Behavioural Analysis of Sessions using the Calculus of Structures

Gabriel Ciobanu and Ross Horne

Institute of Computer Science, Romanian Academy, Iasi, Romania
Faculty of Information Technology, Kazakh-British Technical University, Almaty, Kazakhstan
gabriel@info.uaic.ro ross.horne@gmail.com

Abstract. This paper describes an approach to the behavioural analysis of sessions. The approach is made possible by the calculus of structures — a deep inference proof calculus, generalising the sequent calculus, where inference rules are applied in any context. The approach involves specifications of global and local sessions inspired by the Scribble language. The calculus features a novel operator that synchronises parts of a protocol that must be treated atomically. Firstly, the calculus can be used to determine whether local sessions can be compose in a type safe fashion such that sessions are capable of successfully completing. Secondly, the calculus defines a subtyping relation for sessions that allows causal dependencies to be weakened while retaining termination potential. Consistency and complexity results follow from proof theory.

1 Introduction

This work is the first to draw connections between a calculus that originates in proof theory, namely *the calculus of structures* [9], and the static analysis of sessions using *session types* [10, 12].

In many systems, a protocol is initiated by opening a session between the parties involved. Participants in a protocol typically exchange a number of messages before closing the session. Each session can be characterised by the types of messages exchanged and also the order in which messages are sent. Such information can be captured in a session type. The session type declares a specification that can be used for both static and runtime analysis of the protocol concerned.

On the other hand, the calculus of structure is a proof calculus that was originally discovered by studying non-commutative operators, i.e. operators, say op, where A op B is not the same as B op A. Such operators are useful for expressing causal dependencies, such as the concept of A happening before B, which is clearly a non-commutative concept. There are simple and natural connections between the calculus of structures and session types that can be understood immediately:

– The calculus of structures is a *term rewriting system modulo an equational theory* [16]. Term rewriting systems modulo an equational theory are analogous to reduction systems modulo a structural congruence as typically used

to express the operational semantics that govern the behaviour of session types.
- A *proof* in the calculus of structures is a special derivation that may *terminate successfully*. Session types can be used to analyse whether a family of participants in a session may together successfully complete the session, in which case they are multiparty compatible [5].

Further striking connections between session types and the calculus of structures are forthcoming. In the paper [10], which initiated interest in session types, Honda introduces the notion of a co-type, which respects the following properties resembling De Morgan dualities:

$$\sim(P \,\&\, Q) = \sim P \oplus \sim Q \qquad \sim(P \oplus Q) = \sim P \,\&\, \sim Q \qquad \sim(P\,;Q) = \sim P\,;\sim Q$$

It is no secret that the first two De Morgan dualities above were inspired by the additive operators *with* and *plus* (& and $\oplus$) of linear logic. Abramsky had already suggested [1] that & and $\oplus$ could be interpreted as a *external* choice and *internal* choice respectively.

More recently, proof theorists have independently devised proof calculi, by using the calculus of structures [9], that exhibit the De Morgan duality for sequential composition above. Because the De Morgan dual of sequential composition is also sequential composition, it is considered to be a *self-dual* operator. In this work, we argue that the self-dual non-commutative operator found in session types and the self-dual non-commutative operator found in the calculus of structures are essentially the same operator.

Section 2 introduces the running example of a simple two-phase commit protocol in a language inspired by the session type based language Scribble. The syntax of global and local types are defined as well as the projection from global types to local types. Section 3 defines the semantics of local types in the calculus of structures, and presents consistency and compatibility results. Section 4 compares our work to the body of work bridging session types and proof calculi, and highlights open problems.

2 A Core Calculus Inspired by Scribble

We begin with a concrete example of a ubiquitous protocol from distributed computing. Two-phase commit (2PC) ensures the atomicity of a transaction involving data persisted across distributed nodes. The language we introduce is heavily influenced by the Scribble language [11], which can be used to specify the global behaviour of the two-phase commit protocol. The Scribble language is an implementation of multiparty asynchronous session types [12, 5], with a syntax deliberately chosen to appeal to software engineers that uses curly braces for disambiguation.

The example protocol is presented in Fig. 1. The protocol describes interactions between three parties: a *Client* that initiates a transaction, a *Leader* that

```
par p_begin(Payload) from Client to Participant
and l_begin(Payload) from Client to Leader ;
prepare(Timestamp) from Participant to Leader ;
par c_commit(Timestamp) from Leader to Client
and p_commit(Timestamp) from Leader to Participant
```

Fig. 1. A global protocol for client driven two-phase commit.

coordinates the transaction, and a *Participant* that must coordinate with the leader.

Global types of the form *prepare*(*Timestamp*) `from` *Participant* `to` *Leader* mean that role *Participant* sends a message of type *Timestamp* to the role *Leader*, using the channel *prepare*. A channel is assumed to be some messaging middleware that the sender passes a message to, and the receiver listens on. The type constructor `par` ...`and`... represents the parallel composition of two protocols. For example, in 2PC the leader sends commit timestamps to the *Client* and *Participant* in parallel. The semi-colon represents sequential composition. We assume that `par` takes higher operator precedence than semi-colon.

The variant of the Two Phase Commit protocol described in the global protocol is a client-driven two phase commit with one participant. The *Client* initiates the protocol by sending a payload to the *Leader* and *Participant*. The leader and the participant manage a disjoint range of keys associated with data. The payload for the leader and the client contains updates to apply at each respective node. When the participant receives the payload, the participant acquires locks for the relevant data, logs the transaction, picks a timestamp that is greater than the timestamp applied to any previous transaction, then sends the timestamp to the leader. Upon receiving the prepare timestamp from the *Participant* and the begin message from the *Client*, the leader locks its own data and picks a timestamp greater than the prepare timestamp from the *Participant* and greater than the timestamp applied to any transaction at the *Leader*. The leader then logs its own updates and the timestamp. Finally, the *Leader* notifies the *Client* and *Participant* about the timestamp chosen for the whole transaction, the *Participant* logs the timestamp and all locks are released.

