Aspectual Session Types

Nicolas Tabareau Mario Südholt
ASCOLA Team
Mines Nantes & Inria & LINA, Nantes, France
nicolas.tabareau@inria.fr
mario.sudholt@mines-nantes.fr

Éric Tanter *
PLEIAD Laboratory
Computer Science Department (DCC)
University of Chile
etanter@dcc.uchile.cl

Abstract

Multiparty session types allow the definition of distributed processes with strong communication safety properties. A global type is a choreographic specification of the interactions between peers, which is then projected locally in each peer. Well-typed processes behave accordingly to the global protocol specification. Multiparty session types are however monolithic entities that are not amenable to modular extensions. Also, session types impose conservative requirements to prevent any race condition, which prohibit the uniform application of extensions at different points in a protocol. In this paper, we describe a means to support modular extensions with *aspectual session types*, a static pointcut/advice mechanism at the session type level. To support the modular definition of crosscutting concerns, we augment the expressivity of session types to allow harmless race conditions. We formally prove that well-formed aspectual session types entail communication safety. As a result, aspectual session types make multiparty session types more flexible, modular, and extensible.

Categories and Subject Descriptors D.3.3 [*Programming Languages*]: Language Constructs and Features

Keywords session types, aspect-oriented programming

1. Introduction

Interaction protocols for the orchestrations of services, such as BPEL, are nowadays a common means to define Cloud-based applications. Most frequently, finite-state protocols are used to define interactions that are, however, not expressive enough to define many interaction patterns. More general approaches, such as communicating finite-state machines (CFSMs) are too expressive to support static correctness guarantees.

Session types [12] have been developed as an expressive means to define interaction protocols that provide correctness guarantees such as deadlock freedom, the absence of stuck messages, and the correctness of message reception by concurrent threads. In recent years, the expressivity of session types has been extended through the integration of multiple interaction parties, multiple roles, and the generalization of the underlying type system using a new subclass of CFSMs [7, 8, 13].

The main characteristic of multiparty session types consists in the provision of global types that define the overall interactions of a system and an automatic projection mechanism from a global type into local types. Local types define the interaction behavior of local processes, all of which, executed together, realize exactly the interaction behavior defined by the global type without any local process requiring any non-local interaction information.

The most recent version of session types developed by Deniélou and Yoshida [8] introduces more general parallel and choice flows that are expressed respectively using fork/joins and choice/merge nodes on the global type level and in a new class of graphs, the so-called multiparty session automata that constitute a subclass of communicating finite state machines (CFSMs). Global types with these generalized flows, the corresponding automata, local projections and processes have been shown to meet the three main correctness properties introduced above.

Despite these advances, session types suffer from a number of restrictions that hamper their adoption. First, session types do not support modular extensions. This is problematic because protocol-like service compositions are well-known to benefit from extension mechanism that allow functionalities to be added in a modular fashion. Otherwise the underlying protocols often have to be extensively rewritten, thus complexifying the protocol structure, a tedious and error-prone task. Aspect-oriented systems have been proposed for this purpose, notably to manipulate service orchestrations modularly, for example, using AO4BPEL [6] and in particular for the control of the QoS properties of orchestrations [2]. However, no aspect system exists that enables the modification of session types while maintaining their strong correctness guarantees.

Second, in order to guarantee strong properties such as deadlock freedom, session types generally have to impose restrictions on the interactions they support. One important restriction consists in the so-called *linearity* condition, which means that a participant should never be faced with two concurrent receptions where messages can have the same label. This is to forbid potential race conditions, which can lead to deadlocks. Thus, identical modifications that introduce the same new behavior in different threads cannot be expressed using session types because doing so directly breaks linearity.

We address these issues by introducing *aspectual session types* that enable the addition of new functionalities to multiparty generalized session types [8] in a modular way, supporting the introduction of uniform behavior in multiple places in session types. Intuitively, the latter is achieved by allowing race conditions between advice bodies, but by ensuring the preservation of the correctness conditions of session types by executing them in mutual exclusion.

