Session Subtyping and Multiparty Compatibility using Circular Sequents

Ross Horne
Computer Science, University of Luxembourg, Esch-sur-Alzette, Luxembourg
ross.horne@uni.lu

Abstract

We present a structural proof theory for multi-party sessions, exploiting the expressive power of non-commutative logic which can capture explicitly the message sequence order in sessions. The approach in this work uses a more flexible form of subtyping than standard, for example, allowing a single thread to be substituted by multiple parallel threads which fulfil the role of the single thread. The resulting subtype system has the advantage that it can be used to capture compatibility in the multiparty setting (addressing limitations of pairwise duality). We establish standard results: that the type system is algorithmic, that multiparty compatible processes which are race free are also deadlock free, and that subtyping is sound with respect to the substitution principle. Interestingly, each of these results can be established using cut elimination. We remark that global types are optional in this approach to typing sessions; indeed we show that this theory can be presented independently of the concept of global session types, or even named participants.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Proof theory; Theory of computation → Linear logic; Theory of computation → Process calculi

Keywords and phrases session types, subtyping, compatibility, linear logic, deadlock freedom

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.6

Acknowledgements This paper benefits hugely from discussions with Mariangiola Dezani-Ciancaglini and Paola Giannini who suggested restricting to a regular calculus.

1 Introduction

Session types are a class of type systems for modelling protocols that prescribe, not only the types of messages exchanged, but also the sequence in which they are communicated. The first session type systems were constrained to two parties. For such binary sessions, given a session type prescribing the behaviour of each of the participants, it is possible to determine whether the two behaviours are compatible, in the sense that they can interact together to successfully realise a protocol.

Here, in the introduction, we first make it clear there are obvious, underexploited, connections between compatibility in the binary setting and provability in non-commutative extensions of linear logic. The body of this work shows that these observations extend elegantly to the multiparty setting [32, 33], where multiparty compatibility is the problem of whether two or more participants realise a protocol when they communicate together.

On the binary setting and non-commutative logic. In the binary setting, compatibility holds when the two parties are dual to each other [30]. For example, $!\lambda_1;(?\lambda_2 \wedge ?\lambda_3)$ is dual to $?\lambda_1;(!\lambda_2 \vee !\lambda_3)$. The former types a process that is ready to output a message of type λ_1, and then receives either a message of type λ_2 or λ_3. The latter types a process that is ready to receive a message of type λ_1, and then makes a choice between two branches, sending a message of type λ_2 or λ_3. By building subtyping into the system [24, 23, 18], duality becomes a more flexible concept. For example, two processes of respective types $!\lambda_1;(?\lambda_2 \wedge ?\lambda_3)$ and $?\lambda_1;!\lambda_2$ are also compatible. Notice a process of the type $!\lambda_1;(?\lambda_2 \wedge ?\lambda_3)$ offers two possible inputs, so is more than capable of responding correctly to $?\lambda_1;!\lambda_2$, which

31st International Conference on Concurrency Theory (CONCUR 2020).
Editors: Igor Konnov and Laura Kovács; Article No. 6; pp. 6:1–6:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

always chooses to send λ_2 as its second action.

For binary sessions, compatibility is proven by showing that the dual of a type is a subtype of another type, for example establishing $!\lambda_1;(?\lambda_2 \wedge ?\lambda_3) \leq !\lambda_1;?\lambda_2$. In the original paper on session types [30], it was explicit that internal and external choice were inspired by the additive operators in linear logic [27, 26, 1]. For example, interpreting $\wedge$ as additive conjunction in linear logic, subtype relation $?\lambda_2 \wedge ?\lambda_3 \leq ?\lambda_2$ is a provable implication in linear logic. While pure linear logic has no concept of sequentiality (all operators are commutative), linear logic can be extended with non-commutative operators explicitly capturing sequentiality, allowing the above subtype judgement involving prefixing to be proven. In this work, we restrict ourselves to a fragment of non-commutative logic with action prefixing only, allowing us to retain a sequent calculus presentation. Full sequential composition can be achieved [34]. However, for full sequential composition, it is necessary [50] to employ the calculus of structures [28]. The compromise adopted in this work, of restricting non-commutative logic to prefixing, allows us to formulate our subtype system using the sequent calculus, whilst still working within a fragment of a conservative extension of linear logic.

Contribution to the multiparty setting. Using non-commutative extensions of linear logic to model multiparty session types provides additional expressive power. In particular, the subtype system obtained allows more session types to be compared than possible using established subtype systems [25]. Indeed the subtype system obtained is sufficiently rich, so that subtyping can be used to evaluate compatibility in the multi-party setting. The notion of *multiparty compatibility* enforced by this methodology allows session types to be used to guarantee that multiparty sessions are deadlock free without the need for a global type choreographing all processes. An advantage of avoiding global types is that we can check compatibility for protocols for which no global type exists [48].

Problems with pairwise duality resolved. Early work on multi-party session types [8, 21] employed a notion of compatibility based on the notion of duality for binary types applied pairwise. In that early work, we take each pair of participants and *restrict them only to the inputs and outputs between the participants selected*, and then check whether each pair of projections are dual. Pair-wise duality fails to guarantee deadlock freedom, since process $?\lambda_1;!\lambda_2 \parallel ?\lambda_2;!\lambda_3 \parallel ?\lambda_3;!\lambda_1$ deadlocks, despite participants being pair-wise dual (e.g., restricting the first two participants to their mutual communications gives types $!\lambda_2$ and $?\lambda_2$, which are dual). The process above consists of three participants in parallel each waiting to receive a message, from another process before producing an output. The process is clearly deadlocked since all inputs await a message that never arrives.

The current work, and related work [20, 15, 48, 39, 19], addresses the above limitation of pair-wise duality by proposing more sophisticated notions of *multi-party compatibility*. The work on which this builds [15] (which concerned a finite fragment of Scribble [31]), handles multiparty compatibility as a special case of subtyping. In this work, as required, our example processes in the previous paragraph would **not** be multi-party compatible. The rules of the system in this paper are determined by logical principles (cut-elimination).

Related paradigms. This paper does not follow the Curry-Howard inspired proofs-as-processes school; instead, it follows a processes-as-formulae [10, 36] approach closer to intersection types [44] and algorithmic subtyping [47]. For multiparty sessions, the processes-as-formulae [15] and proofs-as-processes paradigms [12, 11] emerged simultaneously. Papers following the Curry-Howard approach typically aim to design new (higher-order) session calculi where the process terms are proofs in an established logic. In contrast, in the processes-as-formulae approach pursued here, we typically harness the power of structural proof theory to design new logics that can directly embed established session calculi [17], while respecting their semantics. In this work, linear implication in the logical system introduced provides us with a notion of session subtyping preserving deadlock freedom.

Summary. In Section 2, we explain how the notion of multiparty subtyping is more flexible than established notions of subtyping for multiparty sessions, illustrated using an example where a participant is substituted by two participants. Section 3 formally develops a theory of session subtyping and multiparty compatibility in a coinductive sequent calculus. That section concludes with an example where we guarantee the deadlock freedom of a session for which no global type exists.

2 Motivating Example: A Generalised Substitution Principle

The problem of defining a subtype system for multiparty sessions is *in a sense* solved in the synchronous setting [14, 25]. Soundness in that work is defined according to a *substitution principle* [41], informally stated in related work [25] as: "If $\mathsf{T}\ \mathcal{R}\ \mathsf{T}'$, then a process of type T' engaged in a well-typed session may be safely replaced with a process of type T." Here $\mathcal{R}$ is a candidate subtype relation and "safely" is formalised in terms of deadlock freedom.

In the above related work, the substitution principle allows one (single threaded) participant to replace another participant. In the current paper, we take a broader interpretation of the substitution principle, permitting more parallelism to be introduced. We allow participants in a session to be replaced by any number of participants, e.g., a single thread of type T can be replaced by two parallel participants of type T_1 and T_2, where $\mathsf{T}_1 \otimes \mathsf{T}_2 \leq \mathsf{T}$. This allows parallel components to be introduced with additional communications, while preserving the ability of the multiple components to fulfil the role of the original components. An example is provided next.

An authorisation protocol. We provide an example that is out of scope of the substitution principle in related work mentioned above, but within the scope of the substitution principal in the current paper. In the example that follows, we consider an application where a *Trusted App* is replaced by an *Untrusted App* and an *OAuth Server*. This demands a rich multi-party subtype system accounting for parallelism and interactions.

Consider the protocol realised by the three participants in Fig. 1, which are modelled as threads in a typical session calculus. In this authorisation protocol, the *Trusted App* asks the *Owner* of a resource for permission before it accesses the *Resource*.

$$
\begin{array}{ll}
\textit{Owner:} & ?\mathit{login\_page}(\mathit{app\_ID}, \mathit{scope}); (!\mathit{deny} \oplus !\mathit{authorise}(\mathit{name}, \mathit{password})) \\[1ex]
\textit{Resource:} & \mathtt{rec}X.(?\mathit{release} + ?\mathit{request}(\mathit{token});(!\mathit{revoke} \oplus !\mathit{response}(\mathit{data});X)) \\[1ex]
\textit{Trusted App:} & !\mathit{login\_page}(\mathit{app\_ID}, \mathit{scope}); \\
& ?\mathit{deny};!\mathit{release} + \ ?\mathit{authorise}(\mathit{name}, \mathit{password}); \\
& \qquad\qquad \mathtt{rec}Y.!\mathit{release} \oplus \ !\mathit{request}(\mathit{token}); \\
& \qquad\qquad\qquad\qquad ?\mathit{revoke} + ?\mathit{response}(\mathit{data});Y
\end{array}
$$

Figure 1 The local behaviours of three participants in an authorisation protocol.

Owner: This could be you — the human user, who owns the resource. You get redirected to a login page containing the *app_ID* for the *Trusted App* and a *scope* indicating the resources requested (e.g., personal contact details). If you chose to approve authorisation, you grant access to the resource by providing your *name* and *password*. You do however have the ability to chose not to approve, choosing the branch $!deny$ in the **internal choice**, notated $\oplus$ in the process *Owner* in Fig. 1.

Resource: A token is used by the *Trusted App* to prove it has the right to access the resource. The *Resource* can be accessed many times by the *Trusted App* until the token expires or is revoked. The expiry of a token is modelled here by the *Resource* making an internal choice, deciding whether to provide data or revoke.

Trusted App: Since the App is trusted it presents directly the login page to the user. If the *Resource Owner* approves, the same App manufactures a token which is used to access the resource. Notice **external choice**, notated $+$, is used for inputs.

A problem with the above protocol is that user credentials are provided directly to the *Trusted App*. Furthermore, the *Trusted App* does not only know the credentials of the owner of the resource, it must also know how to manufacture tokens to access the resource; hence, in principle, has the right to freely access the resource without asking permission. Thus, there is no security offered to the *Resource Owner* or *Resource* if the app is compromised.

Substituting one participant with two participants. We can address the above limitation by making use of the OAuth 2.0 protocol [29] where handling of credentials and generation of tokens is handled by an *OAuth Server* that the *Owner* trusts more than the app. We can refine the above protocol by substituting *Trusted App* with two processes in parallel: an *Untrusted App* and *OAuth Server*, defined in Fig. 2.

$$
\begin{array}{l}
\textit{Untrusted App:}\\
\quad !initiate(add\_ID, scope);\\
\quad ?close + \;\; ?authorisation\_code(code);\\
\qquad\qquad\;\; !exchange(app\_ID, secret, code);\\
\qquad\qquad\;\; ?close + \;\; ?access\_token(token);\\
\qquad\qquad\qquad\qquad\;\; \mathtt{rec}Y.!request(token);(?revoke + ?response(data);Y)\\
\textit{OAuth Server:}\\
\quad ?initiate(app\_ID, scope);\\
\quad !login\_page(app\_ID, scope);\\
\quad (?deny;!close;!release) + \;\; ?authorise(name, password);\\
\qquad\qquad\qquad\qquad\qquad\quad (!close;!release) \oplus \;\; !authorisation\_code(code);\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ?exchange(app\_ID, secret, code);\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad (!close;!release) \oplus \;\; !access\_token(token)
\end{array}
$$

Figure 2 Two participants that can safely replace the *Trusted App* in Fig. 1

The OAuth protocol enables the *Untrusted App* to access the *Resource*, for which permission is required from the *Owner*, in such a way that the *Owner* never discloses their credentials to the *Untrusted App*. The *Owner* permits the *OAuth Server* to grant an access token to the *Untrusted App* that can be used to access the *Resource*. We briefly describe informally each process.

