \cup \{*\} \cup \{(n^p,k),(n^p,i),(n^p,k,\mathsf{u}),(n^p,l,k), n\cdot k \mid n,k \in \mathbf{T} \wedge i \in \mathbb{N} \wedge l \in \mathscr{L} \wedge p \in \{+,-\}\} \\
\mathbf{C} & = & \{t_1 \leftrightarrow t_2, t_1 \mathrel{\dot{\prec}} t_2, t_1 \mathrel{\dot{\succ}} t_2 \mid t_1,t_2 \in \mathbf{T}\} \cup \{\mathsf{true}\} \\
\mathbf{A} & = & \mathbf{T} \to \mathbb{N}
\end{array}$$

We define the $\otimes$ operator (here defined as multiset union) and the $\vdash$ relation:

$$(f \otimes g)(n) = \begin{cases} f(n)+g(n) & \text{if } n \in dom(f) \cap dom(g) \\ f(n) & \text{if } n \in dom(f) \\ g(n) & \text{if } n \in dom(g) \\ \text{undefined} & \text{otherwise} \end{cases}
\quad\Bigg|\quad
\begin{array}{lcl}
\Psi & \vdash & (s^{p_1},k,\mathsf{u}) \leftrightarrow (s^{p_2},j,\mathsf{u}) \text{ iff } \Psi(k) = \Psi(j) \\
\Psi & \vdash & (s^+,k) \mathrel{\dot{\prec}} (s^+,i) \text{ iff } \Psi(k) = i \\
\Psi & \vdash & (s^+,i) \mathrel{\dot{\succ}} (s^-,k) \text{ iff } \Psi(k) = i \\
\Psi & \vdash & \mathsf{true} \qquad \Psi \vdash a \leftrightarrow a \in \mathscr{N}
\end{array}$$

It can be easily checked that the definition is indeed a broadcast ψ-calculus. We write $\Sigma_{i\in I}P$ as a shorthand for **case** $\widetilde{\mathtt{true}} : \widetilde{P}$, and $P+Q$ for **case** $\mathtt{true} : P \mathbin{[]} \mathtt{true} : Q$

The translation is parameterised by ρ, which tracks the enumeration of session prefixes, represented by multisets of asserted names $\llparenthesis k \rrparenthesis$. The replication in $s^+?(x,u^i);P$ implements the iterative broadcast receive. We annotated the prefixes $s^+?(x,u^i);^b P$ and $\mu X^b.P \bowtie R$ with $b \in \{0,1\}$ to capture their translation as a two step (0 and 1) iterative process. The recovery process can be chosen in a non-deterministic way instead of a s^- prefix. Otherwise it is pushed in the continuation of the translation.

$$
\begin{array}{l}
\llbracket \overline{a}s^-.P \bowtie R \rrbracket_\rho = (\nu k)(\overline{a}s^-.\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}}) \qquad\qquad \llbracket ax.P \bowtie R \rrbracket_\rho = (\nu k)(a(\lambda x)x.\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^-:k\}}) \\
\llbracket s^+!\langle v\rangle;P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}} = \overline{(s^+,k)}v.(\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}} \mid \llparenthesis k \rrparenthesis) \\
\llbracket s^-!\langle v\rangle;P \bowtie R \rrbracket_{\rho\cup\{s^-:k\}} = \overline{(s^-,k,\mathsf{u})}v.(\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^-:k\}} \mid \llparenthesis k \rrparenthesis) + \llbracket R \rrbracket_{\rho\cup\{s^-:k\}} \\
\quad \begin{array}{rl}
\llbracket s^+?(x,u);^0 P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}} = & (\nu\, n)(\overline{n}u.\mathbf{0} \mid !(n(\lambda x)x.((s^+,k,\mathsf{u})(\lambda y)y.\overline{n}(x\cdot y).\mathbf{0}) \\
& +\tau.(\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}} \mid \llparenthesis k \rrparenthesis))) \\
\llbracket s^+?(x,u);^1 P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}} = & (\nu\, n)((((s^+,k,\mathsf{u})(\lambda y)y.\overline{n}(u\cdot y).\mathbf{0}) + \tau.(\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}}[x:=u] \mid \llparenthesis k \rrparenthesis) \\
& \mid !(n(\lambda x)x.((s^+,k,\mathsf{u})(\lambda y)y.\overline{n}(x\cdot y).\mathbf{0}) + \tau.(\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}} \mid \llparenthesis k \rrparenthesis)))
\end{array} \\
\llbracket s^-?(x);P \bowtie R \rrbracket_{\rho\cup\{s^-:k\}} = (s^-,k)(\lambda x)x.(\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^-:k\}} \mid \llparenthesis k \rrparenthesis) + \llbracket R \rrbracket_{\rho\cup\{s^-:k\}} \\
\llbracket s^+ \oplus l;P \bowtie R \rrbracket_{\rho\cup\{s^-:k\}} = \overline{(s^+,l,k)}*.(\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}} \mid \llparenthesis k \rrparenthesis) \qquad \llbracket \llparenthesis k \rrparenthesis \rrbracket_{\rho\cup\{s^p:k\}} = \llparenthesis k \rrparenthesis \\
\llbracket s^-\&\{l_i : P_i\}_{i\in I} \bowtie R \rrbracket_{\rho\cup\{s^-:k\}} = \Sigma_{i\in I}(s^-,l_i,k)(\lambda)*.(\llbracket P_i \bowtie R \rrbracket_{\rho\cup\{s^-:k\}} \mid \llparenthesis k \rrparenthesis) + \llbracket R \rrbracket_{\rho\cup\{s^-:k\}} \\
\llbracket \mu X^0.P \bowtie R \rrbracket_\rho = (\nu\, n)(!(n(\lambda)*.\llbracket P \bowtie R \rrbracket_{\rho\cup\{X:n\}}) \mid \overline{n}*.\mathbf{0}) \\
\llbracket \mu X^1.P \bowtie R \rrbracket_\rho = (\nu\, n)(\llbracket P \bowtie R \rrbracket_{\rho\cup\{X:n\}} \mid !(n(\lambda)*.\llbracket P \bowtie R \rrbracket_{\rho\cup\{X:n\}})) \\
\llbracket X \rrbracket_{\rho\cup\{X:n\}} = \overline{n}*.\mathbf{0} \quad \llbracket \mathbf{0} \rrbracket_\rho = \mathbf{0} \quad \llbracket \mathbf{0} \bowtie R \rrbracket_\rho = \mathbf{0} \quad \llbracket P \mid Q \rrbracket_\rho = \llbracket P \rrbracket_\rho \mid \llbracket Q \rrbracket_\rho \quad \llbracket (\nu\, n)P \rrbracket_\rho = (\nu n)\llbracket P \rrbracket_\rho
\end{array}
$$