* É. Tanter is partially funded by FONDECYT project 1110051.
This work is supported by the INRIA Associated Team REAL.

MODULARITY '14, April 22–26, 2014, Lugano, Switzerland.
Copyright © 2014 ACM 978-1-4503-2772-5/14/04…$15.00.
http://dx.doi.org/10.1145/2577080.2577085

Concretely, we provide the following contributions:

- We formally define *aspectual session types* that allow messages in session types to be matched and to introduce complex behavior in addition or in place of the matched messages.
- We explain a safe execution strategy for the weaving of advice at multiple places in session types and thus support a particular kind of race conditions, which we call *harmless race conditions*.
- We formally define the weaving of such aspects into global and local types by extending the formal framework of Deniélou and Yoshida [8].
- We provide a proof that the resulting types preserve all properties of session types despite the presence of harmless race conditions: absence of deadlocks, orphan messages (messages that will never be received) and the correct reception of messages in concurrent contexts (in the sense that messages of one thread can not be mistakenly received by another thread).

Section 2 briefly illustrates aspectual session types through examples. Section 3 provides the necessary background on generalized multiparty session types based on [8]. Section 4 introduces aspectual session types. Section 5 describes different properties of aspectual session types necessary for weaving to preserve the good properties of session types. Section 6 proves that harmless race conditions introduced by aspect weaving are indeed harmless: all properties are preserved. Section 7 describes an application of aspectual session types to data parallel programming. Section 8 discusses related work and Section 9 concludes.

2. Overview of Aspectual Session Types

We now provide an example-driven overview of the approach. Figure 1 presents the graphical representation of session types and aspects (their formal definitions are given in Appendix A). The exhaustive description of the formal syntax and semantics is developed in later sections.

A simple trade session. Figure 1a shows the graphical representation of a simple global type for a trade-like interaction, simplified and adapted from [8]. A seller S initially sends an item that contains its price to a broker B. The broker knows about an interested client C beforehand (no negotiation is taking place here) and then informs, in parallel, S whether the item is acceptably priced for C and the latter if he can purchase the item from the former.

Supporting modular extensions. Consider the introduction of a price negotiation between the broker and the client. This change would require the global interaction type to be rewritten[1]. In addition, the changes on the global types would cascade as changes to the corresponding local types, which in turn means that local processes have to be modified accordingly.

Aspectual session types enable modular extensions to existing session types. For instance, we can use the negotiation aspect shown in Figure 1b: this aspect matches the interaction `S→B:Item` above the large delimited arrow and adds a negotiation loop after the interaction. The matched message is dynamically bound to the keyword proceed. Weaving of aspectual session types can be defined to ensure that modular extensions preserve the properties of well-formed session types.

The logging aspect shown in Figure 1c shows how multiple interactions, here B→S or B→C, may be matched through quantification, allowing the modular definition of a crosscutting concern.

[1] The original trade example in [8] includes the negotiation phase built in the interaction.