OAuth Server: As a mediator between the *Untrusted App* and *Resource Owner*, the *OAuth Server* receives an initiate request from the *Untrusted App*, resulting in the *Resource Owner* being redirected to a login page. Notice the *OAuth Server* reacts to the decision of the *Resource Owner* to either provide credentials or end the session, indicated by an *external* choice. Notice, after that point, that the server makes two internal choices: the first issuing a *code* to the *Untrusted App* only if the correct credentials were provided by the *Owner*; the second issuing an access token only if the *Untrusted App* provides its correct credentials (and the correct *code*). If all is correct, a *token* is eventually issued to the *Untrusted App*.

Untrusted App: The *Untrusted App* initiates the protocol. It then reacts, indicated by external choices, to whether the *Resource Owner* and *OAuth Server* grant access. If an access *token* is granted, the token can be used repeatedly to access the resource requested.

What the subtype system guarantees here. The *Trusted App* can be replaced by *Untrusted App* $\|$ *OAuth Server* while preserving deadlock freedom of the protocol. We know this because the type of *App* $\|$ *OAuth* is a subtype of the type of *Trusted App*, by using the subtype system introduce in the next section. Furthermore, for protocols of the complexity of this OAuth example, it is not immediately obvious whether all roles are correctly implemented such that deadlock freedom is guaranteed. We can also use the subtype system introduced in the next section to check whether participants together are multiparty compatible.

3 A Proof System for Subtyping and Multiparty Compatibility

In this section, we introduce session types and a proof system for expressing session types called Session, which defines our subtype system for multiparty sessions. Later in this section, having introduced Session, we define multiparty compatibility and race freedom, and use these properties to establish our main deadlock freedom result.

Session types are defined according to the following syntax. Note we could have propositional data types (*nat*, *bool*, etc.), but accommodating such data types is a perpendicular issue to this work, hence we simply label messages (λ_1, λ_2, etc.).

▶ **Definition 1** (session types). Session types for threads *are defined by:*

$$L \quad ::= \quad \bigwedge_{i\in I} ?\lambda_i;L_i \;\mid\; \bigvee_{i\in I} !\lambda_i;L_i \;\mid\; \mu\mathbf{t}.L \;\mid\; \mathbf{t} \;\mid\; \texttt{OK}$$

Session types for networks *are defined by:*

$$N \quad ::= \quad L \;\mid\; N \invamp N \;\mid\; N \otimes N$$

We refer to both of the above simply as session types*, which are ranged over by* T*,* U*,* V*. We restrict ourselves to guarded recursion, avoiding the type* $\mu\mathbf{t}.\mathbf{t}$*. Index sets* I *are finite.*

The constant $\texttt{OK}$ is used to type networks that, on all paths, either successfully terminate or progress forever. Intersection types (abbreviated as $\wedge$ when there are two branches) are used to type external choices between inputs; while union types (abbreviated as $\vee$) type internal choices between outputs.

Actions π are either of the form $!\lambda$ or $?\lambda$. Whenever there is only one branch in a union/intersection type, we simply write the action prefixed type $\pi;\mathsf{T}$, which is used to type a process that performs an input or output and then behaves as T. As standard, we allow $\texttt{OK}$ to be omitted, by abbreviating $\pi;\texttt{OK}$ as π.

Notably, the syntax features two commutative multiplicative operators $\invamp$ and $\otimes$. When typing multiparty sessions we employ only $\mathsf{T} \otimes \mathsf{U}$, representing two parallel sessions T and U that may communicate and interleave actions. The operator $\mathsf{T} \invamp \mathsf{U}$ is introduced to complete the theory, as the dual to parallel composition, and is used in subtyping proofs. Future work may also use $\invamp$ as an additional modelling device that prevents one session from interfering with another session. As a consequence of including the pair of multiplicatives, every session type, has a dual type, its co-type, given by the function below.

▶ **Definition 2** (co-type). *Co-types are defined by the following mapping over types, prefixed* *types and actions:*

$$\overline{\bigwedge_{i\in I} \mathsf{T}_i} = \bigvee_{i\in I} \overline{\mathsf{T}_i} \qquad \overline{\bigvee_{i\in I} \mathsf{T}_i} = \bigwedge_{i\in I} \overline{\mathsf{T}_i} \qquad \overline{\pi;\mathsf{T}} = \overline{\pi};\overline{\mathsf{T}} \qquad \overline{!\lambda} = ?\lambda \qquad \overline{?\lambda} = !\lambda$$

$$\overline{\mathsf{T} \otimes \mathsf{U}} = \overline{\mathsf{T}} \parr \overline{\mathsf{U}} \qquad \overline{\mathsf{T} \parr \mathsf{U}} = \overline{\mathsf{T}} \otimes \overline{\mathsf{U}} \qquad \overline{\mu\mathbf{t}.\mathsf{T}} = \mu\mathbf{t}.\overline{\mathsf{T}} \qquad \overline{\mathbf{t}} = \mathbf{t} \qquad \overline{\mathit{OK}} = \mathit{OK}$$

In addition to the duality between the multiplicatives, described above, the de Morgan duality between $\vee$ and $\wedge$ is standard for session types. The co-type of a prefix action interchanges send and receive, and dualises the continuation. The unit OK is self-dual. Since we have only guarded recursion, we treat fixed points equi-recursively, hence the fixed point operator is self-dual. Intuitively, equi-recursive types are treated equivalently to their infinite unfoldings.

Note co-types and the use of two multiplicatives is optional in this work. Having co-types reduces the number of rules in the next section by avoiding two sided sequents.

3.1 Deriving subtype judgements using the rules of Session

The rules of Session are defined in Fig. 3, using, in proof theoretic terms, a circular (or cyclic) sequent calculus [9, 4] — which is, in type theoretical terms, a coinductive algorithmic subtype system [47]. We employ an explicit algorithmic presentation of such a circular system where we have an axiom [Leaf] which is enabled whenever there is a loop in the proof returning to a sequent visited earlier in the proof. This algorithmic approach to coinduction is standard in type theory [2], being sound and complete for infinite proofs such as these due to the restriction to guarded recursion.

We explain the notation $[\Theta]\,\Gamma \vdash$. The sequent Γ is a (comma separated) multiset of types, hence types in a sequent can commute (exchange) inside a sequent, but cannot be duplicated (contraction) or removed (weakening). A **set of sequents** Θ, where each sequent in the set is separated using $\rrbracket$, is employed to define an algorithmic coinductive system, by remembering sequents that may be revisited. We omit Θ if it is empty.

▶ Remark 3. Note that proof systems typically formalise *provability of formulae*, written $\vdash \mathsf{T}$. For a tight match with session type conventions (without breaking the logical convention that $\wedge$ is conjunctive), we instead formulate *provability of duals of formulae.* To emphasise that we formulate probability of duals we write sequents as $\mathsf{T} \vdash$, which is equivalent to $\vdash \overline{\mathsf{T}}$.

Subtypes. Using co-types (Def. 2) and the rules in Fig. 3, subtyping can be defined as follows. Note, a type is closed when no type variables appear free.

▶ **Definition 4** (subtyping). *We say a closed type* T *is a subtype of another closed type* U*, written* $\mathsf{T} \leq \mathsf{U}$*, whenever* $\mathsf{T}\,,\,\overline{\mathsf{U}} \vdash$ *holds in* Session*.*

Note that in linear logic a linear implication $\underline{\mathsf{T} \multimap} \mathsf{U}$ holds whenever $\overline{\mathsf{T} \otimes \overline{\mathsf{U}}}$ is provable. Translating to provability of duals, proving $\overline{\mathsf{T} \otimes \overline{\mathsf{U}}}$ is equivalent to establishing $\mathsf{T}\,,\,\overline{\mathsf{U}} \vdash$. Indeed subtyping as defined above is a conservative extension of linear implication in linear logic (with the mix rule). In what follows, we confirm that standard subtype judgements are covered by the above definition. In addition, some additional subtype judgements hold, which are particular to the multiparty setting.

We briefly highlight that most rules are standard rules from linear logic and coinductive proof systems. Examples appear in the next section. Rules are well-defined over closed types.

Rules from MALL. Most rules of Session are rules of Multiplicative Additive Linear Logic (MALL), dualised in order to formalise provability of duals. The rule [Times] breaks down types into their parallel components. The rule [Par] is required for subtyping in the presence of parallelism. The axiom [OK] indicates that a protocol with no more actions has

$$\frac{}{[\Theta]\ \mathtt{OK}\ ,\ \mathtt{OK}\ ,\ \ldots\ \mathtt{OK} \vdash}\,[\textsc{OK}] \qquad \frac{}{[\Theta\ \llbracket\ \Gamma]\ \Gamma \vdash}\,[\textsc{Leaf}] \qquad \frac{[\Theta\ \llbracket\ \mu\mathbf{t}.\mathsf{T}\ ,\ \Gamma]\ \mathsf{T}\left\{{}^{\mu\mathbf{t}.\mathsf{T}}/_{\mathbf{t}}\right\}\ ,\ \Gamma \vdash}{[\Theta]\ \mu\mathbf{t}.\mathsf{T}\ ,\ \Gamma \vdash}\,[\textsc{Fix-}\mu]$$

$$\frac{[\Theta]\ ?\lambda_j;\mathsf{T}_j\ ,\ \Gamma \vdash \quad \text{for } \mathbf{some}\ j \in I}{[\Theta]\ \bigwedge_{i\in I} ?\lambda_j;\mathsf{T}_i\ ,\ \Gamma \vdash}\,[\textsc{Meet}] \qquad \frac{[\Theta]\ !\lambda_j;\mathsf{T}_j\ ,\ \Gamma \vdash \quad \text{for } \mathbf{all}\ j \in I}{[\Theta]\ \bigvee_{i\in I} !\lambda_j;\mathsf{T}_i\ ,\ \Gamma \vdash}\,[\textsc{Join}]$$

$$\frac{[\Theta]\ \mathsf{T}\ ,\ \mathsf{U}\ ,\ \Gamma \vdash}{[\Theta]\ !\lambda;\mathsf{T}\ ,\ ?\lambda;\mathsf{U}\ ,\ \Gamma \vdash}\,[\textsc{Prefix}] \qquad \frac{[\Theta]\ \mathsf{T}\ ,\ \mathsf{U}\ ,\ \Gamma \vdash}{[\Theta]\ \mathsf{T} \otimes \mathsf{U}\ ,\ \Gamma \vdash}\,[\textsc{Times}] \qquad \frac{[\Theta]\ \mathsf{T}\ ,\ \Gamma_1 \vdash \quad [\Theta]\ \mathsf{U}\ ,\ \Gamma_2 \vdash}{[\Theta]\ \mathsf{T} \parr \mathsf{U}\ ,\ \Gamma_1\ ,\ \Gamma_2 \vdash}\,[\textsc{Par}]$$

Figure 3 A presentation of the algorithmic coinductive proof system Session. Note, to align with session type conventions, the system establishes provability of duals.

successfully terminated (this rule is valid for MALL with mix). Rules [Join] and [Meet] are (dualised) standard rules for the additives of linear logic.

Rules for equi-recursion. Fixed points can be unfolded using the rule [Fix-μ]. Axiom [Leaf] is applied when we reach a previously visited sequent, completing a loop.

Rule [Prefix]. The exception to the above established rules for equi-recursion and MALL is the [Prefix] rule. This is used to model an interaction between two processes where one sends and the other receives. The rule enforces a causal order on interactions.

3.2 On notable admissible rules and algorithmic subtyping

For a proof system, we say a rule is *admissible*, whenever anything provable in the system with the rule present is provable in the same system but with the rule removed. We highlight the following three notable rules that are admissible in Session.

$$\frac{[\Theta]\ \Gamma_1\ ,\ \mathsf{T} \vdash \quad [\Theta]\ \overline{\mathsf{T}}\ ,\ \Gamma_2 \vdash}{[\Theta]\ \Gamma_1\ ,\ \Gamma_2 \vdash}\,[\textsc{Cut}] \qquad \frac{I \subseteq J \quad [\Theta]\ \mathsf{T}_k\ ,\ \mathsf{U}_k\ ,\ \Gamma \vdash \quad \text{for } \mathbf{all}\ k \in I}{[\Theta]\ \bigvee_{i\in I} !\lambda_i;\mathsf{T}_i\ ,\ \bigwedge_{j\in J} ?\lambda_j;\mathsf{U}_j\ ,\ \Gamma \vdash}\,[\textsc{Intr}] \qquad \frac{[\Theta]\ \Gamma_1 \vdash \quad [\Theta]\ \Gamma_2 \vdash}{[\Theta]\ \Gamma_1\ ,\ \Gamma_2 \vdash}\,[\textsc{Mix}]$$

Cut elimination and algorithmic subtyping. The admissibility of [Cut], called cut elimination, is the corner stone of proof theory, since many results in logic (e.g., the consistency of classical logic) can be proven as corollaries of cut elimination. Since cut elimination justifies that rules are consistently defined, we present cut elimination in Session as a theorem.