Session types, such as Scribble, can be used for the design, implementation and verification of distributed systems. A methodology for using session types is as follows. Firstly, a systems analyst designs the global protocol, which describes the message exchanges between all parties in a distributed system. Secondly, the global protocol is automatically projected to local protocols, as presented in Fig. 2 for 2PC, where local protocols specify permitted patterns of sends and receives of messages for each role. Because the projection to local protocols is performed automatically such that a certain semantics is respected, the systems analyst knows that each local protocol is correct with respect to the global protocol. Verified automation eliminates human error that can be introduced when projection is performed manually using the intuition of the systems analyst.

Client:
$\texttt{par}\ {\sim}p\_begin\,(Payload)\ \texttt{to}\ Participant$
$\texttt{and}\ {\sim}l\_begin\,(Payload)\ \texttt{to}\ Leader\,;$
$commit\,(Timestamp)\ \texttt{from}\ Leader$

Participant:
$p\_begin\,(Payload)\ \texttt{from}\ Client\,;$
${\sim}prepare\,(Timestamp)\ \texttt{to}\ Leader\,;$
$p\_commit\,(Timestamp)\ \texttt{from}\ Leader$

Leader:
$l\_begin\,(Payload)\ \texttt{from}\ Client\,;$
$prepare\,(Timestamp)\ \texttt{from}\ Participant\,;$
$\texttt{par}\ {\sim}p\_commit\,(Timestamp)\ \texttt{to}\ Participant$
$\texttt{and}\ {\sim}c\_commit\,(Timestamp)\ \texttt{to}\ Client$

Fig. 2. Local types for roles *Client*, *Participant* and *Leader* projected from Fig. 1.

A local protocol can fulfil several roles in the software engineering process. Firstly, a local protocol can be used as a reference for an engineer who is responsible for implementing the node that performs the role described in the protocol. Furthermore, the local protocol can be used to verify that the engineer's implementation does indeed conform to the given protocol. For some languages, there exist extensions [19, 15] that enable the code itself to be statically analysed. For situations where there are either no static type checking tools or we do not have access to the code, runtime monitors can be automatically generated that observe the behaviour of nodes while they execute to detect whether a protocol violates its specification [14].

The syntax for global and local types is presented in Fig. 3. The syntax is heavily influenced by the global and local protocols of Scribble [11], but with some significant differences:

- We include a complementation operator over atoms, written as a tilde prefix, where atoms represent the separate send and receive events. Atoms and their complements interact to compose local protocols with channels.
- We also include a binary operator `sync`, which ensures that two events occur atomically. Types joined by `sync` "appear" to happen simultaneously, using separate resources, hence they cannot be interleaved. We use `sync` to capture synchrony in the transport mechanism, but we envision that it can form an extension to Scribble where complex atomic transactions are modelled.

Projection. The projection from global G to local types for a given role R is defined as follows, written $G \upharpoonright_R$.

$$(G_0\,;G_1)\upharpoonright_R = (G_0\upharpoonright_R)\,;(G_1\upharpoonright_R)$$
$$(\texttt{par}\ G_0\ \texttt{and}\ G_1)\upharpoonright_R = \texttt{par}\ (G_0\upharpoonright_R)\ \texttt{and}\ (G_1\upharpoonright_R)$$
$$c\,(S)\ \texttt{from}\ P\ \texttt{to}\ Q\upharpoonright_R = \begin{cases} {\sim}c\,(S)\ \texttt{to}\ Q & if P = R \\ c\,(S)\ \texttt{from}\ P & if Q = R \\ \{\} & otherwise \end{cases}$$
$$(\texttt{choice at}\ P\ G_0\ \texttt{or}\ G_1)\upharpoonright_R = \begin{cases} (G_0\upharpoonright_R)\ \texttt{or}\ (G_1\upharpoonright_R) & if P \neq R \\ (G_0\upharpoonright_R)\ \&\ (G_1\upharpoonright_R) & if P = R \end{cases}$$

$$S ::= \mathit{int} \mid \mathit{string} \mid S \times S \mid \ldots \quad \text{sorts}$$

c channel $\quad P$ role

$$A ::= c(S)\ \texttt{to}\ P \quad \text{send to}$$
$$\mid c(S)\ \texttt{from}\ P \quad \text{receive from}$$
$$\mid \sim A \quad \text{complementation}$$

$$G ::= c(S)\ \texttt{from}\ P\ \texttt{to}\ P \quad \text{values}$$
$$\mid G\,;\,G \quad \text{seq}$$
$$\mid \texttt{par}\ G\ \texttt{and}\ G \quad \text{par}$$
$$\mid \texttt{choice at}\ P\ G\ \texttt{or}\ G \quad \text{choice}$$

$$T ::= \{\} \quad \text{unit}$$
$$\mid A \quad \text{atom}$$
$$\mid T\,;\,T \quad \text{seq}$$
$$\mid \texttt{par}\ T\ \texttt{and}\ T \quad \text{par}$$
$$\mid \texttt{sync}\ T\ \texttt{and}\ T \quad \text{sync}$$
$$\mid T\ \texttt{or}\ T \quad \text{internal choice}$$
$$\mid T\,\&\,T \quad \text{external choice}$$

Fig. 3. Syntax of global types (G), local types (T), atoms (A) and sorts (S).

The projection generates a local type for each role that appears in a global type. Notice that send events are prefixed with the complementation operator, which makes explicit the contravariant nature of sending on a channel [20] which we explain further when we introduce subtyping in the next section.