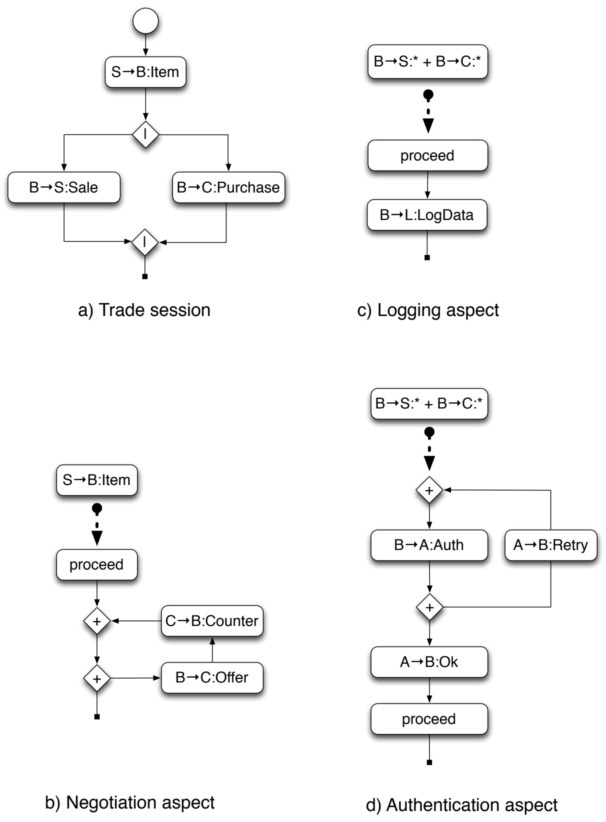

Figure 1: Graphical representation of the trade example, with three aspects

Allowing uniform behavior in parallel threads Once quantification is introduced, uniform behavior can be applied in parallel threads, as in the logging example. A more interesting example is an authentication aspect shown in Figure 1d: it introduces an interaction from the broker to an authentication server B→A before each of the two messages B→S and B→C (represented by proceed). If applied to the base trading session (Figure 1a), it results in both branches after the fork to send two messages from B: respectively to A followed by one to S and to A followed by C.

Note that in these cases, the session types that result from weaving do not respect the traditional linearity condition of session types. Our approach however ensures that such weaving does produce correct session types. Aspectual session types therefore accommodate a strict superset of protocols compared to Daniélou's and Yoshida's session types [8].

3. Generalized Multiparty Session Types

This section briefly reviews the formal treatment of generalized multiparty session types based on the article of Deniélou and Yoshida [8]. We clearly identify the few points in which we depart from their definitions.

3.1 Global Types

The syntax of global session types is given in Figure 2. A global type $\mathbf{G}$ is a set of transitions $\overline{G}$ (the overbar denotes a possibly empty sequence), together with an *initial state* $\mathbf{x}$. *State variables* $\mathbf{x}$ in $\overline{G}$ are the different states of the interaction. Transitions between states include coordination and communication among participants $\mathtt{p} \in \mathcal{P}$. The set of participants is fixed in a given interaction.

$\mathbf{G}$	::=	$\mathsf{def}\ \overline{G}\ \mathsf{in}\ \mathbf{x}$	Global Type
U	::=	$\langle \mathbf{G} \rangle \mid \mathit{bool} \mid \mathit{nat} \mid \ldots$	Sort
G	::=		Transition
	$\mid$	$\mathbf{x} = M; \mathbf{x}'$	Messages
	$\mid$	$\mathbf{x} \mid \mathbf{x}' = \mathbf{x}''$	Join
	$\mid$	$\mathbf{x} = \mathbf{x}' \mid \mathbf{x}''$	Fork
	$\mid$	$\mathbf{x} + \mathbf{x}' = \mathbf{x}'$	Merge
	$\mid$	$\mathbf{x} = \mathbf{x}' + \mathbf{x}''$	Choice
	$\mid$	$\mathbf{x} = \mathsf{end}$	End
M	::=	$\mathsf{p} \to \mathsf{p}' : l\langle U \rangle$	Labeled Message

Figure 2: Syntax of Global Session Types (adapted from [8])

$\mathbf{T}$	::=	$\mathsf{def}\ \overline{T}\ \mathsf{in}\ \mathbf{x}$	Local Type
T	::=		Local Transition
	$\mid$	$\mathbf{x} = M; \mathbf{x}'$	Message
	$\mid$	$\mathbf{x} = \mathbf{x}'$	Indirection
	$\mid$	$\mathbf{x} = \mathbf{x}' \oplus \mathbf{x}''$	Internal Choice
	$\mid$	$\mathbf{x} = \mathbf{x}' \& \mathbf{x}''$	External Choice
	$\mid$	$\mathbf{x} + \mathbf{x}' = \mathbf{x}'$	Merge
	$\mid$	$\mathbf{x} = \mathbf{x}' \mid \mathbf{x}''$	Fork
	$\mid$	$\mathbf{x} \mid \mathbf{x}' = \mathbf{x}''$	Join
	$\mid$	$\mathbf{x} = \mathsf{end}$	End
M	::=	$!\langle \mathsf{p}, l\langle U \rangle \rangle$	Send
	$\mid$	$?\langle \mathsf{p}, l\langle U \rangle \rangle$	Receive
	$\mid$	0	No Message