The encoding respects the following desirable properties.

Lemma 4.1 (Encoding Properties)**.** Let P be a session broadcast process.

1. $\llbracket P[x:=v] \rrbracket = \llbracket P \rrbracket[x:=v]$
2. $\llbracket P \rrbracket \to Q$ implies that for a session broadcast process $P', Q \equiv_\Psi \llbracket P' \rrbracket$.

5 Broadcast Session Types

Broadcast session types syntax is identical to classic binary session type syntax (cf. [7]), with the exception that we do not allow session channel delegation. We assume the duality relation as defined in [7]. Note that we do not need to carry the session prefix enumeration in the session type system or semantics. Session prefix enumeration is used operationaly only to avoid communication missmatch.

$$
\begin{array}{lcl}
S & ::= & !U;S \mid ?U;S \mid \oplus\{l_i : S_i\}_{i\in I} \mid \&\{l_i : S_i\}_{i\in I} \mid \mathtt{end} \mid \mathsf{X} \mid \mu\mathsf{X}.S \\
U & ::= & \langle S \rangle \mid [U]
\end{array}
$$

Typing judgements are: $\Gamma \vdash P$ read as P is typed under environment Γ, with

$$
\Delta \ ::= \ \emptyset \mid \Delta\cdot s^p : S \qquad\qquad \Gamma \ ::= \ \emptyset \mid \Gamma\cdot a : \langle S\rangle \mid \Gamma\cdot s^p : S \mid \Gamma\cdot\mathsf{X} : \Delta
$$

Δ environments map only session names to session types, while Γ maps shared names to shared types, session names to session types and process variables to Δ mappings.

The rules below define the broadcast session type system:

$$
\Gamma\cdot n : U \vdash n : U \ \text{[Name]} \qquad \frac{\Gamma \vdash P \quad s^p \notin \mathtt{fn}(P)}{\Gamma\cdot s^p : \mathtt{end} \vdash P} \ \text{[Weak]} \qquad \frac{s \notin \mathtt{dom}(\Gamma)}{\Gamma \vdash \mathbf{0} \ \text{[Inact]}} \qquad \frac{\Gamma \vdash R \quad s^p \notin \mathtt{dom}(\Gamma)}{\Gamma \vdash \mathbf{0} \bowtie R} \ \text{[Recov]}
$$

$$\frac{\Gamma \vdash a : \langle S \rangle \quad \Gamma \vdash s^+ : S \quad \Gamma \vdash P}{\Gamma \cdot s^- : \overline{S} \vdash \overline{a}s^-.P}\,[\mathrm{BInit}] \qquad \frac{\Gamma \vdash a : \langle S \rangle \quad \Gamma \cdot x : \overline{S} \vdash P}{\Gamma \vdash ax.P}\,[\mathrm{BAcc}]$$