▶ **Theorem 5** (cut elimination). *The [Cut] rule is admissible in Session.*

To see that the above holds, observe that, trivially, the unfolding of a proof in Session to infinite proofs (over infinitely unfolded terms) is sound, and, due to regularity, complete (i.e., an infinite proof will always eventually loop on every branch, allowing [Leaf] to be applied). Thus it is sufficient to show that cut elimination holds for the finite proof system. This follows by observing that the standard normalisation steps for MALL, plus cases for

[Prefix], can be applied to unfold a cut free proof to an arbitrary depth. We show only the principal case for [Prefix], which is given by the following proof normalisation step.

$$\dfrac{\dfrac{\Gamma_1\,,\,\mathsf{U}\,,\,\mathsf{T}\vdash}{\Gamma_1\,,\,?\lambda;\mathsf{U}\,,\,!\lambda;\mathsf{T}\vdash} \qquad \dfrac{\overline{\mathsf{T}}\,,\,\mathsf{V}\,,\,\Gamma_2\vdash}{?\lambda;\overline{\mathsf{T}}\,,\,!\lambda;\mathsf{V}\,,\,\Gamma_2\vdash}}{\Gamma_1\,,\,?\lambda;\mathsf{U}\,,\,!\lambda;\mathsf{V}\,,\,\Gamma_2\vdash}\,[\textsc{Cut}] \quad\rightsquigarrow\quad \dfrac{\dfrac{\Gamma_1\,,\,\mathsf{U}\,,\,\mathsf{T}\vdash \qquad \overline{\mathsf{T}}\,,\,\mathsf{V}\,,\,\Gamma_2\vdash}{\Gamma_1\,,\,\mathsf{U}\,,\,\mathsf{V}\,,\,\Gamma_2\vdash}\,[\textsc{Cut}]}{\Gamma_1\,,\,?\lambda;\mathsf{U}\,,\,!\lambda;\mathsf{V}\,,\,\Gamma_2\vdash}$$

An immediate consequence of cut elimination for session types is that subtyping relation $\leq$ is transitive. It is also reflexive by a simple induction on the structure of types.

▶ **Corollary 6.** *If* $\mathsf{T}\leq\mathsf{U}$ *and* $\mathsf{U}\leq\mathsf{V}$*, then* $\mathsf{T}\leq\mathsf{V}$*. Also, we have* $\mathsf{T}\leq\mathsf{T}$*.*

From the perspective of type theory this is a standard result that **must** hold in order to recommend an *algorithmic subtype system*. An algorithmic subtype system is expressed without a cut (or transitivity) rule, since cut violates what is known as the *sub-formula property*. The sub-formula property states that every formula appearing in the premise is a sub-formula of one of formulae appearing in the conclusion (up to unfolding of equi-recursion, which is allowed here due to regularity). The sub-formula property guarantees that proof search in Session terminates.

Admissibility of [Intr]. Established algorithmic subtype systems usually employ a rule of the form [Intr]. That rule can be simulated by using [Join], [Meet] and [Prefix], without loss of expressive power. For example, the following sequent, provable using the rule [Intr] is also provable as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{ok}\,,\,\texttt{ok}\vdash}\,[\mathrm{OK}]}{?\lambda_1\,,\,!\lambda_1\vdash}\,[\textsc{Prefix}]}{(?\lambda_1\wedge ?\lambda_2)\,,\,!\lambda_1\vdash}\,[\textsc{Meet}] \qquad \dfrac{\dfrac{\dfrac{}{\texttt{ok}\,,\,\texttt{ok}\vdash}\,[\mathrm{OK}]}{?\lambda_2\,,\,!\lambda_2\vdash}\,[\textsc{Prefix}]}{(?\lambda_1\wedge ?\lambda_2)\,,\,!\lambda_2\vdash}\,[\textsc{Meet}]}{(?\lambda_1\wedge ?\lambda_2)\,,\,(!\lambda_1\vee !\lambda_2)\vdash}\,[\textsc{Join}]$$

However, we cannot simulate all proofs involving the three rules discussed above, if, instead, only [Intr] is employed. The following cannot be proven using only [Intr].

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{ok}\,,\,\texttt{ok}\,,\,\texttt{ok}\vdash}\,[\mathrm{OK}]}{!\lambda_3\,,\,\texttt{ok}\,,\,?\lambda_3\vdash}\,[\textsc{Prefix}]}{!\lambda_3\,,\,\texttt{ok}\,,\,?\lambda_2\wedge ?\lambda_3\vdash}\,[\textsc{Meet}]}{!\lambda_1;!\lambda_3\,,\,?\lambda_1\,,\,?\lambda_2\wedge ?\lambda_3\vdash}\,[\textsc{Prefix}]}{!\lambda_1;!\lambda_3\,,\,?\lambda_1\wedge ?\lambda_4\,,\,?\lambda_2\wedge ?\lambda_3\vdash}\,[\textsc{Meet}] \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{ok}\,,\,\texttt{ok}\,,\,\texttt{ok}\vdash}\,[\mathrm{OK}]}{!\lambda_4\,,\,?\lambda_4\,,\,\texttt{ok}\vdash}\,[\textsc{Prefix}]}{!\lambda_4\,,\,?\lambda_1\wedge ?\lambda_4\,,\,\texttt{ok}\vdash}\,[\textsc{Meet}]}{!\lambda_2;!\lambda_4\,,\,?\lambda_1\wedge ?\lambda_4\,,\,?\lambda_2\vdash}\,[\textsc{Prefix}]}{!\lambda_2;!\lambda_4\,,\,?\lambda_1\wedge ?\lambda_4\,,\,?\lambda_2\wedge ?\lambda_3\vdash}\,[\textsc{Meet}]}{!\lambda_1;!\lambda_3\vee !\lambda_2;!\lambda_4\,,\,?\lambda_1\wedge ?\lambda_4\,,\,?\lambda_2\wedge ?\lambda_3\vdash}\,[\textsc{Join}]$$

The following is an example of a coinductive proof that, similarly to the above proof, cannot be established using only [Intr]. In the following proof, assume $\mathsf{T}=\mu\mathbf{t}.(!\lambda_1;\mathbf{t}\vee !\lambda_2;\mathbf{t})$, $\mathsf{U}=\mu\mathbf{u}.(?\lambda_1;\mathbf{u})$, and $\mathsf{V}=\mu\mathbf{v}.(?\lambda_2;\mathbf{v})$. We also abbreviate sequents $\Gamma=\mathsf{T}\,,\,\mathsf{U}\,,\,\mathsf{V}$ and $\Gamma'=!\lambda_1;\mathsf{T}\,,\,\mathsf{U}\,,\,\mathsf{V}$ and $\Gamma''=!\lambda_2;\mathsf{T}\,,\,\mathsf{U}\,,\,\mathsf{V}$, but notice only Γ is used rule [Leaf].

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{[\Gamma'\,]\!\![\,\Gamma]\,\Gamma\vdash}\,[\textsc{Leaf}]}{[\Gamma'\,]\!\![\,\Gamma]\,!\lambda_1;\mathsf{T}\,,\,?\lambda_1;\mathsf{U}\,,\,\mathsf{V}\vdash}\,[\textsc{Prefix}]}{[\Gamma]\,!\lambda_1;\mathsf{T}\,,\,\mathsf{U}\,,\,\mathsf{V}\vdash}\,[\textsc{Fix}\text{-}\mu] \qquad \dfrac{\dfrac{\dfrac{}{[\Gamma''\,]\!\![\,\Gamma]\,\Gamma\vdash}\,[\textsc{Leaf}]}{[\Gamma''\,]\!\![\,\Gamma]\,!\lambda_2;\mathsf{T}\,,\,\mathsf{U}\,,\,?\lambda_2;\mathsf{V}\vdash}\,[\textsc{Prefix}]}{[\Gamma]\,!\lambda_2;\mathsf{T}\,,\,\mathsf{U}\,,\,\mathsf{V}\vdash}\,[\textsc{Fix}\text{-}\mu]}{[\Gamma]\,!\lambda_1;\mathsf{T}\vee !\lambda_2;\mathsf{T}\,,\,\mathsf{U}\,,\,\mathsf{V}\vdash}\,[\textsc{Join}]}{\mathsf{T}\,,\,\mathsf{U}\,,\,\mathsf{V}\vdash}\,[\textsc{Fix}\text{-}\mu]}{\mathsf{T}\,,\,\mathsf{U}\otimes\mathsf{V}\vdash}\,[\textsc{Times}]$$

Notice, the above proof establishes $\mu\mathbf{u}.(?\lambda_1;\mathbf{u}) \otimes \mu\mathbf{v}.(?\lambda_2;\mathbf{v}) \leq \mu\mathbf{t}.(?\lambda_1;\mathbf{t} \wedge ?\lambda_2;\mathbf{t})$ — a subtype judgement decomposing a single threaded participant into two concurrent threads.

Admissibility of [Mix]. The fact that the [Mix] rule is admissible allows scenarios where separate parallel communications can occur. For example, the subtype judgement $!\lambda_1 \otimes ?\lambda_1 \otimes !\lambda_2 \otimes ?\lambda_2 \leq \texttt{OK}$ (which also holds in pure linear logic **with** mix only), can be established by the following proof in Session **without** using mix.

$$\dfrac{\dfrac{\dfrac{}{\texttt{OK}\ ,\ \texttt{OK}\ ,\ \texttt{OK}\ ,\ \texttt{OK}\ ,\ \texttt{OK} \vdash}\ [\text{OK}]}{!\lambda_1\ ,\ ?\lambda_1\ ,\ !\lambda_2\ ,\ ?\lambda_2\ ,\ \texttt{OK} \vdash}\ [\textsc{Prefix}]\ (\textit{twice})}{!\lambda_1 \otimes ?\lambda_1 \otimes !\lambda_2 \otimes ?\lambda_2\ ,\ \texttt{OK} \vdash}\ [\textsc{Times}]\ (\textit{twice})$$

The admissibility of [Mix] is a corollary of Theorem 5.

3.3 Typing multiparty compatible networks, by using subtyping

The syntax of processes is defined by the following grammar.

▶ **Definition 7** (Processes). Processes for threads *are defined by:*

$$\mathbb{P} ::= \quad \Sigma_{i\in I}?\lambda_i;\mathbb{P}_i \quad | \quad \oplus_{i\in I}!\lambda_i;\mathbb{P}_i \quad | \quad \mu X.\mathbb{P} \quad | \quad X \quad | \quad 1$$

Processes for networks *are defined by grammar:* $\mathbb{N} ::= \ \mathbb{P} \ | \ \mathbb{N} \parallel \mathbb{N}$.

We simply refer to both of the above as processes*, ranged over by P, Q, R. . . .*

Internal choice $\oplus$ defines a process ready to perform **any** of the given outputs, and external choice $\sum$ indicates a process ready to perform **some** input. We typically abbreviate $!\lambda;P$ and $!\lambda_1;P_1 \oplus !\lambda_2;P_2$ for the unary and binary versions of the above external choice. Similarly, $?\lambda;P$ and $?\lambda_1;P_1 + ?\lambda_2;P_2$ can be used for internal choices.

$$\dfrac{\Delta \vdash P_i : \mathsf{T}_i \ (i \in I)}{\Delta \vdash \Sigma_{i\in I}?\lambda_i;P_i : \bigwedge_{i\in I}?\lambda_i;\mathsf{T}_i}\ [\text{t-}\textsc{ExtCh}] \qquad \dfrac{\Delta \vdash P_i : \mathsf{T}_i \ (i \in I)}{\Delta \vdash \oplus_{i\in I}!\lambda_i;P_i : \bigvee_{i\in I}!\lambda_i;\mathsf{T}_i}\ [\text{t-}\textsc{IntCh}]$$

$$\Delta, X : \mathbf{t} \vdash X : \mathbf{t} \quad [\text{t-}\textsc{Var}] \qquad \dfrac{\Delta, X : \mathbf{t} \vdash P : \mathsf{T}}{\Delta \vdash \mu X.P : \mu\mathbf{t}.\mathsf{T}}\ [\text{t-}\textsc{Rec}]$$

$$\dfrac{\Delta \vdash P : \mathsf{T} \quad \Delta \vdash Q : \mathsf{U}}{\Delta \vdash P \parallel Q : \mathsf{T} \otimes \mathsf{U}}\ [\text{t-}\textsc{Par}] \qquad \Delta \vdash 1 : \texttt{OK} \quad [\text{t-}1] \qquad \dfrac{\Delta \vdash P : \mathsf{T} \qquad \mathsf{T} \leq \mathsf{U}}{\Delta \vdash P : \mathsf{U}}\ [\textsc{subsumption}]$$

Figure 4 Typing rules for processes, making use of the subtype relation $\leq$ in Def. 4.