For simplicity, we assume synchronous communication where messages are received and delivered atomically. Channels are handled explicitly as special local types defined by the following projection from global types to local types. The projection below maps a message exchange to a send event synchronised with a receive event. Notice that polarities of atoms, indicated by the compliment operator, are opposite to atoms in the projection for roles.

$$d(S)\ \texttt{from}\ P\ \texttt{to}\ Q\ \upharpoonright_c = \begin{cases} \texttt{sync}\ c(S)\ \texttt{to}\ Q\ \texttt{and}\ \sim c(S)\ \texttt{from}\ P & \text{if } c = d \\ \{\} & \text{otherwise} \end{cases}$$
$$(\texttt{choice at}\ P\ T\ \texttt{or}\ U)\ \upharpoonright_c = (T\ \upharpoonright_c)\ \texttt{or}\ (U\ \upharpoonright_c)$$
$$(G_0\,;G_1)\ \upharpoonright_c = \texttt{par}\ (G_0\ \upharpoonright_c)\ \texttt{and}\ (G_1\ \upharpoonright_c)$$
$$(\texttt{par}\ G_0\ \texttt{and}\ G_1)\ \upharpoonright_c = \texttt{par}\ (G_0\ \upharpoonright_c)\ \texttt{and}\ (G_1\ \upharpoonright_c)$$

Asynchronous channels and channels with queues of up to length two can also be handled by this framework. We refer to [12] for constraints restricting the order of events in asynchronous systems and avoiding races.

3 The Subtype System and Multiparty Compatibility

The semantics of local types is defined by a term rewriting system modulo an equational theory. The semantics can be used to define both a subtype relation over local types and the notion of multiparty compatibility.

The rewrite rules and equational theory are presented in Fig. 4. As standard for term rewriting, the equations can be applied at any point in a derivation, and the rules can be applied in any context, where a context $\mathcal{C}\{\ \}$ is any local

type with one hole in which any local type can be plugged. Thus we have the following implicit rules:

$$\mathcal{C}\{\ T\ \} \longrightarrow \mathcal{C}\{\ U\ \} \text{ only if } T \longrightarrow U \quad \text{context closure}$$

$$T \longrightarrow U \text{ only if } T \equiv U \quad \text{congruence}$$

We name the term rewriting system in Fig 4 multiplicative additive system virtual (MAV) since our system combines systems multiplicative additive linear logic with mix (MALL) [17] and basic system virtual (BV) [9], both of which are consistent proof calculi. The equational system ensures that ; is a monoid and `par` and `sync` are commutative monoids. We briefly explain the rewrite rules.

- The atomic interaction rules enable a negative atom and positive atom to annihilate each other, whenever the carried sort of the negated atom is a subsort of the carried sort of the positive atom. Assuming we have a subsorting system, that defines any preorder, we extend the system atoms as follows.

  $$c\,(S0)\ \texttt{to}\ P \leq c\,(S1)\ \texttt{to}\ P \text{ only if } S0 \leq S1$$
  $$c\,(S0)\ \texttt{from}\ P \leq c\,(S1)\ \texttt{from}\ P \text{ only if } S0 \leq S1$$

  For example, the subsorting can be given by subtyping for XML Schema [2].
- The switch rule generalises the rule for the tensor product in linear logic. The rule focusses a parallel composition on where an interaction takes place.
- The seq rule arises in the theory of pomsets [8]. The rule strengthens causal dependencies to introduce a barrier across two parallel threads.
- The left choice and right choice rules represent an internal choice where the protocol has control over the branch to select. The external choice rule, represents when we cannot determine at compile time what branch will be selected; hence must analyse both possibilities. The tidy rule simply acknowledges when two branches in an external choice have successfully closed. The medial is essential for the co-existence of seq and external choice.

We extend the complementation operator over atoms to all local types using the following function that transforms a type into its *co-type* [10].

$$\sim(P\ \&\ Q) = \sim P\ \texttt{or}\ \sim Q \qquad \sim(P\ \texttt{or}\ Q) = \sim P\ \&\ \sim Q$$

$$\sim\texttt{par}\ T\ \texttt{and}\ U = \texttt{sync}\ \sim T\ \texttt{and}\ \sim U \qquad \sim\texttt{sync}\ T\ \texttt{and}\ U = \texttt{par}\ \sim T\ \texttt{and}\ \sim U$$

$$\sim(T\ ;\ U) = \sim T\ ;\ \sim U \qquad \sim\{\} = \{\} \qquad \sim\sim A = A$$

The above function transforms any local type into a local type in *negation normal form*, where complementation applies only to atoms, as permitted by the syntax in Fig. 3. We deliberately do not include complementation for arbitrary types in the syntax for local types, since the contravariant nature of complementation complicates the rewriting system without any gain in expressive power [9].

In the calculus of structures a *proof* is a special derivation that reduces to the unit type $\{\}$, representing a successfully completed session.

$$\texttt{par} \sim A \texttt{ and } B \longrightarrow \{\} \text{ only if } A \leq B \quad \text{atomic interaction}$$

$$\texttt{par } \{ \texttt{ sync } T \texttt{ and } U \} \texttt{ and } V \longrightarrow \texttt{sync } T \texttt{ and } \{ \texttt{ par } U \texttt{ and } V \} \quad \text{switch}$$

$$\texttt{par } \{ T \ ; \ U \} \texttt{ and } \{ V \ ; \ W \} \longrightarrow \{ \texttt{ par } T \texttt{ and } V \} \ ; \ \{ \texttt{ par } U \texttt{ and } W \} \quad \text{seq}$$

$$T \texttt{ or } U \longrightarrow T \quad \text{left choice} \qquad T \texttt{ or } U \longrightarrow U \quad \text{right choice} \qquad \{\} \,\&\, \{\} \longrightarrow \{\} \quad \text{tidy}$$

$$\texttt{par } T \texttt{ and } \{ U \,\&\, V \} \longrightarrow \{ \texttt{ par } T \texttt{ and } U \} \,\&\, \{ \texttt{ par } T \texttt{ and } V \} \quad \text{external choice}$$

$$\{ T \ ; \ U \} \,\&\, \{ V \ ; \ W \} \longrightarrow \{ T \ \& \ V \} \ ; \ \{ U \ \& \ W \} \quad \text{medial}$$

$$\begin{gathered}
\texttt{par } \{ \texttt{ par } T \texttt{ and } U \} \texttt{ and } V \equiv \texttt{par } T \texttt{ and } \{ \texttt{ par } U \texttt{ and } V \} \\
\texttt{sync } \{ \texttt{ sync } T \texttt{ and } U \} \texttt{ and } V \equiv \texttt{sync } T \texttt{ and } \{ \texttt{ sync } U \texttt{ and } V \} \\
\texttt{sync } T \texttt{ and } \{\} \equiv T \qquad \texttt{par } T \texttt{ and } \{\} \equiv T \\
\{ T \,; U \} \ ; \ V \equiv T \,; \ \{ U \ ; \ V \} \qquad \{\} \ ; \ T \equiv T \qquad T \ ; \ \{\} \equiv T \\
\texttt{par } T \texttt{ and } U \equiv \texttt{par } U \texttt{ and } T \qquad \texttt{sync } T \texttt{ and } U \equiv \texttt{sync } U \texttt{ and } T
\end{gathered}$$

Fig. 4. Term rewriting system modulo an equational theory for local types.