$$\frac{\Gamma \cdot s^+ : S \vdash P \bowtie R \quad \Gamma \vdash v : \langle S' \rangle}{\Gamma \cdot s^+ :!\langle S' \rangle;S \vdash s^+!\langle v \rangle;P \bowtie R}\,[\mathrm{BSend}] \qquad \frac{\Gamma \cdot s^- : S \vdash P \bowtie R \quad \Gamma \vdash v : \langle S' \rangle \quad s^- \notin \mathtt{dom}(\Gamma)}{\Gamma \cdot s^- :!\langle S' \rangle;S \vdash s^-!\langle v \rangle;P \bowtie R}\,[\mathrm{USend}]$$

$$\frac{\Gamma \cdot s^+ : S \cdot x : \langle S' \rangle \vdash P \bowtie R \quad \Gamma \vdash u : [\langle S' \rangle]}{\Gamma \cdot s^+ :?\langle S' \rangle;S \vdash s^+?(x,u);^b P \bowtie R}[\mathrm{URcv}] \qquad \frac{\Gamma \cdot s^- : S \cdot x : \langle S' \rangle \vdash P \bowtie R \quad s^- \notin \mathtt{dom}(\Gamma)}{\Gamma \cdot s^- :?\langle S' \rangle;S \vdash s^-?(x);P \bowtie R}\,[\mathrm{BRcv}]$$

$$\frac{\Gamma \cdot s^+ : S_k \vdash P \bowtie R \quad k \in I}{\Gamma \cdot s^+ : \oplus\{l_i : S_i\}_{i \in I} \vdash s^+ \oplus l_k;P \bowtie R}\,[\mathrm{Sel}] \qquad \frac{\Gamma \cdot s^- : S_i \vdash P_i \bowtie R \quad s^- \notin \mathtt{dom}(\Gamma)}{\Gamma \cdot s^- : \&\{l_i : S_i\}_{i \in I} \vdash s^- \&\{l_i : P_i\}_{i \in I} \bowtie R}\,[\mathrm{Bra}]$$

$$\frac{\Gamma_1 \vdash P_1 \quad \Gamma_2 \vdash P_2 \quad s^+ \notin \mathtt{dom}(\Gamma_1) \cap \mathtt{dom}(\Gamma_2)}{\Gamma_1 \cup \Gamma_2 \vdash P_1 \mid P_2}\,[\mathrm{Par}] \qquad \frac{\Gamma \cdot s^+ : S \cdot \{s^- : \overline{S_i}\}_{i \in I} \vdash P \quad S = S_i}{\Gamma \vdash (\nu\ s)P}\,[\mathrm{SRes}]$$

$$\frac{\Gamma \cdot a : \langle S \rangle \vdash P}{\Gamma \vdash (\nu\ a)P}\,[\mathrm{ShRes}] \qquad \frac{\Gamma \cup \Delta \cdot X : \Delta \vdash P \quad s^p \notin \mathtt{dom}(\Gamma)}{\Gamma \cup \Delta \vdash \mu X^b.P}\,[\mathrm{Rec}] \quad \Gamma \cup \Delta \cdot X : \Delta \vdash X\ [\mathrm{RVar}]$$

Rule [Recov] types the recovery process. We expect no free session names in a recover process. Rules [BInit], [BAcc], [BSend], [Usend], [BRcv], [BRcv], [Sel] and [Bra] type prefixes in the standard way, i.e. check for object and the subject type match. Rule [URcv] types both binary instances of the unicast receive prefix with the same type. We require that the recovery process is carried and typed inductively in the structure of a process. A recovery process must not (re)use any session endpoints ([Recov]). Also we require the s^- to be the only one in Γ. Multiple s^- endpoints are collected using the [Par] rule. The [Par] rule expects that there is no duplicate s^+ endpoint present inside a process. When restricting a session name we check endpoint s^+ and the set of endpoints s^- to have dual types. The rest of the rules are standard.

5.1 Soundness and Safety

We use the standard notion of a context $\mathscr{C}$ on session types S with a single hole denoted as $[]$. We write $\mathscr{C}[S]$ for filling a hole in C with the type S. We define the set of non-live sessions in a context as $d(\Gamma) = \{s^- : S \mid s^+ : S' \in \Gamma \text{ and } \overline{S} = C[S'] \text{ with } C \neq []\}$ and live $l(\Gamma) = \Gamma \setminus d(\Gamma)$. We say that Γ is well typed iff $\forall s^+ : S \in l(\Gamma)$ then $\{s^- : \overline{S_i}\}_{i \in I} \subset l(\Gamma)$ with $S = S_i$ or $S = ?U;S_i$.