Multiparty compatible processes are those with type $\texttt{OK}$. Note, for any interesting example, this will involve applying SUBSUMPTION.

▶ **Definition 8** (compatibility). *Process P is multiparty compatible whenever $\vdash P : \texttt{OK}$, according to the rules of Fig. 4, where environment Δ associates process variables to type variables.*

Any application of the [SUBSUMPTION] rule can always be delayed to the final step. I.e., we calculate the minimal type for the whole network, then apply [SUBSUMPTION].

▶ **Theorem 9** (algorithmic typing). *If $\vdash P : \mathsf{U}$ then we can construct a T such that $\mathsf{T} \leq \mathsf{U}$ holds and $\vdash P : \mathsf{T}$ holds without using the* [SUBSUMPTION] *rule.*

The above result is another consequence of cut elimination.

An immediate consequence is that, if P is multiparty compatible, there exists T such that $\vdash P\colon \mathsf{T}$, without using the *subsumption* rule, and $\mathsf{T} \vdash$ holds. For example, proofs from the previous section can be used to established that networks such as the following are multiparty compatible: $!\lambda_1;!\lambda_3 \oplus !\lambda_2;!\lambda_4 \parallel ?\lambda_1 + ?\lambda_4 \parallel ?\lambda_2 + ?\lambda_3$ and $\mu\mathbf{t}.(!\lambda_1;\mathbf{t} \oplus !\lambda_2;\mathbf{t}) \parallel \mu\mathbf{u}.(?\lambda_1;\mathbf{u}) \parallel \mu\mathbf{v}.(?\lambda_2;\mathbf{v})$. Furthermore, the multiparty compatibility of the processes from Sec. 2 can be established in this way.

Note on open sessions. We select a flexible presentation in Fig. 4, since, as a bonus, we can also use the above type system to reason about open sessions, which may be missing participants in order for multiparty compatibility to hold. For example, by using [SUBSUMPTION] and the processes from Sec. 2, we have the following type judgement.

$$\vdash \mathit{Owner} \parallel \mathit{Untrusted\ App} \parallel \mathit{OAuth\ Server}\colon \mu\mathbf{t}.\big(!\mathit{release} \oplus \ !\mathit{request}(\mathit{token}); \\ (?\mathit{revoke} + ?\mathit{response}(\mathit{data});\mathbf{t})\ \big)$$

The above type judgement indicates an "interface" exposed by the open session given by network *Owner* ∥ *Untrusted App* ∥ *OAuth Server*. Hence, if composed with a process that interacts with the interface given by the dual of the above type (such as *Resource* from Sec 2) we can judge the whole system to be multiparty compatible. Composition of two open sessions can be performed by using [T-PAR] and then applying [SUBSUMPTION] to the resulting type to show that, together, they inhabit type $\mathtt{OK}$, assuming that together the processes are multiparty compatible (alternatively, when composed, they may expose another interface if the composition of two open sessions is still an open session). Note this achieves the same effect as applying a rule of the following form.

$$\frac{\Delta \vdash P : \mathsf{T} \otimes \mathsf{U} \quad \Delta \vdash Q : \overline{\mathsf{U}} \otimes \mathsf{V}}{\Delta \vdash P \parallel Q : \mathsf{T} \otimes \mathsf{V}}\ [\textsc{t-Cut}]$$

The above rule, derivable using [T-PAR] and [SUBSUMPTION], achieves the effect of a "connecting cut", as desired in recent work on open multiparty sessions [6].

3.4 Guaranteeing deadlock freedom (via race freedom)

In order to prove deadlock freedom of multiparty compatible networks, we require a reduction system for closed networks, defined by the rules in Fig. 5. As standard [17], different behaviours are forced for internal choice and external choice. When ranging over all executions, for external choice, we consider all branches, as indicated by the transition rule for *internal choice* ($\oplus$). Notice that, in order for a communication to occur, we must have committed to a single branch of the internal choice, forcing all branches to be resolved. However, we need only select one of the inputs with the corresponding output label in an external choice ($\sum$) for a communication to occur.

$$\frac{j \in I}{\oplus_{i\in I}!\lambda_i;Q_i \longrightarrow !\lambda_j;Q_j} \qquad \frac{j \in I}{!\lambda_j;P \parallel \Sigma_{i\in I}?\lambda_i;Q_i \longrightarrow P \parallel Q_j} \qquad \frac{}{\mathtt{rec}X.P \longrightarrow P\{^{\mathtt{rec}X.P}/_X\}}$$

$$\frac{P \longrightarrow P'}{P \parallel Q \longrightarrow P' \parallel Q} \qquad \frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q} \qquad \begin{array}{l} P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R \\ P \parallel Q \equiv Q \parallel P \quad P \parallel 1 \equiv P \end{array}$$

Figure 5 Reduction system for networks.

Race freedom. Some multiparty compatible networks with race conditions are not deadlock free. Races can be avoided by naming participants and ensuring each branch of an external choice awaits a message from the same participant but is labelled differently compared to other branches of that external choice. For example, the following multiparty compatible networks have races, hence should be rejected. For network $!\lambda_1;!\lambda_2 \oplus !\lambda_1;!\lambda_3 \parallel ?\lambda_1;?\lambda_2 + ?\lambda_1;?\lambda_3$, when λ_1 is sent it may be received by the wrong branch of the external choice resulting in deadlock. Similarly, network $!\lambda_1;?\lambda_2 \parallel !\lambda_1 \parallel ?\lambda_1;!\lambda_2;?\lambda_1$, may deadlock if the second process engages in a communication before the first.

While explicitly naming participants, as described above, would avoid such examples, for added flexibility we show that we can also achieve race freedom without naming participants. This additional flexibility is necessary for examples such as in Sec. 2, where one participant is replaced by two or more participants (hence if participants were named we would require a mechanism such as internal delegation [13] to allow one participant act on behalf of another). An added benefit of avoiding races without naming participants is that we may guarantee race freedom without relying on participant names to guide reductions.

Race freedom can be formulated in terms of a type inference problem using the race type system in Fig. 6, where A *race type* is of the form $\langle \texttt{o}{:}\alpha, \texttt{i}{:}\chi \rangle$, where α and χ are sets of sets of labels. The former, α, represents a set of sets of output labels — one set of labels for each thread in a network. The latter χ represents a set of sets of inputs — one set of labels for each external choice somewhere in the network. We also require a "participant condition" ensuring all branches of a choice talk to the same process, formalised as follows.

▶ **Definition 10.** *A race type $\langle \texttt{o}{:}\alpha, \texttt{i}{:}\chi \rangle$ satisfies the participant condition whenever, for all $x \in \chi$ and $y, z \in \alpha$, if $x \cap y \neq \emptyset$ and $x \cap z \neq \emptyset$ then $y = z$. A process P is race free, whenever there exists a race type $\langle \texttt{o}{:}\alpha, \texttt{i}{:}\chi \rangle$ satisfying the participant condition such that $\vdash P \colon \langle \texttt{o}{:}\alpha, \texttt{i}{:}\chi \rangle$ using the rules of Fig. 6.*

$$\frac{\Sigma \vdash P_i : \langle \texttt{o}{:}\{x_i\}, \texttt{i}{:}\chi_i \rangle \;\; (i \in I) \quad (\forall i, j \in I)\lambda_i = \lambda_j \text{ implies } i = j}{\Sigma \vdash \Sigma_{i \in I} ?\lambda_i;P_i : \left\langle \texttt{o}{:}\left\{ \bigcup_{i \in I} x_i \right\}, \texttt{i}{:}\bigcup_{i \in I} \chi_i \cup \{\{\lambda_i \colon i \in I\}\} \right\rangle} \;[\textsc{r-ExtCh}]$$

$$\frac{\Sigma \vdash P_i : \langle \texttt{o}{:}\{x_i\}, \texttt{i}{:}\chi_i \rangle \;\; (i \in I)}{\Sigma \vdash \oplus_{i \in I} !\lambda_i;P_i : \left\langle \texttt{o}{:}\left\{ \bigcup_{i \in I} x_i \cup \{\lambda_i \colon i \in I\} \right\}, \texttt{i}{:}\bigcup_{i \in I} \chi_i \right\rangle} \;[\textsc{r-IntCh}]$$

$$\frac{\Sigma \vdash P : \langle \texttt{o}{:}\alpha, \texttt{i}{:}\chi \rangle \quad \Sigma \vdash Q : \langle \texttt{o}{:}\beta, \texttt{i}{:}\zeta \rangle \quad \left(\bigcup \alpha\right) \cap \left(\bigcup \beta\right) = \emptyset \quad \left(\bigcup \chi\right) \cap \left(\bigcup \zeta\right) = \emptyset}{\Sigma \vdash P \parallel Q : \langle \texttt{o}{:}\alpha \cup \beta, \texttt{i}{:}\chi \cup \zeta \rangle} \;[\textsc{r-Par}]$$

$$\Sigma, X : \langle \texttt{o}{:}\alpha, \texttt{i}{:}\chi \rangle \vdash X : \langle \texttt{o}{:}\alpha, \texttt{i}{:}\chi \rangle \;\; [\textsc{r-Var}] \qquad \Sigma \vdash 1 : \langle \texttt{o}{:}\emptyset, \texttt{i}{:}\emptyset \rangle \;\; [\textsc{r-1}]$$

$$\frac{\Sigma, X : \langle \texttt{o}{:}\alpha, \texttt{i}{:}\chi \rangle \vdash P : \langle \texttt{o}{:}\alpha, \texttt{i}{:}\chi \rangle}{\Sigma \vdash \mu X.P : \langle \texttt{o}{:}\alpha, \texttt{i}{:}\chi \rangle} \;[\textsc{r-Rec}]$$

Figure 6 Type rules for checking race freedom.

The above *race-freedom* property we propose is satisfied whenever the unfolding of all fixed points of a process satisfies the following:

- Branches of an external choice use distinct labels for their **immediately enabled** inputs (see [R-EXTCH]).
- For any external choice, the set of immediately enabled input labels in an external choice must be disjoint from the set of all output labels of **all but one** of the parallel components (the *participant condition*). This ensures one participant is listening to at most one other participant at a time.
- For parallel processes, $P \parallel Q$, the set of **all** input labels of P and the set of **all** input labels of Q are disjoint; and, similarly, the sets of all output labels of P and Q are disjoint (see [R-PAR]).

The above property is efficient to check, since it simply builds up the relevant sets of sets of labels. Note, a single thread always has a singleton set of outputs.

A critical example the participant condition rejects is the following process, which is of the race type indicated below.

$$\vdash\, !\lambda_1 \parallel \mathtt{rec}X.\,(?\lambda_1 + ?\lambda_2;X) \parallel \mathtt{rec}Y.!\lambda_2;Y:\ \langle \mathtt{o}:\{\{\lambda_1\},\emptyset,\{\lambda_2\}\},\mathtt{i}:\{\{\lambda_1,\lambda_2\}\}\rangle$$

The above example processes contains a race. Two parallel outputs with different labels contact a process ready to receive a message from either process and, if actions labelled λ_1 are played, the process deadlocks. The above example is forbidden by the participant condition since we have $\{\lambda_1,\lambda_2\}\cap\{\lambda_1\}\neq\emptyset$ and $\{\lambda_1,\lambda_2\}\cap\{\lambda_2\}\neq\emptyset$ but $\{\lambda_1\}\neq\{\lambda_2\}$.

Deadlock freedom. Deadlock freedom can be defined as follows (coinductively): at any point we can either make progress or we have successfully terminated.