Definition 1. *If for any type $T \longrightarrow \{\}$, according to the term rewriting system in Fig. 4, then we write $\vdash T$, and say that T is provable.*

The following result is a generalisation of a consistency result called *cut elimination* that appears commonly in proof theory.

Theorem 1. *Suppose that there is a proof using the extra rules:*

- $\texttt{par} \sim T \texttt{ and } T \longrightarrow \{\}$ *(interact)*
- $\{\} \longrightarrow \texttt{sync} \sim T \texttt{ and } T$ *(co-interact)*

Given such a proof, a new proof can be constructed that uses only the rules in Fig. 4. We say that the rules interact and co-interact are admissible.

The main results in this paper are corollaries of the above proof-theoretic result. The proof of Theorem 1 involves a technique known as *splitting* introduced in [9]. Choice operators, known as additives, are handled using techniques similar to Theorem 6 in [4], for which we require the following notion of a killing context.

Definition 2. *A killing context $\mathcal{T}\{ \cdot, \cdot, \ldots, \cdot \}$ is an n-ary context such that*

$$\mathcal{T}\{ \ \} ::= \{ \cdot \} \mid \mathcal{T}\{ \ \} \,\&\, \mathcal{T}\{ \ \}$$

where $\{ \cdot \}$ is a hole into which any local type can be plugged.

The splitting lemma below simulates sequent calculus style rules in a context where the root formula is a parallel composition. The proof of the splitting lemma is quite involved, so receive special attention in a companion paper [13].

Lemma 1 (Splitting). *The following statements hold.*

- *For any atom A, if $\vdash \mathtt{par} \sim A \ \mathtt{and}\ T$, then there exist atoms B_1,B_2,...,B_n such that $A \leq B_i$ for $1 \leq i \leq n$ and n-ary killing context $\mathcal{T}\{\ \}$ where $T \longrightarrow \mathcal{T}\{\ B_1, B_2, \ldots, B_n\ \}$.*
- *For any atom A, if $\vdash \mathtt{par}\ A\ \mathtt{and}\ T$, then there exist atoms B_1,B_2,...,B_n such that $B_i \leq A$ for $1 \leq i \leq n$ and n-ary killing context $\mathcal{T}\{\ \}$ where $T \longrightarrow \mathcal{T}\{\ \sim B_1, \sim B_2, \ldots, \sim B_n\ \}$.*
- *If $\vdash \mathtt{par}\ \{T \,\&\, U\}\ \mathtt{and}\ V$, then $\vdash \mathtt{par}\ T\ \mathtt{and}\ V$ and $\vdash \mathtt{par}\ U\ \mathtt{and}\ V$.*
- *If $\vdash \mathtt{par}\ \{T\ \mathtt{or}\ U\}\ \mathtt{and}\ V$, then there exist local types W_i such that either $\vdash \mathtt{par}\ T\ \mathtt{and}\ W_i$ or $\vdash \mathtt{par}\ U\ \mathtt{and}\ W_i$, for $1 \leq i \leq n$, and n-ary killing context $\mathcal{T}\{\ \}$ where $V \longrightarrow \mathcal{T}\{\ W_1, W_2, \ldots, W_n\ \}$.*
- *If $\vdash \mathtt{par}\ \{\ \mathtt{sync}\ S\ \mathtt{and}\ T\ \}\ \mathtt{and}\ U$, then there exist local types V_i and W_i such that $\vdash \mathtt{par}\ S\ \mathtt{and}\ V_i$ and $\vdash \mathtt{par}\ T\ \mathtt{and}\ W_i$, for $1 \leq i \leq n$, and n-ary killing context $\mathcal{T}\{\ \}$ where $U \longrightarrow \mathcal{T}\{\ \mathtt{par}\ V_1\ \mathtt{and}\ W_1, \ldots, \mathtt{par}\ V_n\ \mathtt{and}\ W_n\ \}$.*
- *If $\vdash \mathtt{par}\ \{S\ ;\ T\}\ \mathtt{and}\ U$, then there exist local types V_i and W_i such that $\vdash \mathtt{par}\ S\ \mathtt{and}\ V_i$ and $\vdash \mathtt{par}\ T\ \mathtt{and}\ W_i$, for $1 \leq i \leq n$, and n-ary killing context $\mathcal{T}\{\ \}$ where $U \longrightarrow \mathcal{T}\{\ V_1\,;W_1, V_2\,;W_2, \ldots, V_n\,;W_n\ \}$.*

The above splitting result is key to solving two further lemmas, the proof of which are provided in a companion paper [13]. Hence, in the interest of focusing on the relevance of MAV to sessions, we provide only statements of the lemmas.

Lemma 2 (Context reduction). *If, for any type V, $\vdash \mathtt{par}\ T\ \mathtt{and}\ V$ yields $\vdash \mathtt{par}\ U\ \mathtt{and}\ V$, then, for any context $\mathcal{C}\{\ \}$, $\vdash \mathcal{C}\{\ T\ \}$ yields $\vdash \mathcal{C}\{\ U\ \}$.*

The following result, shows that rules complimentary to those that appear in Fig. 4 can be eliminated. By a complementary rule, or *co-rule*, we mean a rule where the direction of rewriting is reversed and co-typing is applied to both local types in the rewrite rule. The proof follows from splitting and the context lemma.