Theorem 5.1 (Subject Congruence)**.** If $\Gamma \vdash P$ with Γ well typed and $P \equiv P'$ then $\Gamma \vdash P'$.

Theorem 5.2 (Subject Reduction)**.** If $\Gamma \vdash P$ with Γ well typed, $dom(\rho) \subseteq dom(\Gamma)$ and $[\![P]\!]_\rho \to Q$, then there is P' such that $[\![P']\!]_\rho \equiv_\Psi Q$, $\Gamma' \vdash P'$ and Γ' well typed with either $\Gamma' = d(\Gamma) \cup l(\Gamma')$ or $\Gamma' = d(\Gamma) \setminus \{s^- : S\} \cup l(\Gamma')$ or $\Gamma' = d(\Gamma) \cup \{s^- : S\} \cup l(\Gamma')$.

Definition 5.1 (Error Process)**.** Let s-prefix processes to have the following form:

1. $s^+!\langle v \rangle;P$ 2. $s^+ \oplus l;P$ 3. $s^+?(x);P$ 4. $\prod_{i \in I} s^-?(x);P_i \mid \prod_{j \in J} C_j[s^-?(x);P_j]$
5. $\prod_{i \in I} s^-!\langle v_i \rangle;P_i \mid \prod_{k \in K} P_k \mid \prod_{j \in J} C_j[s^-?(x);P_j]$
where $\prod_{i \in I} P_i \mid \prod_{k \in K} P_k \mid \prod_{j \in J} C_j[s^-?(x);P_j]$ forms an s-redex.
6. $\prod_{i \in I} s^- \&\{l_k : P_k\}_{k \in K_i} \mid \prod_{j \in J} C_j[s^- \&\{l_k : P_k\}_{k \in K_j}]$

with $C_j[]$ being a context that contains s^- prefixes.

A valid s-redex is a parallel composition of either s-prefixes 1 and 4, s-prefixes 2 and 6, or s-prefixes 3 and 5. Every other combination of s-prefixes is invalid. An error process is a process of the form $P \equiv (\nu\ \tilde{n})(R \mid Q)$ where R is an invalid s-redex and Q does not contain any other s-prefixes.

Theorem 5.3 (Type Safety)**.** A well typed process will never reduce into an error process.

Proof. The proof is a direct consequence of the Subject Reduction Theorem (5.2) since error process are not well typed. □

6 Conclusion

We have defined a system of session types for a calculus based on unreliable broadcast communication. This is the first time that session types have been generalised beyond reliable point-to-point communication. We defined the operational semantics of our calculus by translation into an instantiation of broadcast ψ-calculi, and proved subject reduction and safety results. The use of the ψ-calculi framework means that we can try to use its general theory of bisimulation for future work on reasoning about session-typed broadcasting systems. The definition of a session typing system is also a new direction for the ψ-calculi framework.

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

[1] Jesper Bengtson, Magnus Johansson, Joachim Parrow & Björn Victor (2009): *Psi-calculi: Mobile Processes, Nominal Data, and Logic*. In: *LICS*, pp. 39–48, doi:10.1109/LICS.2009.20.

[2] Johannes Borgström, Shuqin Huang, Magnus Johansson, Palle Raabjerg, Björn Victor, Johannes Åman Pohjola & Joachim Parrow (2011): *Broadcast Psi-calculi with an Application to Wireless Protocols.* In Gilles Barthe, Alberto Pardo & Gerardo Schneider, editors: *SEFM*, *Lecture Notes in Computer Science* 7041, Springer, pp. 74–89, doi:10.1007/978-3-642-24690-6_7.

[3] Sara Capecchi, Elena Giachino & Nobuko Yoshida (2014): *Global Escape in Multiparty Sessions*. *Mathematical Structures in Computer Science*. To appear.

[4] Marco Carbone, Kohei Honda & Nobuko Yoshida (2008): *Structured Interactional Exceptions in Session Types*. In: *CONCUR*, *LNCS* 5201, Springer, pp. 402–417, doi:10.1007/978-3-540-85361-9_32.

[5] Kohei Honda, Vasco T. Vasconcelos & Makoto Kubo (1998): *Language Primitives and Type Disciplines for Structured Communication-based Programming*. In: *ESOP'98*, *LNCS* 1381, Springer, pp. 22–138, doi:10.1007/BFb0053567.

[6] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty Asynchronous Session Types*. In: *POPL'08*, ACM, pp. 273–284, doi:10.1145/1328897.1328472.

[7] Nobuko Yoshida & Vasco Thudichum Vasconcelos (2007): *Language Primitives and Type Discipline for Structured Communication-Based Programming Revisited: Two Systems for Higher-Order Session Communication*. *Electr. Notes Theor. Comput. Sci.* 171(4), pp. 73–93, doi:10.1016/j.entcs.2007.02.056.