Figure 3: Syntax of Local Session Types (adapted from [8])

Compared to [8], we introduce a dedicated syntactic category M to describe messages, which we extend later. A message transition $\mathbf{x} = M; \mathbf{x}'$, with $M = \mathsf{p} \to \mathsf{p}' : l\langle U \rangle$, specifies that in state $\mathbf{x}$, the sender p can go to the continuation $\mathbf{x}'$ by sending a message labeled l (taken from a set $\mathbb{L}$) with payload argument type U. p' can go from $\mathbf{x}$ to $\mathbf{x}'$ by receiving the message, while other participants can freely go to state $\mathbf{x}'$. The choice transition $\mathbf{x} = \mathbf{x}' + \mathbf{x}''$ represents a choice made by exactly one participant to go to either $\mathbf{x}'$ or $\mathbf{x}''$. $\mathbf{x} = \mathbf{x}' \mid \mathbf{x}''$ represents forking the interactions in parallel threads. Forks are collected by joins, and choices are closed by merges. $\mathbf{x} = \mathsf{end}$ marks the end of the session.

3.2 Local Types

Figure 3 gives the syntax of local session types. Local types directly correspond to global types except for a couple of refinements. First, a message globally described as $\mathsf{p} \to \mathsf{p}' : l\langle U \rangle$ is reflected locally by a send action $!\langle \mathsf{p}', l\langle U \rangle \rangle$ on p, and by a receive action $?\langle \mathsf{p}, l\langle U \rangle \rangle$ on p'. Second, there are two variants of the global choice: the internal choice $\oplus$ is used on the participant that drives the choice, while the external choice + is used on other participants, which are passive observers of the choice made by another participant.

Following our introduction of the category M for messages, we introduce 0 to denote the absence of action. We extend the congruence relation on local types $\equiv$ of [8] to account for the introduction of 0 for messages:

$$\mathbf{x} = 0; \mathbf{x}' \quad \equiv \quad \mathbf{x} = \mathbf{x}'$$

The projection from global types to local types $\upharpoonright$ is direct and given in Figure 4. As hinted previously, the local projection of a message is either a send, a receive or a null action. Fork, join, choice and merge transitions are projected as corresponding local

$\mathsf{def}\ \overline{G}\ \mathsf{in}\ \mathbf{x} \upharpoonright \mathsf{p}$	=	$\mathsf{def}\ \overline{G} \upharpoonright_{\overline{G}} \mathsf{p}\ \mathsf{in}\ \mathbf{x}$
$\mathbf{x} = M; \mathbf{x}' \upharpoonright_{\overline{G}} \mathsf{p}$	=	$\mathbf{x} = M \upharpoonright \mathsf{p}; \mathbf{x}'$
$\mathsf{p} \to \mathsf{p}' : l\langle U \rangle \upharpoonright \mathsf{p}$	=	$!\langle \mathsf{p}', l\langle U \rangle \rangle$
$\mathsf{p} \to \mathsf{p}' : l\langle U \rangle \upharpoonright \mathsf{p}'$	=	$?\langle \mathsf{p}, l\langle U \rangle \rangle$
$\mathsf{p} \to \mathsf{p}' : l\langle U \rangle \upharpoonright \mathsf{p}''$	=	$0\ (\mathsf{p}'' \notin \{\mathsf{p}, \mathsf{p}'\})$
$\mathbf{x} \mid \mathbf{x}' = \mathbf{x}'' \upharpoonright_{\overline{G}} \mathsf{p}$	=	$\mathbf{x} \mid \mathbf{x}' = \mathbf{x}''$
$\mathbf{x} = \mathbf{x}' \mid \mathbf{x}'' \upharpoonright_{\overline{G}} \mathsf{p}$	=	$\mathbf{x} = \mathbf{x}' \mid \mathbf{x}''$
$\mathbf{x} = \mathbf{x}' + \mathbf{x}'' \upharpoonright_{\overline{G}} \mathsf{p}$	=	$\mathbf{x} = \mathbf{x}' \oplus \mathbf{x}''$ (if $\mathsf{p} = \mathit{ASend}(\overline{G})(\mathbf{x})$)
$\mathbf{x} = \mathbf{x}' + \mathbf{x}'' \upharpoonright_{\overline{G}} \mathsf{p}$	=	$\mathbf{x} = \mathbf{x}' \& \mathbf{x}''$ (otherwise)
$\mathbf{x} + \mathbf{x}' = \mathbf{x}'' \upharpoonright_{\overline{G}} \mathsf{p}$	=	$\mathbf{x} + \mathbf{x}' = \mathbf{x}''$
$\mathbf{x} = \mathsf{end} \upharpoonright_{\overline{G}} \mathsf{p}$	=	$\mathbf{x} = \mathsf{end}$