▶ **Definition 11** (deadlock freedom). *A network P is deadlock free whenever:*

- *either $P \equiv 1$, or there exists network Q such that $P \longrightarrow Q$;*
- *and, for all R such that $P \longrightarrow R$ we have R is deadlock free.*

The theory developed in this work guarantees deadlock freedom as in Def. 11.

▶ **Theorem 12.** *Any race-free multiparty-compatible network satisfies deadlock freedom.*

The proof of this result [see Appendix] relies on Theorem 5 and builds on novel proof normalisation techniques developed for giving computational interpretations of formulae in extensions of linear logic [35, 36].

▶ Remark 13. Note often deadlock freedom is referred to as "progress" which is an overloaded word in the literature. Deadlock freedom does not necessarily prevent starvation, as for notions such as lock freedom [37, 46]. Restricted variants of Session can be tightened to enforce stronger liveness properties — an observation deserving of attention in future work.

Soundness of the subtype system with respect to our multithreaded liberalisation of the substitution principle [25] is precisely formulated below, which is an immediate consequence of Theorem 5 and Theorem 12. Notice the flexible subtype system in this work, which permits networks consisting of parallel threads to be compared, allows a thread to be substituted by more than one thread, as motivated in Sec. 2.

▶ **Corollary 14** (substitution principle). *Assume P, Q and R are closed networks. If $\vdash P\colon \mathsf{T}$, $\vdash Q\colon \mathsf{T}'$ and $\mathsf{T} \leq \mathsf{T}'$, then if $\vdash Q \parallel R\colon$ OK, and $P \parallel R$ is race free, then $P \parallel R$ is deadlock free.*

Proof. Assume $\vdash P \colon \mathsf{T}$ and $\vdash Q \colon \mathsf{T}'$ without using [SUBSUMPTION], and also assume $\mathsf{T} \leq \mathsf{T}'$, $\vdash Q \parallel R \colon \texttt{OK}$ and $P \parallel R$ is race free. By Theorem 9, there exists a type U such that $\vdash Q \parallel R \colon \mathsf{U}$ without using [SUBSUMPTION] and $\mathsf{U} \leq \texttt{OK}$. By Lemma 15 there exists V such that $\mathsf{U} = \mathsf{T}' \otimes \mathsf{V}$ and $\vdash P \parallel R \colon \mathsf{T}' \otimes \mathsf{V}$ without using [SUBSUMPTION]. By Theorem 5, we have $\mathsf{T} \otimes \mathsf{V} \leq \mathsf{T}' \otimes \mathsf{V}$, and hence, by Theorem 5 again, $\mathsf{T} \otimes \mathsf{V} \leq \texttt{OK}$. Thereby $\vdash P \parallel R \colon \texttt{OK}$, and hence, by race freedom and Theorem 12, we have $P \parallel R$ is deadlock free, as required. ◀

Importance of avoiding races. The following example emphasises the importance of checking races are avoided. Consider the multiparty compatible network $1 \parallel !\lambda_1 \parallel (?\lambda_1 + ?\lambda_2)$. Observe we have $\vdash\ !\lambda_2 \parallel ?\lambda_2 \colon \texttt{OK}$ hence process 1 can be substituted by $!\lambda_2 \parallel ?\lambda_2$ while preserving multiparty compatibility. Now, if we remove the condition concerning races in the substitution principle, after applying the above substitution in the network at the top of the paragraph, we should have $!\lambda_2 \parallel ?\lambda_2 \parallel !\lambda_1 \parallel (?\lambda_1 + ?\lambda_2)$ is deadlock free. However, this network is in fact not deadlock free, due to the presence of a race.

Note our race-freedom property does not require output labels in an internal choice to be distinct. For example, the network $(!\lambda_1;!\lambda_2 \oplus !\lambda_1;!\lambda_3) \parallel ?\lambda_1;(?\lambda_2 + ?\lambda_3)$ is race free, multiparty compatible and deadlock free. Note this is example would not be typeable using established session type systems.

3.5 Typeable sessions for which there is no global type

Multiparty compatibility is defined independently from global types. Theories that rely on global types run into the problem that many reasonable protocols have no global type. Such problematic protocols typically feature branching under a recursion where different participants are contacted in each branch. The problem of typing protocols for which there is no established theory in which they can be assigned a global type has been explored in recent work [48].

To emphasise that Session can also be used to type multiparty sessions for which there is no global type, we adapt one of the key examples from related work (Figure 4, (2) [48]). In this recursive two-buyer protocol a buyer repeatedly asks another buyer to split the price. Assume we have the following types.

- $\mathsf{T}_{\mathtt{A}} = !\mathtt{query};?\mathtt{price};\mu\mathbf{t}.\mathsf{T}_1$ where $\mathsf{T}_1 = (!\mathtt{split};\mathsf{T}_2 \vee !\mathtt{cancel};!\mathtt{no})$ and $\mathsf{T}_2 = (?\mathtt{yes};!\mathtt{buy} \wedge ?\mathtt{no};\mathbf{t})$
- $\mathsf{T}_{\mathtt{B}} = \mu\mathbf{t}.\mathsf{T}_3$ where $\mathsf{T}_3 = (?\mathtt{split};\mathsf{T}_4 \wedge ?\mathtt{cancel})$ and $\mathsf{T}_4 = !\mathtt{yes} \vee !\mathtt{no};\mathbf{t}$
- $\mathsf{T}_{\mathtt{S}} = ?\mathtt{query};!\mathtt{price};\mathsf{T}_5$ where $\mathsf{T}_5 = ?\mathtt{buy} \wedge ?\mathtt{no}$.

Also assume we have sequents $\Gamma = \mu\mathbf{t}.\mathsf{T}_1\ ,\ \mathsf{T}_5\ ,\ \mathsf{T}_{\mathtt{B}}$ and $\Gamma' = \mathsf{T}_1\left\{{}^{\mu\mathbf{t}.\mathsf{T}_1}/_{\mathbf{t}}\right\}\ ,\ \mathsf{T}_5\ ,\ \mathsf{T}_{\mathtt{B}}$ (only the former is used in a [LEAF] axiom). The following proof can be used to establish $\mathsf{T}_{\mathtt{A}} \otimes \mathsf{T}_{\mathtt{B}} \otimes \mathsf{T}_{\mathtt{S}} \leq \texttt{OK}$, which can be used in a multiparty compatibility judgement. Notice we use the admissible compound rule [INTR] to shorten the proof.

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{[OK]}
\UnaryInfC{$\left[\Gamma' \rrbracket \Gamma\right] \texttt{OK}\ ,\ \texttt{OK}\ ,\ \texttt{OK} \vdash$}
\RightLabel{[INTR]}
\UnaryInfC{$\left[\Gamma' \rrbracket \Gamma\right] !\mathtt{buy}\ ,\ \mathsf{T}_5\ ,\ \texttt{OK} \vdash$}
\AxiomC{}
\RightLabel{[LEAF]}
\UnaryInfC{$\left[\Gamma' \rrbracket \Gamma\right] \mu\mathbf{t}.\mathsf{T}_1\ ,\ \mathsf{T}_5\ ,\ \mathsf{T}_{\mathtt{B}} \vdash$}
\RightLabel{[INTR]}
\BinaryInfC{$\left[\Gamma' \rrbracket \Gamma\right] \mathsf{T}_2\left\{{}^{\mu\mathbf{t}.\mathsf{T}_1}/_{\mathbf{t}}\right\}\ ,\ \mathsf{T}_5\ ,\ \mathsf{T}_4\left\{{}^{\mu\mathbf{t}.\mathsf{T}_3}/_{\mathbf{t}}\right\} \vdash$}
\AxiomC{}
\RightLabel{[OK]}
\UnaryInfC{$\left[\Gamma' \rrbracket \Gamma\right] \texttt{OK}\ ,\ \texttt{OK}\ ,\ \texttt{OK} \vdash$}
\RightLabel{[INTR]}
\UnaryInfC{$\left[\Gamma' \rrbracket \Gamma\right] !\mathtt{no}\ ,\ \mathsf{T}_5\ ,\ \texttt{OK} \vdash$}
\RightLabel{[INTR]}
\BinaryInfC{$\left[\Gamma' \rrbracket \Gamma\right] \mathsf{T}_1\left\{{}^{\mu\mathbf{t}.\mathsf{T}_1}/_{\mathbf{t}}\right\}\ ,\ \mathsf{T}_5\ ,\ \mathsf{T}_3\left\{{}^{\mu\mathbf{t}.\mathsf{T}_3}/_{\mathbf{t}}\right\} \vdash$}
\RightLabel{[FIX-μ]}
\UnaryInfC{$\left[\Gamma\right] \mathsf{T}_1\left\{{}^{\mu\mathbf{t}.\mathsf{T}_1}/_{\mathbf{t}}\right\}\ ,\ \mathsf{T}_5\ ,\ \mathsf{T}_{\mathtt{B}} \vdash$}
\RightLabel{[FIX-μ]}
\UnaryInfC{$\mu\mathbf{t}.\mathsf{T}_1\ ,\ \mathsf{T}_5\ ,\ \mathsf{T}_{\mathtt{B}} \vdash$}
\RightLabel{[PREFIX]}
\UnaryInfC{$?\mathtt{price};\mu\mathbf{t}.\mathsf{T}_1\ ,\ !\mathtt{price};\mathsf{T}_5\ ,\ \mathsf{T}_{\mathtt{B}} \vdash$}
\RightLabel{[PREFIX]}
\UnaryInfC{$\mathsf{T}_{\mathtt{A}}\ ,\ \mathsf{T}_{\mathtt{S}}\ ,\ \mathsf{T}_{\mathtt{B}} \vdash$}
\end{prooftree}
$$

In the above example, it is possible that processes typed with $\mathsf{T}_{\mathtt{A}}$ and $\mathsf{T}_{\mathtt{B}}$ negotiate forever and a process typed with $\mathsf{T}_{\mathtt{S}}$, after reaching a state typed by T_5, waits forever. Such starvation is permitted by our classic notion of progress in Def. 11, i.e., deadlock freedom.

4 Related Work and Future Work

A closely related line of work studies the problem of synthesising a "coherent" global type for multi-party compatible types [43]. The approach in the current paper can be used to expose the structural proof theoretic content of a closely related system proposed for such a synthesis problem [38]. There is much work providing notions of semantic subtyping for session types [7, 5, 45], whose resulting systems can be interpreted proof theoretically using subsystems and variants of Session (at least for the first-order fragment without delegation).

It could be valuable to explore connections between Session, which follows a processes-as-formulas approach, and a variety of Curry-Howard inspired systems. There are intersection type systems, satisfying subject expansion, that completely characterise deadlock freedom for a fragment of the asynchronous π-calculus where a name can only be used as an input channel by the process that created the name [16]. Process in that work are quite different from those in our session calculus, since, in this work, we neither consider channel passing (delegation) nor asynchrony, while they do not consider choice. Challenges concerning duality of binary sessions in the presence of delegation and recursion are explored through a linear λ-calculus typed using explicit least and greatest fixedpoints rather than equi-recursion [40]. Regarding circular proofs, Derakhshan and Pfenning propose a calculus for binary sessions with delegation in a Curry-Howard style [22]. In their work, they propose a locally checkable condition that guarantees a well-typed session will always terminate either in an empty configuration or a configuration attempting to communicate along external channels.

In future work, it would be valuable to investigate variants of the rules, notably a focussed variant of Session [3, 4]. In a focussed system, rules such as Join are treated *asynchronously*, meaning that we can immediately apply the rule without backtracking; whereas rules such as Meet are *synchronous*, meaning that, in general, backtracking may be required during proof search. The important observation is that, for race-free sessions there will only be one way to apply synchronous rules, thereby eliminating the need to backtrack in the search for a proof, i.e., proof search can be conducted deterministically. The ability to search for proofs in this uniform manner is connected with goal-directed search in logic programming [42].

The system designed in this work preserves deadlock freedom for race-free processes, as established in Theorem 12; but does not guarantee stronger livelock freedom properties (sometimes referred to as lock freedom) [37, 46, 49]. Livelock freedom strengthens deadlock freedom by ensuring that no parties are starved of resources; however, there are many subtle variations on precisely how livelock freedom is defined. Hence we push the investigation of refinements of Session that can guarantee notions livelock freedom to future work.