Lemma 3 (Co-rule elimination). *The following statements hold.*

- *If $\vdash \mathcal{C}\{\ \mathtt{sync}\ A\ \mathtt{and} \sim B\ \}$, where $A \leq B$, then $\vdash \mathcal{C}\{\ \}$.*
- *If $\vdash \mathcal{C}\{\ \mathtt{sync}\ \{\ T\,;U\ \}\ \mathtt{and}\ \{\ V\,;W\ \}\ \}$,*
  then $\vdash \mathcal{C}\{\ \{\ \mathtt{sync}\ T\ \mathtt{and}\ V\ \}\,;\{\ \mathtt{sync}\ U\ \mathtt{and}\ W\ \}\ \}$.
- *If $\vdash \mathcal{C}\{\ T \,\&\, U\ \}$, then $\vdash \mathcal{C}\{\ T\ \}$.*
- *If $\vdash \mathcal{C}\{\ T \,\&\, U\ \}$, then $\vdash \mathcal{C}\{\ U\ \}$.*
- *If $\vdash \mathcal{C}\{\ \mathtt{sync}\ T\ \mathtt{and}\ \{\ U\ \mathtt{or}\ V\ \}\ \}$,*
  then $\vdash \mathcal{C}\{\ \{\ \mathtt{sync}\ T\ \mathtt{and}\ U\ \}\ \mathtt{or}\ \{\ \mathtt{sync}\ T\ \mathtt{and}\ V\ \}\ \}$.
- *If $\vdash \mathcal{C}\{\ \{\}\ \mathtt{or}\ \{\}\ \}$, then $\vdash \mathcal{C}\{\ \}$.*
- *If $\vdash \mathcal{C}\{\ \{\ P\ ;\ Q\ \}\ \mathtt{or}\ \{\ R\ ;\ S\ \}\ \}$,*
  then $\vdash \mathcal{C}\{\ \{\ P\ \mathtt{or}\ R\ \}\ ;\ \{\ Q\ \mathtt{or}\ S\ \}\ \}$.

Theorem 1, follows from the above result, by induction on the size of the local type in any interaction or co-interaction rule. Thereby we have established the consistency of the system MAV. The following section, demonstrates why consistency of a calculus is more than a theoretical curiosity.

3.1 A Subtyping Relation for Sessions

We use the semantics of local types to define a subtype relation over local types.

Definition 3. *For local types T and U, if $\vdash$ `par` $\sim T$ `and` U, then $T \leq U$, pronounced T is a subtype of U.*

From Theorem 1 we immediately establish that subtyping is consistent in the following sense.

Corollary 1. *Subtyping is a precongruence: a reflexive, transitive relation that holds in any context.*

Consider the running 2PC example. The **Leader** local protocol in Fig 2 is a subtype of the following local protocol **Leader**$'$.

$$\begin{array}{l}\texttt{par}\ \mathit{prepare}(\mathit{Timestamp})\ \texttt{from}\ \mathit{Participant}\\ \texttt{and}\ \mathit{l\_begin}(\mathit{Payload})\ \texttt{from}\ \mathit{Client}\ ;\\ \texttt{par}\ \sim\mathit{p\_commit}(\mathit{Timestamp})\ \texttt{to}\ \mathit{Participant}\\ \texttt{and}\ \sim\mathit{c\_commit}(\mathit{Timestamp})\ \texttt{to}\ \mathit{Client}\end{array}$$

The difference between the local types for **Leader** and **Leader**$'$ is that **Leader**$'$ waits for the *l_begin* and *prepare* messages in parallel. Thus **Leader**; can potentially consume the *prepare* message before consuming the *l_begin* message, which is not possible in **Leader**.

Subsorting. The use of subsorts allows us to recover a classic property of subtyping for channel types [20]: send types are contravariant and receive types are covariant. Immediately, from the atomic interaction rules we obtain, that if $S0 \leq S1$, then both $\sim c(S1)\ \texttt{to}\ P \leq \sim c(S0)\ \texttt{to}\ P$ and $c(S0)\ \texttt{from}\ P \leq c(S1)\ \texttt{from}\ P$ hold. Notice that the complementation operator prefixing the send event induces the expected contravariance. For example, in 2PC the sender may send a natural number timestamp, when an integer timestamp was expected, assuming that $nat \leq int$ is in the subsorting system. This agrees with related work [20] on subtyping with respect to I/O types for channels.

Subtyping for choice. The subtype system derived from the extended calculus, reflects existing work on session subtyping involving choice [6]. Consider the extended two phase commit example, with local types **Leader**$''$, **Participant**$'$, and **Client**$'$ in Fig. 5. Note that `par`, `sync` and semi-colon are assumed to have a higher precedence than & and `or`.

The example local types **Leader**$''$ and **Client**$'$ involve an external choice. The local type **Leader**$''$ has a choice that allows an abort message to be received from the participant, at which point the client must be notified about the abort. The client has the choice of receiving the commit message or receiving the abort message.

The local type **Leader**$''$ is a super-type of the local type **Leader**, since **Leader**$''$ can always (internally) choose the left branch of the choice in the protocol. Similarly, the local type **Client**$'$ is a super-type of **Client**.

Client′ :

```
par ∼p_begin(Payload) to Participant
and ∼l_begin(Payload) to Leader ;
{ commit(Timestamp) from Leader
  or
  c_abort(Error) from Leader }
```

Participant′ :

```
p_begin(Payload) from Client ;
{ ∼prepare(Timestamp) to Leader ;
  p_commit(Timestamp) from Leader
  &
  ∼p_abort(Error) to Leader }
```

Leader″ :

```
l_begin(Payload) from Client;
{ prepare(Timestamp) from Participant ;
  par ∼p_commit(Timestamp) to Participant
  and ∼c_commit(Timestamp) to Client
  or
  p_abort(Error) from Participant ;
  ∼c_abort(Error) to Client }
```

Fig. 5. Example of roles in a commit protocol with the choice to abort.