Figure 4: Projection Algorithm (adapted from [8])

operations. The only exception is the choice transition which is projected to either an internal or an external choice. The decision is based on whether or not the considered participant is the *active sender* at state $\mathbf{x}$. The computation of the active sender at state $\mathbf{x}$ where $\mathbf{x} = \mathbf{x}' + \mathbf{x}'' \in \overline{G}$ is expressed by an auxiliary function $\mathit{ASend}(\overline{G})(\mathbf{x})$, formally defined in [9, Figure 17], the companion appendix of [8]. This function asserts that there is a unique sender in each branch of a choice and identifies such sender. On the active sender, the choice is projected to an internal choice, while on other participants it is projected to an external choice.

3.3 Well-formedness

Deniélou and Yoshida provide an interpretation of local types as Communicating Finite State Machines (CFSM) that defines a formal semantics for global types. More precisely, they establish a correspondence between the local projections of a *well-formed global type* and a new class of CFSM named Multiparty Session Automata (MSA) that satisfy safety properties (free of deadlocks, orphan messages, and reception errors), as well as progress and liveness.

Deniélou and Yoshida then establish that all the good properties enjoyed by the MSAs generated from a well-formed global type $\mathbf{G}$ also hold in processes typed by the same $\mathbf{G}$. Well-formedness of global types is therefore the key from which all interesting properties are derived. Well-formedness of a global type is expressed in terms of three conditions: sanity, local choice and linearity.

Sanity. To prevent syntactic confusion about which continuations to follow at any given point, a global type $\mathsf{def}\ \overline{G}\ \mathsf{in}\ \mathbf{x}_0$ must satisfy a number of conditions: a) every state variable $\mathbf{x}$ except $\mathbf{x}_0$ appear exactly once on the left- and right-hand side of all transitions in $\overline{G}$; b) $\mathbf{x}_0$ appears exactly once, on the left-hand side; c) end appears at most once; d) transitions in $\overline{G}$ define a connected graph where threads are always collected by joins. Note that this last condition, called *thread correctness*, is less trivial to check. Deniélou and Yoshida present a polynomial verification algorithm based on Petri nets. Thread correctness expresses connectivity, the ability to reach end (liveness), and the fact that join transitions always correspond to concurrent threads.

Local choice. Introducing choices in an interaction must not induce confusion among participants. First, a choice must be local to a participant, meaning only one participant must proactively decide and act according to its decision. This is verified by the active sender computation *ASend*, discussed above. In addition, the choice must be propagated to the other participants, a condition called *choice awareness*. This is expressed with another function, *Rcv*, which ensures that each other participant is either oblivious to the choice (that is, it does not expect any message before the

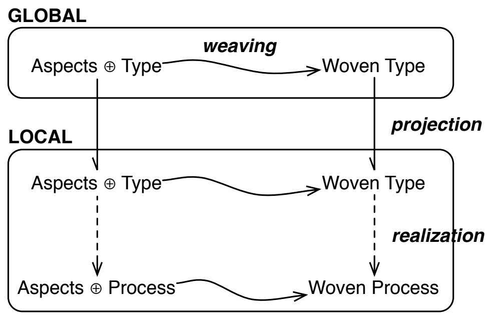

Figure 5: Aspectual Session Types: weaving, projection and realization

choice is merged), or expects messages with different labels in each branch. The definition of *Rcv* can be found in [8, page 7].