To illustrate the above point, we observe some more unexpected properties of Session. Observe, the process $\mu X.?\lambda_1;X \parallel ?\lambda_2 \parallel \mu Y.!\lambda_1;Y$ is race-free and multiparty compatible, and hence deadlock free. However, it has a hanging input $?\lambda_2$ that never receives a message, hence it is not livelock free in any sense. Using a proof of the multiparty compatibility of the above process, we can also establish subtype judgement $\mu\mathbf{t}.?\lambda_1;\mathbf{t} \otimes ?\lambda_2 \leq \mu\mathbf{t}.?\lambda_1;\mathbf{t}$. This subtype judgements allows inactive parallel components to be typed using the subtype system, as long as they rest of the system is deadlock free. Thus the current formulation of Session guarantees no property stronger than deadlock freedom.

For a more subtle example outside the scope of established session type systems, consider the types $\mathsf{T} = \mu\mathbf{t}.(!\lambda_1;\mathbf{t} \vee !\lambda_2;!\lambda_3)$ and $\mathsf{U} = \mu\mathbf{t}.(?\lambda_1;\mathbf{t} \wedge ?\lambda_2)$. We have $\mathsf{T} \otimes \mathsf{U} \leq !\lambda_3$ thus a thread that sends λ_2 can be replaced by two threads that may choose to talk internally on λ_1 forever, although there is always the possibility of a branching taken where λ_3 is sent. This subtype judgement does preserve some notions of livelock freedom (it is always possible for everyone to eventually act [46]), but not stronger notions of livelock freedom (always everyone must act eventually [37]). An objective for future work would be to explain how Session can be refined by restricting circular proofs so that they preserve a strong form of livelock freedom. The key idea is to check that at all threads in a network act at least once in every unfolding of a recursion, thereby rejecting both subtype judgements above.

5 Conclusion

The proof calculus Session, introduced in Fig. 3, showcases tools of structural proof theory, i.e., analytic calculi satisfying cut elimination (Theorem 5), which can be used in the design of rich multiparty session type systems. Session defines an algorithmic subtype system (Definition 4), the transitivity of which follows from cut elimination (Corollary 6). The subtype system admits a more flexible substitution principle (Corollary 14) than standard. This flexibility enables subtyping to be used directly to decide multiparty compatibility (Definition 8) and also opens up fresh problems that can be tackled using subtyping, not limited to scenarios where extra parallelism is introduced, as illustrated in Sec. 2.

Race freedom may be guaranteed by naming participants; however, for extra flexibility we propose a type system for race freedom (Definition 10). From these definitions, we establish our main result (Theorem 12) guaranteeing deadlock freedom for networks that are both multiparty compatible and race free. In this line of work, global types are optional, allowing networks for which no global type exists to be typed.

References

1. Samson Abramsky. Computational interpretations of linear logic. *Theoretical computer science*, 111(1):3–57, 1993. `doi:10.1016/0304-3975(93)90181-R`.
2. Roberto M. Amadio and Luca Cardelli. Subtyping recursive types. *ACM Trans. Program. Lang. Syst.*, 15(4):575–631, September 1993. `doi:10.1145/155183.155231`.
3. Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992. `doi:10.1093/logcom/2.3.297`.
4. David Baelde, Amina Doumane, and Alexis Saurin. Infinitary Proof Theory: the Multiplicative Additive Case. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *LIPIcs*, pages 42:1–42:17. Schloss Dagstuhl, 2016. `doi:10.4230/LIPIcs.CSL.2016.42`.
5. Franco Barbanera and Ugo de'Liguoro. Sub-behaviour relations for session-based client/server systems. *Mathematical Structures in Computer Science*, 25(6):1339–1381, 2015. `doi:10.1017/S096012951400005X`.
6. Franco Barbanera and Mariangiola Dezani-Ciancaglini. Open multiparty sessions. In Massimo Bartoletti, Ludovic Henrio, Anastasia Mavridou, and Alceste Scalas, editors, *Proceedings 12th Interaction and Concurrency Experience, ICE 2019, Copenhagen, Denmark, 20-21 June 2019.*, volume 304 of *EPTCS*, pages 77–96, 2019. `doi:10.4204/EPTCS.304.6`.
7. Giovanni Bernardi and Matthew Hennessy. Using higher-order contracts to model session types. *Logical Methods in Computer Science*, 12(2), 2016. `doi:10.2168/LMCS-12(2:10)2016`.
8. Eduardo Bonelli and Adriana Compagnoni. Multipoint session types for a distributed calculus. In Gilles Barthe and Cédric Fournet, editors, *Trustworthy Global Computing*, pages 240–256. Springer, 2008. `doi:10.1007/978-3-540-78663-4_17`.

9 James Brotherston and Alex Simpson. Sequent calculi for induction and infinite descent. *Journal of Logic and Computation*, 21(6):1177–1216, 10 2010. doi:10.1093/logcom/exq052.

10 Paola Bruscoli. A purely logical account of sequentiality in proof search. In Peter J. Stuckey, editor, *Logic Programming*, pages 302–316. Springer, 2002. doi:10.1007/3-540-45619-8_21.

11 Luís Caires and Jorge A. Pérez. Multiparty session types within a canonical binary theory, and beyond. In Elvira Albert and Ivan Lanese, editors, *FORTE 2016*, volume 9688 of *Lecture Notes in Computer Science*, pages 74–95. Springer, 2016. doi:10.1007/978-3-319-39570-8_6.

12 Marco Carbone, Fabrizio Montesi, Carsten Schürmann, and Nobuko Yoshida. Multiparty session types as coherence proofs. *Acta Informatica*, 54(3):243–269, 2017. doi:10.1007/s00236-016-0285-y.

13 Ilaria Castellani, Mariangiola Dezani-Ciancaglini, Paola Giannini, and Ross Horne. Global types with internal delegation. *Theoretical Computer Science*, 807:128–153, 2020. doi:10.1016/j.tcs.2019.09.027.

14 Tzu-chun Chen, Mariangiola Dezani-Ciancaglini, Alceste Scalas, and Nobuko Yoshida. On the Preciseness of Subtyping in Session Types. *Logical Methods in Computer Science*, Volume 13, Issue 2, 2017. doi:10.23638/LMCS-13(2:12)2017.

15 Gabriel Ciobanu and Ross Horne. Behavioural analysis of sessions using the calculus of structures. In Manuel Mazzara and Andrei Voronkov, editors, *Perspectives of System Informatics - 10th International Andrei Ershov Informatics Conference, PSI 2015*, volume 9609 of *Lecture Notes in Computer Science*, pages 91–106. Springer, 2015. doi:10.1007/978-3-319-41579-6_8.

16 Ugo Dal Lago, Marc de Visme, Damiano Mazza, and Akira Yoshimizu. Intersection types and runtime errors in the pi-calculus. *Proc. ACM Program. Lang.*, 3(POPL), 2019. doi:10.1145/3290320.

17 Rocco De Nicola and Matthew Hennessy. CCS without τ's. In Hartmut Ehrig, Robert Kowalski, Giorgio Levi, and Ugo Montanari, editors, *TAPSOFT '87*, pages 138–152. Springer, 1987. doi:10.1007/3-540-17660-8_53.

18 Romain Demangeon and Kohei Honda. Full abstraction in a subtyped pi-calculus with linear types. In *CONCUR*, volume 6901 of *LNCS*, pages 280–296. Springer, 2011. doi:10.1007/978-3-642-23217-6_19.

19 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In Helmut Seidl, editor, *Programming Languages and Systems*, pages 194–213. Springer, 2012. doi:10.1007/978-3-642-28869-2_10.

20 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *Automata, Languages, and Programming*, pages 174–186. Springer, 2013. doi:10.1007/978-3-642-39212-2_18.

21 Pierre-Malo Deniélou, Nobuko Yoshida, Andi Bejleri, and Raymond Hu. Parameterised multiparty session types. *Logical Methods in Computer Science*, 8(4), 2012. doi:10.2168/LMCS-8(4:6)2012.

22 Farzaneh Derakhshan and Frank Pfenning. Circular proof as session-typed processes: a local validity condition. (arXiv:1908.01909), 2019. URL: https://arxiv.org/abs/1908.01909.

23 Simon Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, 2005. doi:10.1007/s00236-005-0177-z.

24 Simon J. Gay and Malcolm Hole. Types and subtypes for client-server interactions. In *ESOP*, volume 1576 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 1999. doi:10.1007/3-540-49099-X_6.

25 Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, Alceste Scalas, and Nobuko Yoshida. Precise subtyping for synchronous multiparty sessions. *J. Log. Algebr. Meth. Program.*, 104:127–173, 2019. doi:10.1016/j.jlamp.2018.12.002.

26 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–112, 1987. doi:10.1016/0304-3975(87)90045-4.

27 Jean-Yves Girard and Yves Lafont. Linear logic and lazy computation. In Hartmut Ehrig, Robert Kowalski, Giorgio Levi, and Ugo Montanari, editors, *TAPSOFT '87*, pages 52–66. Springer, 1987. doi:10.1007/BFb0014972.

28 Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Compututational Logic*, 8, 2007. doi:10.1145/1182613.1182614.

29 Dick Hardt. The OAuth 2.0 authorization framework. standard rfc6749, Internet Engineering Task Force, 2012. URL: https://tools.ietf.org/html/rfc6749.

30 Kohei Honda. Types for dyadic interaction. In *CONCUR'93*, pages 509–523. Springer, 1993. doi:10.1007/3-540-57208-2_35.

31 Kohei Honda, Aybek Mukhamedov, Gary Brown, Tzu-Chun Chen, and Nobuko Yoshida. Scribbling interactions with a formal foundation. In Raja Natarajan and Adegboyega Ojo, editors, *Distributed Computing and Internet Technology*, pages 55–75. Springer, 2011. doi:10.1007/978-3-642-19056-8_4.

32 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '08, pages 273–284. ACM, 2008. doi:10.1145/1328438.1328472.

33 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *J. ACM*, 63(1):9:1–9:67, 2016. doi:10.1145/2827695.

34 Ross Horne. The consistency and complexity of multiplicative additive system virtual. *Scientific Annals of Computer Science*, 25(2):245–316, 2015. doi:10.7561/SACS.2015.2.245.

35 Ross Horne. The sub-additives: A proof theory for probabilistic choice extending linear logic. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019*, volume 131 of *LIPIcs*, pages 23:1–23:16. Schloss Dagstuhl, 2019. doi:10.4230/LIPIcs.FSCD.2019.23.

36 Ross Horne and Alwen Tiu. Constructing weak simulations from linear implications for processes with private names. *Mathematical Structures in Computer Science*, 29(8):1275–1308, 2019. doi:10.1017/S0960129518000452.

37 Naoki Kobayashi. A type system for lock-free processes. *Information and Computation*, 177(2):122–159, 2002. doi:10.1016/S0890-5401(02)93171-8.

38 Julien Lange and Emilio Tuosto. Synthesising choreographies from local session types. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR 2012 – Concurrency Theory*, pages 225–239. Springer, 2012.

39 Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From communicating machines to graphical choreographies. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 221–232. ACM, 2015. doi:10.1145/2676726.2676964.

40 Sam Lindley and J. Garrett Morris. Talking bananas: Structural recursion for session types. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, ICFP 2016, page 434–447. ACM, 2016. doi:10.1145/2951913.2951921.

41 Barbara Liskov and Jeannette M. Wing. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, 16(6):1811–1841, 1994. doi:10.1145/197320.197383.

42 Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51(1):125–157, 1991. doi:10.1016/0168-0072(91)90068-W.

43 Dimitris Mostrous, Nobuko Yoshida, and Kohei Honda. Global principal typing in partially commutative asynchronous sessions. In *Programming Languages and Systems*, pages 316–332. Springer, 2009. doi:10.1007/978-3-642-00590-9_23.

44 Luca Padovani. Session types = intersection types + union types. In Elaine Pimentel, Betti Venneri, and Joe B. Wells, editors, *Proceedings Fifth Workshop on Intersection Types and Related Systems, ITRS 2010, Edinburgh, U.K., 9th July 2010.*, volume 45 of *EPTCS*, pages 71–89, 2010. doi:10.4204/EPTCS.45.6.

45 Luca Padovani. On projecting processes into session types. *Mathematical Structures in Computer Science*, 22(2):237–289, 2012. `doi:10.1017/S0960129511000405`.

46 Luca Padovani. Deadlock and lock freedom in the linear pi-calculus. In *CSL-LICS*, pages 72:1–72:10. ACM Press, 2014. `doi:10.1145/2603088.2603116`.

47 Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996. `doi:10.1017/S096012950007002X`.

48 Alceste Scalas and Nobuko Yoshida. Less is more: multiparty session types revisited. *PACMPL*, 3(POPL):30:1–30:29, 2019. `doi:10.1145/3290343`.