Now consider the local type **Participant′**. In contrast to **Leader″** and **Client′**, the local type **Participant′** is *not* a supertype of **Participant**. This contrast is due to the presence of external choice rather than internal choice. An external choice is used since we cannot determine at compile-time whether the participant will commit or abort; hence both branches must be analysed.

3.2 Multiparty Compatibility

The semantics of local types in Fig. 4 can also be used to determine whether a session can successfully close, without hanging sends or receives. The following definition is the essence of the idea of *multiparty compatibility* in [5].

Definition 4 (Multiparty compatibility). *If T_1, T_2, $\ldots T_n$ are local types such that* **par** T_1 **and** T_2 $\ldots$ **and** T_n *is provable, then T_1, T_2, $\ldots T_n$ are said to be multiparty compatible.*

The following example, due to Tiu [21], emphasises that we could not express multiparty compatibility using natural deduction.

```
Role P : ∼begin(Data) to Q ;
         {
           par ∼fun(Control) to Q
           and done(Data) from Q
         }

Role Q : {
           par begin(Data) from P
           and fun(Control) from P
         } ;
         ∼done(Data) to P
```

Notice that the causal dependency forced by role Q between receiving a message on channel *fun* and sending a message on channel *done*, induces a dependency at role P; specifically the send on *fun* must happen before the receive on *done*. In

the proof of multiparty compatibility, this dependency is imposed by applying a rule, within sequential composition structure, within a par structure. Application of rules in a context alternating between structures is known as *deep inference*. Natural deduction, traditionally used to express type systems, cannot express the scenario above.

The projection of any global type onto its local types for each participant and channel is multiparty compatible.

Lemma 4. *For any G with set of roles and channels I, the multiset of local types $(G \upharpoonright_i)_{i \in I}$ is multiparty compatible.*

Proof. The proof is by induction on the structure of G. The only base case is $c(S)$ `from` P `to` Q, for which the following rewrites hold using *switch* and *atomic interaction*.

$$\begin{aligned}
&\texttt{par}\ G \upharpoonright_P\ \texttt{and}\ G \upharpoonright_Q\ \texttt{and}\ G \upharpoonright_c = \\
&\texttt{par} \sim c(S)\ \texttt{to}\ Q\ \texttt{and}\ c(S)\ \texttt{from}\ P\ \texttt{and}\ \{\ \texttt{sync}\ c(S)\ \texttt{to}\ Q\ \texttt{and} \sim c(S)\ \texttt{from}\ P\ \} \\
&\qquad \longrightarrow \texttt{sync}\ \{\ \texttt{par} \sim c(S)\ \texttt{to}\ Q\ \texttt{and}\ c(S)\ \texttt{to}\ Q\ \}\ \texttt{and} \\
&\qquad\quad \{\ \texttt{par} \sim c(S)\ \texttt{from}\ P\ \texttt{and}\ c(S)\ \texttt{from}\ P\ \} \longrightarrow \{\}
\end{aligned}$$

Hence the local types $G \upharpoonright_P$, $G \upharpoonright_Q$, $G \upharpoonright_c$ are multiparty compatible.

Consider the case of sequential composition. Assume that G_0 and G_1 are multiparty compatible and consider $G_0 ; G_1$. Hence for $i_1, i_2, \ldots \in I$. By repeated application of the seq rule and the induction hypotheses.

$$\begin{aligned}
&\texttt{par}\ \{\ (G_0 ; G_1) \upharpoonright_{i_1}\ \}\ \texttt{and}\ \{\ (G_0 ; G_1) \upharpoonright_{i_2}\ \}\ \texttt{and} \ldots \\
&\quad = \texttt{par}\ \{\ G_0 \upharpoonright_{i_1} ; G_1 \upharpoonright_{i_1}\ \}\ \texttt{and}\ \{\ G_0 \upharpoonright_{i_2} ; G_1 \upharpoonright_{i_2}\ \}\ \texttt{and} \ldots \\
&\quad \longrightarrow \texttt{par}\ G_0 \upharpoonright_{i_1}\ \texttt{and}\ G_0 \upharpoonright_{i_2}\ \texttt{and} \ldots\ ;\ \texttt{par}\ G_1 \upharpoonright_{i_1}\ \texttt{and}\ G_1 \upharpoonright_{i_2}\ \texttt{and} \ldots \\
&\quad \longrightarrow \{\}
\end{aligned}$$

Hence $G_0 ; G_1$ is multiparty compatible.

Consider the case of choice. Assume that G_0 and G_1 are multiparty compatible and consider `choice at` P G_0 `or` G_1. Hence for $i_1, i_2, \ldots \in I \setminus \{P\}$. By repeated application of the external choice, left choice, right choice and the induction hypotheses.

$$\begin{aligned}
&\texttt{par}\ (\texttt{choice at}\ P\ G_0\ \texttt{or}\ G_1) \upharpoonright_P \texttt{and}\ (\texttt{choice at}\ P\ G_0\ \texttt{or}\ G_1) \upharpoonright_{i_1}\ \texttt{and} \ldots \\
&\quad = \texttt{par}\ \{\ G_0 \upharpoonright_P \,\&\, G_1 \upharpoonright_P\}\ \texttt{and}\ \{\ G_0 \upharpoonright_{i_1}\ \texttt{or}\ G_1 \upharpoonright_{i_1}\}\ \texttt{and} \ldots \\
&\quad \longrightarrow \texttt{par}\ G_0 \upharpoonright_P\ \texttt{and}\ \{\ G_0 \upharpoonright_{i_1}\ \texttt{or}\ G_1 \upharpoonright_{i_1}\ \}\ \texttt{and} \ldots \\
&\qquad \,\&\, \texttt{par}\ G_1 \upharpoonright_P\ \texttt{and}\ \{\ G_0 \upharpoonright_{i_1}\ \texttt{or}\ G_1 \upharpoonright_{i_1}\ \}\ \texttt{and} \ldots \\
&\quad \longrightarrow \texttt{par}\ G_0 \upharpoonright_P\ \texttt{and}\ G_0 \upharpoonright_{i_1}\ \texttt{and} \ldots\ \,\&\,\ \texttt{par}\ G_1 \upharpoonright_P\ \texttt{and}\ G_1 \upharpoonright_{i_1}\ \texttt{and} \ldots \\
&\quad \longrightarrow \{\}\ \,\&\,\ \{\} \longrightarrow \{\}
\end{aligned}$$

Hence `choice at` P G_0 `or` G_1 is multiparty compatible.