Linearity. To avoid race conditions in processes, participants should never be faced with two concurrent receptions where messages can have the same label. This linearity condition is enforced with another auxiliary function (not included here). Intuitively, linearity is to fork what choice awareness is to choice.

Well-formedness of a global type is defined as follows:

Definition 1 (Well-formedness). *A global type* $\mathbf{G}$ *is* well-formed *if it satisfies the* sanity, local choice *and* linearity *conditions.*

4. Aspectual Session Types

Programmers define global session types and global aspects. We define weaving at the global level. The global woven type is interesting for checking properties of the woven session (see next section), but it is not very practical for defining processes. Indeed, the local types obtained by projection of the global woven type contain all modifications made by aspects. This means that processes have to programmed as if aspect weaving was done by hand.

From a practical point of view, it is more interesting to first project the global session type and aspects to local types and aspects, realize them modularly, and then weave aspects.

Figure 5 depicts the different paths to deal with aspectual session types, starting from global aspects and session type, to obtain the local woven types and processes, either through weaving and projection, or vice versa, through projection and weaving. The realization of local types with processes is checked by a type system that is described in [8] and that we do not consider in this paper.

We define aspects and aspect weaving on global types in Section 4.1. The projection of the global woven type relies on the standard projection operation (Figure 4). In Section 4.2 we describe local aspects, the projection from global aspects to local ones, and local aspect weaving.

4.1 Aspects on Global Types

Syntax. Figure 6 presents the syntax of global aspectual session types $\mathbf{G}_a$. In addition to the base session type $\mathbf{G}$, the programmer can specify a number of aspects $\overline{A}$ of the form of pointcut/advice pairs $\langle pc, adv\rangle$. Messages are the only join points in this model. A pointcut specifies messages of interest, possibly using wildcards on the label (resp. payload type) to represent any label (resp. payload type). The disjunction of pointcuts is noted +, in line with the choice operator of session types. An advice is similar to a standard global type, except for the fact that it can use proceed as a message.

$$\begin{array}{rcll}
\mathbf{G}_a & ::= & \overline{A} \;\oplus\; \mathbf{G} & \text{Global Type with Aspects} \\
A & ::= & \langle pc, adv\rangle & \text{Aspect} \\
pc & ::= & \mathtt{p} \to \underline{\mathtt{p}'} : l_*\langle U_*\rangle \mid pc + pc & \text{Pointcut} \\
adv & ::= & \mathsf{def}\ \overline{G_a}\ \mathsf{in}\ \mathbf{x} & \text{Advice} \\
G_a & ::= & \mathbf{x} = \mathsf{proceed}; \mathbf{x}' \mid G & \text{Advice Transition} \\
l_* & ::= & l \mid * & \text{Labels with wildcard} \\
U_* & ::= & U \mid * & \text{Sorts with wildcard}
\end{array}$$

Figure 6: Syntax of Global Aspectual Session Types

$$\begin{array}{rl}
\left.\begin{array}{r}
match(\mathtt{p} \to \mathtt{p}' : l_*\langle U_*\rangle, \mathtt{p} \to \mathtt{p}' : l\langle U\rangle) \\
match(!\langle \mathtt{p}, l_*\langle U_*\rangle\rangle, !\langle \mathtt{p}, l\langle U\rangle\rangle) \\
match(?\langle \mathtt{p}, l_*\langle U_*\rangle\rangle, ?\langle \mathtt{p}, l\langle U\rangle\rangle)
\end{array}\right\} & \begin{array}{l}\text{if } l_* \in \{l, *\}\wedge \\ \quad U_* \in \{U, *\}\end{array} \\
match(pc_1 + pc_2, M) & \begin{array}{l}\text{if } match(pc_1, M) \vee \\ \quad match(pc_2, M)\end{array}
\end{array}$$