49 Paula Severi and Mariangiola Dezani-Ciancaglini. Observational equivalence for multiparty sessions. *Fundam. Inform.*, 170(1-3):267–305, 2019. `doi:10.3233/FI-2019-1863`.

50 Alwen Tiu. A system of interaction and structure II: The need for deep inference. *Logical Methods in Computer Science*, 2(2), 2006. `doi:10.2168/LMCS-2(2:4)2006`.

A Proof of Theorem 12: well-typed networks are deadlock free

We require the following standard lemmas, which follow by structural induction.

▶ **Lemma 15** (inversion lemma). *In the following, we do not use the subsumption rule.*

- *If* $\vdash P \parallel Q \colon \mathsf{T}$*, there exists* U *and* V *such that* $\mathsf{T} = \mathsf{U} \otimes \mathsf{V}$ *and* $\vdash P \colon \mathsf{U}$ *and* $\vdash Q \colon \mathsf{V}$.
- *If* $\vdash \oplus_{i \in I} !\lambda_i;P_i \colon \mathsf{T}$*, there exists* T_i *such that* $\mathsf{T} = \bigvee_{i \in I} !\lambda_i;T_i$ *and* $\vdash P_i \colon \mathsf{T}_i$.
- *If* $\vdash \Sigma_{i \in I} ?\lambda_i;P_i \colon \mathsf{T}$*, there exists* T_i *such that* $\mathsf{T} = \bigwedge_{i \in I} !\lambda_i;T_i$ *and* $\vdash P_i \colon \mathsf{T}_i$.
- *If* $\vdash \mathtt{rec} X.P \colon \mathsf{T}$*, there exists* U *and* $\mathbf{t}$ *such that* $\mathsf{T} = \mu\mathbf{t}.\mathsf{U}$ *and* $X \colon \mathbf{t} \vdash P \colon \mathsf{U}$.
- *If* $\vdash 1 \colon \mathsf{T}$ *then* $\mathsf{T} = \mathit{OK}$.

▶ **Lemma 16.** *If* $\vdash \mathtt{rec} X.P \colon \mu\mathbf{t}.\mathsf{T}$ *then* $\vdash P\left\{{}^{X.P}/_{X}\right\} \colon \mathsf{T}\left\{{}^{\mu\mathbf{t}.\mathsf{T}}/_{\mathbf{t}}\right\}$.

We also require that race freedom is preserved by the reduction system. This is effectively a subject reduction theorem for the race free property.

▶ **Lemma 17** (race freedom). *If* P *is race free and* $P \longrightarrow Q$*, then* Q *is race free.*

The following condition follows from inverting the type system for race freedom.

▶ **Lemma 18.** *If* $P \parallel Q$ *is race free and* $\vdash P \colon \mathsf{T}$ *and* $\vdash Q \colon \mathsf{U}$*, then if* π *appears in* T*, then* π *does not appear in* U.

Since we employ a reduction semantics, we require that the rules of the structural congruence preserve multiparty compatibility.

▶ **Lemma 19.** *If* $\vdash P \colon \mathit{OK}$ *and* $P \equiv Q$*, then* $\vdash Q \colon \mathit{OK}$.

We also require a subject reduction result, where proofs that $\mathsf{T} \leq \mathsf{OK}$ and race freedom play the role that a global type normally plays in such proofs. Note we avoid the term session fidelity since fidelity is typically expressed in terms of global types [32].

▶ **Lemma 20** (subject reduction). *If* $\vdash P \colon \mathit{OK}$*, and* P *is race free, then for all* Q *such that* $P \longrightarrow Q$*, we have* $\vdash Q \colon \mathit{OK}$.

Proof. If there exists a reduction, we can apply the structural congruence to a process to reach one of the following forms. By Lemma 19, the use of the structural congruence preserves multiparty compatibility.

Case of internal choice. Assume we have $\vdash \oplus_{i \in I} !\lambda_i;P_i \parallel Q \colon \mathsf{OK}$. By Theorem 9, for some T, we have $\vdash \oplus_{i \in I} !\lambda_i;P_i \parallel Q \colon \mathsf{T}$, without using subsumption, and $\mathsf{T} \leq \mathsf{OK}$. Consider the transition $\oplus_{i \in I} !\lambda_i;P_i \parallel Q \longrightarrow !\lambda_k;P_k \parallel Q$, where $k \in I$.

By Lemma 15, we have there exists U_i and V such that $\mathsf{T} = \bigvee_{i \in I} !\lambda_i;\mathsf{U}_i \otimes \mathsf{V}$ and $\vdash P_i \colon \mathsf{U}_i$, for all i, and $\vdash Q \colon \mathsf{V}$. Therefore $\vdash !\lambda_k;P_k \parallel Q \colon !\lambda_i;\mathsf{U}_i \otimes \mathsf{V}$.

Now, since $\bigvee_{i \in I} !\lambda_i;\mathsf{U}_i \mathbin{,} \mathsf{V} \vdash$ is provable and so is $\bigwedge_{i \in I} ?\lambda_i;\overline{\mathsf{U}_i} \mathbin{,} !\lambda_k;\mathsf{U}_k \vdash$, by Theorem 5, $!\lambda_k;\mathsf{U}_k \mathbin{,} \mathsf{V} \vdash$ holds. Hence $!\lambda_k;\mathsf{U}_k \otimes \mathsf{V} \leq \mathsf{OK}$, as required.

Case of external choice. Assume we have $\vdash \Sigma_{i \in I} ?\lambda_i;P_i \parallel !\lambda_k;Q \parallel R \colon \mathsf{OK}$, where $k \in I$ and $\Sigma_{i \in I} ?\lambda_i;P_i \parallel !\lambda_k;Q \parallel R$ is race free. Consider transition $\Sigma_{i \in I} ?\lambda_i;P_i \parallel !\lambda_k;Q \parallel R \longrightarrow P_k \parallel Q \parallel R$.

By Theorem 9, for some T, we have that $\vdash \Sigma_{i \in I} ?\lambda_i;P_i \parallel !\lambda_k;Q \parallel R \colon \mathsf{T}$ holds without using subsumption, and $\mathsf{T} \leq \mathsf{OK}$. By Lemma 15 we have there exists U_i, V and W such that we have $\mathsf{T} = \bigwedge_{i \in I} ?\lambda_i;\mathsf{U}_i \otimes !\lambda_k;\mathsf{V} \otimes \mathsf{W}$ and $\vdash P_i \colon \mathsf{U}_i$, for all i, and $\vdash Q \colon \mathsf{V}$ and $\vdash R \colon \mathsf{W}$. Therefore we have $\vdash P_k \parallel Q \parallel R \colon \mathsf{U}_k \otimes \mathsf{V} \otimes \mathsf{W}$ holds.

Now, consider the proof of $\bigwedge_{i \in I} ?\lambda_i;\mathsf{U}_i \mathbin{,} !\lambda_k;\mathsf{V} \mathbin{,} \mathsf{W} \vdash$. Since we have the type judgements $\vdash \Sigma_{i \in I} ?\lambda_i;P_i \parallel !\lambda_k;Q \colon \bigwedge_{i \in I} ?\lambda_i;\mathsf{U}_i \otimes !\lambda_k;\mathsf{V}$ and $\vdash R \colon \mathsf{W}$ and $\Sigma_{i \in I} ?\lambda_i;P_i \parallel !\lambda_k;Q \parallel R$ is race free, by Lemma 18, neither $!\lambda_i$ nor $?\lambda_k$ appear in W. Hence there are only two possibilities, for **every branch of the proof tree**:

1. Either we eventually reach an application of rule [Prefix], possibly via an application of [Meet] as follows:

$$
\cfrac{\cfrac{\cfrac{\vdots}{[\Theta] \vdash \mathsf{U}_k \;,\; \mathsf{V} \;,\; \Gamma \vdash}}{[\Theta]\, ?\lambda_k;\mathsf{U}_k \;,\; !\lambda_k;\mathsf{V} \;,\; \Gamma \vdash}\;[\textsc{Prefix}]}{\cfrac{\vdots}{\cfrac{[\Theta']\, ?\lambda_k;\mathsf{U}_k \;,\; !\lambda_k;\mathsf{V} \;,\; \Gamma' \vdash}{[\Theta']\, \bigwedge_{i \in I} ?\lambda_i;\mathsf{U}_i \;,\; !\lambda_k;\mathsf{V} \;,\; \Gamma' \vdash}\;[\textsc{Meet}]}}
\\ \vdots
$$

Note, by race freedom, if $\lambda_j = \lambda_k$ then $j = k$, hence only one branch can be selected in rule [Meet] to enable the rule [Prefix]. Hence the above application of rule [Intr] is deterministic.

2. Alternatively, on some path no [Prefix] is ever applied to type $!\lambda_k;\mathsf{V}$ and there is a [Leaf] axiom of the following form, with an corresponding ancestor [Fix-μ] rule as follows:

$$
\cfrac{\cfrac{}{[\Theta' \;\rrbracket\; !\lambda_k;\mathsf{V} \;,\; \mu\mathbf{t}.\mathsf{W}' \;,\; \Gamma]\; !\lambda_k;\mathsf{V} \;,\; \mu\mathbf{t}.\mathsf{W}' \;,\; \Gamma \vdash}\;[\textsc{Leaf}]}{\cfrac{\vdots}{\cfrac{[\Theta \;\rrbracket\; !\lambda_k;\mathsf{V} \;,\; \mu\mathbf{t}.\mathsf{W}' \;,\; \Gamma]\; !\lambda_k;\mathsf{V} \;,\; \mathsf{W}'\left\{{}^{\mu\mathbf{t}.\mathsf{W}'}\!/\mathbf{t}\right\} \;,\; \Gamma \vdash}{[\Theta]\; !\lambda_k;\mathsf{V} \;,\; \mu\mathbf{t}.\mathsf{W}' \;,\; \Gamma \vdash}\;[\textsc{Fix-}\mu]}}
$$

In this case, by the participant condition in the race free condition, each λ_j such that $j \in I$ can only match an output in the type $!\lambda_k;\mathsf{V}$. Hence there must also be no [Prefix] applied to any λ_i in $\bigwedge_{i \in I} ?\lambda_i;\mathsf{U}_i$ between the [Leaf] and the corresponding [Fix-μ]. Hence either $\bigwedge_{i \in I} ?\lambda_i;\mathsf{U}_i$ appears in Γ, or there is some $j \in I$ such that $?\lambda_j;\mathsf{U}_j$ for $j \in I$ appears in Γ.

In paths in the proof satisfying the first case above, simply remove the relevant instance of the rule [Intr] below the rule in the proof, replace $\bigwedge_{i \in I} ?\lambda_i;\mathsf{U}_i$ and $!\lambda_k;\mathsf{V}$ with U_k and V.

In paths in the proof satisfying the second case above where both $\bigwedge_{i \in I} ?\lambda_i;\mathsf{U}_i$ and $!\lambda_k;\mathsf{V}$ are never touched, simply replacing these formulae with U_k and V everywhere in the given path. In cases where $?\lambda_j;\mathsf{U}_i$ appears in Γ, there must be an instance of rule [Join] below the rule [Fix-μ] that introduced Γ or the following form.

$$
\frac{[\Theta'']\; !\lambda_k;\mathsf{V} \;,\; ?\lambda_j;\mathsf{U}_j \;,\; \Gamma'' \vdash}{[\Theta'']\; !\lambda_k;\mathsf{V} \;,\; \bigwedge_{i \in I} ?\lambda_i;\mathsf{U}_i \;,\; \Gamma'' \vdash}
$$

Since, by the participant condition, we know that in this path we never apply [Prefix] to λ_j, we can safely remove the above rule instances from the proof and replace $?\lambda_j;\mathsf{U}_j$ with U_k along that path.

After applying the above proof transformation, we obtain a proof of $\mathsf{U}_k \;,\; \mathsf{V} \;,\; \mathsf{W} \vdash$. Hence $\mathsf{U}_k \otimes \mathsf{V} \otimes \mathsf{W} \leq \mathtt{OK}$ as required.

Case of fixed points. Assume $\vdash \mathtt{rec}X.P \parallel Q \colon \mathtt{OK}$ holds. By Theorem 9, for some T, we have $\vdash \mathtt{rec}X.P \parallel Q \colon \mathsf{T}$, without using subsumption, and $\mathsf{T} \leq \mathtt{OK}$. Consider the transition $\mathtt{rec}X.P \parallel Q \longrightarrow P\left\{{}^{\mathtt{rec}X.P}\!/X\right\} \parallel Q$.