The inductive case for parallel composition is similar. Hence the result follows by induction on the structure of G. □

A consequence of the above lemma is that the family of all projections from the global type in Fig. 1 is multiparty compatible.

Subtyping allows local protocols to be weakened while retaining multiparty compatibility. The following lemma is an immediate consequence of Corollary 1.

Lemma 5. *If T_1, T_2, ..., T_n are multiparty compatible, and $T_i \leq U_i$, for $1 \leq i \leq n$, then U_1, U_2, ... U_n are multiparty compatible.*

We now introduce the notion of coherence, which defines families of local protocols and channels, that respect a global type.

Definition 5. *A multiset of local types $(T_i)_{i \in I}$, where I is a set of roles and channels, is coherent (with respect to G) if there exists a global type G such that for all $i \in I$, $G \upharpoonright_i \leq T_i$, where $\leq$ is the subtyping relation.*

The protocol based on Tiu's counter example is *coherent* with respect to the following global type.

$$\begin{array}{l} \mathit{begin}\,(\mathit{Data})\ \texttt{from}\ P\ \texttt{to}\ Q\ ; \\ \mathit{fun}\,(\mathit{Function})\ \texttt{from}\ P\ \texttt{to}\ Q\ ; \\ \mathit{done}\,(\mathit{Data})\ \texttt{from}\ Q\ \texttt{to}\ P \end{array}$$

Notice that, if type equality rather than subtyping is used in the definition of coherence, then Tiu's counter example is not coherent. Thus subtyping relaxes the corresponding definition in [12].

The proof of the following proposition follows from Lemma 4 and Lemma 5.

Proposition 1. *Any coherent protocol is multiparty compatible.*

Proof. Assume that $(T_i)_{i \in I}$ is coherent. Hence there exists G such that for all $i \in I$, $G \upharpoonright_i \leq T_i$. By Lemma 4, the multiset of of protocols $(G \upharpoonright_i)_{i \in I}$ is multiparty compatible. Furthermore, by Lemma 5, since for all $i \in I$, $G \upharpoonright_i \leq T_i$, $(T_i)_{i \in I}$ is also multiparty compatible. □

The converse of Proposition 1 is more difficult, since given only the local protocols, we must construct, or *synthesise*, a global protocol given only the local types. In general synthesis of a global protocol is not possible [5], hence an open question is the following: under what conditions is a multiparty compatible family of local protocols is coherent.

The protocols **Client**, **Leader**$'$, **Participant** are coherent, since the definition of coherence permits subtyping. However they are also coherent with respect to the global protocol below.

$$\begin{array}{l} \texttt{par}\ \{\ \mathit{p\_begin}\,(\mathit{Payload})\ \texttt{from}\ \mathit{Client}\ \texttt{to}\ \mathit{Participant}\ ; \\ \qquad\ \ \mathit{prepare}\,(\mathit{Timestamp})\ \texttt{from}\ \mathit{Participant}\ \texttt{to}\ \mathit{Leader}\ \} \\ \texttt{and}\ \mathit{l\_begin}\,(\mathit{Payload})\ \texttt{from}\ \mathit{Client}\ \texttt{to}\ \mathit{Leader}\ ; \\ \texttt{par}\ \mathit{c\_commit}\,(\mathit{Timestamp})\ \texttt{from}\ \mathit{Leader}\ \texttt{to}\ \mathit{Client} \\ \texttt{and}\ \mathit{p\_commit}\,(\mathit{Timestamp})\ \texttt{from}\ \mathit{Leader}\ \texttt{to}\ \mathit{Participant} \end{array}$$

The global type above is more general than the global type in Fig. 1, with respect to the subtyping relation over global types defined below.

Definition 6. *For global types G_0 and G_1 with set of participants and channels I, if for all $i \in I$, $G_0 \upharpoonright_i \leq G_1 \upharpoonright_i$, then $G_0 \leq G_1$.*

Following [18], we say that the most general coherent global protocol, with respect to the above subtyping relation, with respect to the multiset of protocols, is the *principal global type* for that multiset of local protocols. Hence the above global type is the principal global type for **Client**, **Leader**$'$, **Participant**.

For the local protocols **Client**$'$, **Leader**$''$ and **Participant**$'$ from Fig. 5, the principal global type is as follows.

```
par p_begin(Payload) from Client to Participant
and l_begin(Payload) from Client to Leader ;
choice at Participant
   { prepare(Timestamp) from Participant to Leader ;
     par c_commit(Timestamp) from Leader to Client
     and p_commit(Timestamp) from Leader to Participant }
or { p_abort(Error) from Participant to Leader ;
     c_abort(Error) from Leader to Client }
```

Notice that, since subtyping for external choice and internal choice are dual to each other, there is no subtype relationship between the global type above and the global type in Fig. 1. Subtyping of global protocols should preserve coherence.

Proposition 2. *If $(T_i)_{i\in I}$ is coherent with respect to G_1 and $G_0 \leq G_1$, then $(T_i)_{i\in I}$ is coherent with respect to G_0.*

The proof follows immediately from the definitions.

Asynchronous sessions. Assume that we use asynchronous communication channels, where a send message event happens before the corresponding receive message event, and furthermore the order messages are received are not related to the order in which messages are sent. The projection to asynchronous channels is the same as the projection for synchronous channels except the following case.

$$d(S)\ \texttt{from}\ P\ \texttt{to}\ Q\upharpoonright_c = \begin{cases} \texttt{sync}\ c(S)\ \texttt{to}\ Q\ ;\ {\sim}c(S)\ \texttt{from}\ P & \text{if } c = d \\ \{\} & \text{otherwise} \end{cases}$$

We can establish that, for protocols that use synchronous channels and can successfully complete, the protocols can successfully complete using asynchronous channels in place of synchronous channels. This observation follows immediately from Lemma 4, Corollary 1, and the observation that $\texttt{sync}\ T\ \texttt{and}\ U \leq T\ ;\ U$.

Corollary 2. *If G is a global protocol, where I is the set of roles and channels appearing in G, then using asynchronous channels, $(G\upharpoonright_i)_{i\in I}$ is multiparty compatible.*

However, the converse does not hold. There exist sessions that may successfully complete using asynchronous channels, that can never complete successfully using synchronous channels. The methodology in this work can also be adapted to evaluate the termination potential of protocols where channels are queues of length up to two, for which stronger guarantees on the order in which messages are receive on channels are guaranteed [12].

3.3 Complexity and a path to implementation

Deciding provability in the calculus MAV is a PSPACE-complete problem. This complexity bound can be established directly from existing results. In particular, since the calculus extends MALL which is PSPACE-hard [17], there is a trivial polynomial reduction from MALL to MAV, i.e. the direct embedding of propositions, such that a proposition is provable in MALL if and only if its embedding is provable in MAV. Furthermore, by the argument in [17], the length of each independent branch in a proof is polynomial.

Proposition 3. *The problem of deciding whether a local type in MAV is provable is PSPACE-complete.*

A consequence of the above complexity result is provability of local types in MAV can be reduced to QBF, the canonical PSPACE-complete problem, for which sufficiently efficient solvers exist. Thereby properties of sessions, such as multiparty compatibility, can be verified for protocols of a realistic size.

4 Related Work and Conclusion

Correspondences between session types and variants or extensions of linear logic have been studied in related work. The study of the "proofs as processes" interpretation of *intuitionistic linear logic* using the linear λ-calculus was initiated by Abramsky [1], and has been adapted explicitly to asynchronous sessions [7]. An alternative approach due to Caires and Pfenning [3] assigns propositions in intuitionistic linear logic to channel names, where the proposition represents the session performed on the named channel. Wadler [22] brings light to the linear λ-calculus and channel approaches, by proving that a variant of the linear λ-calculus can be translated faithfully into a process calculus and type system where propositions in (classical) linear logic are assigned to channel names. In both the intuitionistic [3] and classical [22] interpretations, intuitively, the tensor product is interpreted asymmetrically as follows: "$A \otimes B$ is the type of a channel that outputs an A *and then* behaves as B." Both approaches argue that the symmetry of the tensor product can be recovered using an isomorphism induced by a process that flips the order of actions.

We make a different choice. We interpret sequentiality using an explicit non-commutative operator. This relieves the commutative operators so that they can be used to model symmetric features such as parallel composition. For the calculus in this work, the results presented, notably the consistency of subtyping (Corollary 1) and the multiparty compatibility of coherent protocols (Proposition 1), follow directly from the proof theoretic result in Theorem 1. Thus, by adopting the calculus of structures to express the semantics of local types, we reduce several problems in session types to a generalised cut elimination result.

Acknowledgements. This work is supported by a grant of the Romanian National Authority for Scientific Research, project number PN-II-ID-PCE-2011-3-0919.

References

1. Samson Abramsky. Computational interpretations of linear logic. *Theoretical computer science*, 111(1):3–57, 1993.
2. Véronique Benzaken, Giuseppe Castagna, and Alain Frisch. CDuce: an XML-centric general-purpose language. *ACM SIGPLAN Notices*, 38(9):51–63, 2003.
3. Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *CONCUR 2010*, pages 222–236. Springer, 2010.
4. Kaustuv Chaudhuri, Nicolas Guenot, and Lutz Straßburger. The focused calculus of structures. In *EACSL*, volume 12, pages 159–173, 2011.
5. 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.
6. Simon Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, 2005.
7. Simon J Gay and Vasco T Vasconcelos. Linear type theory for asynchronous session types. *Journal of Functional Programming*, 20(1):19, 2010.
8. Jay L. Gischer. The equational theory of pomsets. *Theoretical Computer Science*, 61(2-3):199–224, 1988.
9. Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Compututational Logic*, 8, 2007.
10. Kohei Honda. Types for dyadic interaction. In *CONCUR'93*, pages 509–523. Springer, 1993.
11. Kohei Honda, Aybek Mukhamedov, Gary Brown, Tzu-Chun Chen, and Nobuko Yoshida. Scribbling interactions with a formal foundation. In *Distributed Computing and Internet Technology*, pages 55–75. Springer, 2011.
12. Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *ACM SIGPLAN Notices*, 43(1):273–284, 2008.
13. Ross Horne. The consistency and complexity of multiplicative additive system virtual. *Scientific Annals of Computer Science*. to appear.
14. Raymond Hu, Rumyana Neykova, Nobuko Yoshida, Romain Demangeon, and Kohei Honda. Practical interruptible conversations. In *RV*, pages 130–148, 2013.
15. Raymond Hu, Nobuko Yoshida, and Kohei Honda. Session-based distributed programming in java. In *ECOOP*, pages 516–541. Springer, 2008.
16. Ozan Kahramanogullari. Maude as a platform for designing and implementing deep inference systems. *ENTCS*, 219:35–50, 2008.
17. Patrick Lincoln and other. Decision problems for propositional linear logic. *Ann. Pure Appl. Logic*, 56(1):239–311, 1992.
18. 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.
19. Nicholas Ng, Nobuko Yoshida, and Kohei Honda. Multiparty session C: Safe parallel programming with message optimisation. In *Objects, Models, Components, Patterns*, pages 202–218. Springer, 2012.
20. Benjamin Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. In *LICS'93*, pages 376–385. IEEE, 1993.
21. Alwen Tiu. A system of interaction and structure II: The need for deep inference. *Logical Methods in Computer Science*, 2(2), 2006.
22. Philip Wadler. Propositions as sessions. *Journal of Functional Programming*, 24(2-3):384–418, 2014.