By Lemma 15, we have there exist types U and V and type variable $\mathbf{t}$ such that $\mathsf{T} = \mu\mathbf{t}.\mathsf{U} \otimes \mathsf{V}$ and $\vdash \mathtt{rec}X.P \colon \mu\mathbf{t}.\mathsf{U}$ and $\vdash Q \colon \mathsf{V}$. Now, by Lemma 16, $\vdash P\left\{{}^{\mathtt{rec}X.P}/_{X}\right\} \colon \mathsf{U}\left\{{}^{\mu\mathbf{t}.\mathsf{U}}/_{\mathbf{t}}\right\}$. Therefore, we have $\vdash P\left\{{}^{\mathtt{rec}X.P}/_{X}\right\} \parallel Q \colon \mathsf{U}\left\{{}^{\mu\mathbf{t}.\mathsf{U}}/_{\mathbf{t}}\right\} \otimes \mathsf{V}$.

Now, since $\vdash \mu\mathbf{t}.\mathsf{U} \,,\, \mathsf{V}$ is provable and $\mu\mathbf{t}.\mathsf{U}, \mathsf{U}\left\{{}^{\mu\mathbf{t}.\mathsf{U}}/_{\mathbf{t}}\right\} \vdash$ is provable, by Theorem 5, we have $\mathsf{U}\left\{{}^{\mu\mathbf{t}.\mathsf{U}}/_{\mathbf{t}}\right\} \,,\, \mathsf{V} \vdash$ is provable. Hence $\mathsf{U}\left\{{}^{\mu\mathbf{t}.\mathsf{U}}/_{\mathbf{t}}\right\} \otimes \mathsf{V} \leq \mathtt{OK}$, as required. ◀

▶ **Theorem 21** (Theorem 12)**.** *Any race-free multiparty-compatible network is deadlock free.*

Proof. Assume $\vdash P \colon \mathtt{OK}$ holds and P is race free. Consider the form of P. Either P has a fixed point or internal choice at the head of a process, hence is ready to act. Hence, there exists Q such that $P \longrightarrow Q$. Otherwise we have a process equivalent to the following form.

$$!\lambda_1;Q_1 \parallel \ldots \parallel !\lambda_m;Q_m \parallel \Sigma_{i \in I_1} ?\lambda^1_i;R^1_i \parallel \ldots \parallel \Sigma_{i \in I_n} ?\lambda^n_i;R^n_i \parallel 1 \parallel \ldots \parallel 1$$

There are two cases to consider as follows.

In the first case, $m = n = 0$; hence we have $P = 1 \parallel \ldots \parallel 1$. Therefore, $P \equiv 1$ and hence the processes is successfully terminated.

Otherwise, observe, by Theorem 9, there exists T such that $\vdash P \colon \mathsf{T}$ without using subsumption and $\mathsf{T} \leq \mathtt{OK}$. Also, observe, by Theorem 15, there exists U_i and V^i_i such that $\mathsf{T} = !\lambda_1;\mathsf{U}_1 \otimes \ldots \otimes !\lambda_m;\mathsf{U}_m \otimes \bigwedge_{i \in I_1} ?\lambda^1_i;\mathsf{V}^1_i \parallel \ldots \otimes \bigwedge_{i \in I_n} ?\lambda^n_i;\mathsf{V}^n_i \otimes \mathtt{OK} \otimes \ldots \otimes \mathtt{OK}$ and $\vdash Q_k \colon \mathsf{U}_k$ and $\vdash R^\ell_j \colon \mathsf{V}^\ell_j$, for all j, k and ℓ.

In the proof of $T \vdash$, there must be at least one application of the rule [Prefix]. Due to the absence of $\parr$ in T, the only other rules that may be applied before the bottommost instances of rule [Prefix] are the rules [Par] and [Meet]. In order to apply the rule [Prefix], there exists j, k and ℓ such $j \in I_\ell$ and $\lambda_k = \lambda^\ell_j$, allowing a proof tree of the following form.

$$\cfrac{\cfrac{\cfrac{\cfrac{[\Theta]\, \mathsf{T}_k \,,\, \mathsf{U}^\ell_i \,,\, \Gamma \vdash}{[\Theta]\, !\lambda_k;\mathsf{T}_k \,,\, ?\lambda^\ell_j;\mathsf{U}^\ell_j \,,\, \Gamma \vdash}}{\vdots}}{\cfrac{[\Theta]\, !\lambda_k;\mathsf{T}_k \,,\, ?\lambda^\ell_j;\mathsf{U}^\ell_j \,,\, \Gamma \vdash}{[\Theta]\, !\lambda_k;\mathsf{T}_k \,,\, \bigwedge_{i \in I_\ell} ?\lambda^\ell_i;\mathsf{U}^\ell_i \,,\, \Gamma \vdash}}}{\cfrac{\vdots}{T \vdash}}$$

Thus, simply due to the existence of such a matching pair of inputs and outputs, we have a transition of the form.

$$\begin{array}{l} !\lambda_1;Q_1 \parallel \ldots \parallel !\lambda_k;Q_k \parallel \ldots \parallel !\lambda_m;Q_m \\ \parallel \Sigma_{i \in I_1} ?\lambda^1_i;R^1_i \parallel \ldots \parallel \Sigma_{i \in I_\ell} ?\lambda^n \ell_i;R^\ell_i \parallel \ldots \parallel \\ \Sigma_{i \in I_n} ?\lambda^n_i;R^n_i \parallel 1 \parallel \ldots \parallel 1 \end{array} \longrightarrow \begin{array}{l} !\lambda_1;Q_1 \parallel \ldots \parallel Q_k \parallel \ldots \parallel !\lambda_m;Q_m \\ \parallel \Sigma_{i \in I_1} ?\lambda^1_i;R^1_i \parallel \ldots \parallel R^\ell_j \parallel \ldots \\ \parallel \Sigma_{i \in I_n} ?\lambda^n_i;R^n_i \parallel 1 \parallel \ldots \parallel 1 \end{array}$$

Thus we certainly have that either $P \equiv 1$ or there exists Q such that $P \longrightarrow Q$.

Finally, by Lemma 20, since R is race free, we have that for all R such that $P \longrightarrow R$, $\vdash R \colon \mathtt{OK}$ and furthermore, by Lemma 17, R is race free, as required. Hence, deadlock freedom is established by coinduction. ◀

B The Precise Relationship to Linear Logic

For a self-contained presentation, we summarise the related non-commutative logic [15] on which this work builds, formulated in the calculus of structures [28]. We adjust the syntax to match the body of the paper. The rules of MAV [34] are presented as in Fig. 7, where the calculus of structures allows rules to be applied in any context $\mathcal{C}\{\ \cdot\ \}$ and the structural congruence $\equiv$ can be applied at any point in a proof.

$$\frac{}{\textsf{OK} \vdash}\ \text{success} \qquad \frac{\mathcal{C}\{\ \textsf{OK}\ \} \vdash}{\mathcal{C}\{\ !\lambda \otimes ?\lambda\ \} \vdash}\ \text{atomic interaction}$$

$$\frac{\mathcal{C}\{\ (\mathsf{T} \otimes \mathsf{V})\ ;\ (\mathsf{U} \otimes \mathsf{W})\ \} \vdash}{\mathcal{C}\{\ (\mathsf{T}\ ;\ \mathsf{U}) \otimes (\mathsf{V}\ ;\ \mathsf{W})\ \} \vdash}\ \text{seq} \qquad \frac{\mathcal{C}\{\ \mathsf{T} \parr (\mathsf{U} \otimes \mathsf{V})\ \} \vdash}{\mathcal{C}\{\ (\mathsf{T} \parr \mathsf{U}) \otimes \mathsf{V}\ \} \vdash}\ \text{switch}$$

$$\frac{\mathcal{C}\{\ (\mathsf{T} \vee \mathsf{V})\ ;\ (\mathsf{U} \vee \mathsf{W})\ \} \vdash}{\mathcal{C}\{\ (\mathsf{T}\ ;\ \mathsf{U}) \vee (\mathsf{V}\ ;\ \mathsf{W})\ \} \vdash}\ \text{medial} \qquad \frac{\mathcal{C}\{\ (\mathsf{T} \otimes \mathsf{U}) \vee (\mathsf{T} \otimes \mathsf{V})\ \} \vdash}{\mathcal{C}\{\ \mathsf{T} \otimes (\mathsf{U} \vee \mathsf{V})\ \} \vdash}\ \text{external}$$

$$\frac{\mathcal{C}\{\ \mathsf{T}\ \} \vdash}{\mathcal{C}\{\ \mathsf{T} \wedge \mathsf{U}\ \} \vdash}\ \text{left} \qquad \frac{\mathcal{C}\{\ \mathsf{U}\ \} \vdash}{\mathcal{C}\{\ \mathsf{T} \wedge \mathsf{U}\ \} \vdash}\ \text{right} \qquad \frac{\mathcal{C}\{\ \textsf{OK}\ \} \vdash}{\mathcal{C}\{\ \textsf{OK} \vee \textsf{OK}\ \} \vdash}\ \text{tidy}$$

$$\begin{array}{ccc}
(\mathsf{T} \parr \mathsf{U}) \parr \mathsf{V} \equiv \mathsf{T} \parr (\mathsf{U} \parr \mathsf{V}) & \textsf{OK}\ ;\ \mathsf{T} \equiv \mathsf{T} & (\mathsf{T} \otimes \mathsf{U}) \otimes \mathsf{V} \equiv \mathsf{T} \otimes (\mathsf{U} \otimes \mathsf{V}) \\
\mathsf{T} \parr \textsf{OK} \equiv \mathsf{T} & \mathsf{T}\ ;\ \textsf{OK} \equiv \mathsf{T} & \mathsf{T} \otimes \textsf{OK} \equiv \mathsf{T} \\
\mathsf{T} \parr \mathsf{U} \equiv \mathsf{U} \parr \mathsf{T} & (\mathsf{T}\ ;\ \mathsf{U})\ ;\ \mathsf{V} \equiv \mathsf{T}\ ;\ (\mathsf{U}\ ;\ \mathsf{V}) & \mathsf{T} \otimes \mathsf{U} \equiv \mathsf{U} \otimes \mathsf{T}
\end{array}$$

Figure 7 Inference and structural rules for proof system MAV (formalising provability of duals).

We extend the notion of a co-type to local types with sequential composition.

$$\overline{(\mathsf{T} \wedge \mathsf{U})} = \overline{\mathsf{T}} \vee \overline{\mathsf{U}} \qquad \overline{(\mathsf{T} \vee \mathsf{U})} = \overline{\mathsf{T}} \wedge \overline{\mathsf{U}} \qquad \overline{\mathsf{T} \parr \mathsf{U}} = \overline{\mathsf{T}} \otimes \overline{\mathsf{U}} \qquad \overline{\mathsf{T} \otimes \mathsf{U}} = \overline{\mathsf{T}} \parr \overline{\mathsf{U}}$$

$$\overline{(\mathsf{T}\ ;\ \mathsf{U})} = \overline{\mathsf{T}}\ ;\ \overline{\mathsf{U}} \qquad \overline{\textsf{OK}} = \textsf{OK} \qquad \overline{!\lambda} = ?\lambda \qquad \overline{?\lambda} = !\lambda$$

Notice the only difference compared to the co-type transformation for Session (Def. 2) is that any type may appear to the left of sequential composition, not only an atomic send or receive action. The following result generalises *cut elimination* to the calculus of structures.

▶ **Theorem 22** (Horne 2015 [34]). *In the system in Fig. 7, if* $\mathcal{C}\{\ \overline{\mathsf{T}} \parr \mathsf{T}\ \} \vdash$ *holds then we can construct a proof of* $\mathcal{C}\{\ \textsf{OK}\ \} \vdash$.

The related work [15, 34], from which the above is extracted, clarifies that, as for Session in the body of this paper, MAV defines a rich notion of multiparty subtyping and compatibility.

The following result formally relating MAV and Session is a corollary of cut elimination (each direction of the implication follows from cut elimination in one of the two systems).

▶ **Corollary 23.** *If* T *is a session type, as in Def. 1 but without fixed points, then* $\mathsf{T} \vdash$ *in* Session *if and only if* $\mathsf{T} \vdash$ *in System* MAV.

Finally, observe that MAV is a conservative extension of linear logic with mix and, the above corollary proves the finite fragment of Session is also a fragment of MAV.