Figure 7: Pointcut Matching

$$\text{(NG-Start)}\ \frac{\overline{A} \vdash \overline{G} \rightsquigarrow_N \overline{G'}}{\overline{A} \;\oplus\; \mathsf{def}\ \overline{G}\ \mathsf{in}\ \mathbf{x} \rightsquigarrow_N \mathsf{def}\ \overline{G'}\ \mathsf{in}\ \mathbf{x}}$$

$$\text{(NG-StepA)}\ \frac{A \vdash \overline{G} \rightsquigarrow_N \overline{G'} \qquad \overline{A} \vdash \overline{G'} \rightsquigarrow_N \overline{G''}}{A\overline{A} \vdash \overline{G} \rightsquigarrow_N \overline{G''}}$$

$$\text{(NG-StepG)}\ \frac{A \vdash G_1 \rightsquigarrow_N \overline{G'_1} \qquad A \vdash \overline{G_2} \rightsquigarrow_N \overline{G'_2}}{A \vdash G_1\overline{G_2} \rightsquigarrow_N \overline{G'_1 G'_2}}$$

$$\text{(NG-Skip)}\ \frac{G \neq \mathbf{x} = M; \mathbf{x}' \vee \neg match(pc, M)}{\langle pc, adv\rangle \vdash G \rightsquigarrow_N G}$$

$$\text{(NG-Weave)}\ \frac{\overline{G}'_A = localize(\overline{G}_A, \mathbf{x}) \qquad match(pc, M)}{\begin{array}{l}\langle pc, \mathsf{def}\ \overline{G}_A\ \mathsf{in}\ \mathbf{x}_A\rangle \vdash \mathbf{x} = M; \mathbf{x}' \rightsquigarrow_N \\ \qquad \mathbf{x} = \mathbf{x}^{\mathbf{x}}_A \\ \qquad \overline{G}'_A[\mathsf{proceed} \mapsto M][\mathsf{end} \mapsto \mathbf{x}']\end{array}}$$

Figure 8: Naive Global Weaving

Pointcut matching. Pointcut matching is straightforward, and defined in Figure 7. Only the first and last definitions apply to the case of global aspects. A pointcut matches a message if both participants are the same and if the pointcut label (resp. payload type) is either the wildcard $*$ or the same as the actual label (resp. payload type). Pointcut disjunction is interpreted as a disjunction of both branches.

Naive Weaving. Weaving of aspectual session types is defined in Figure 8. Aspects are woven one after the other in the order they appear in the list. So once an aspect is woven, it has no effect on the weaving of remaining aspects. This definition of weaving is called *naive*, noted $\rightsquigarrow_N$, because it is simple but can yield ill-formed types, as discussed below. The first three rules express the order of aspect weaving, which processes aspects one at a time and then treats one transition of G at a time. Rule (NG-Skip) applies whenever the transition is not a message or if the pointcut does not match the message. Rule (NG-Weave) specifies the rewriting of the transition $\mathbf{x} = M; \mathbf{x}'$ whenever the pointcut matches M. The rewriting replaces the original transition with the advice definitions, substituting $\mathbf{x}$ for $\mathbf{x}_A$, M for proceed, and $\mathbf{x}'$ for end in the advice body. Note that prior to substitution, all states in the advice body are renamed by annotating them with $\mathbf{x}$, as specified by the localize function:

$$localize(\overline{G}_A, \mathbf{x}) = \overline{G}_A[\mathbf{x}' \mapsto \mathbf{x}'^{\mathbf{x}}] \text{ for all } \mathbf{x}' \in \overline{G}_A$$

This ensures that the uniqueness of states is preserved when the advice is inserted several times within the same global session.

Illustration 1. Consider the naive weaving of the negotiation aspect on $\mathbf{G}_{\text{Trade}}$. The $\mathbf{x}_0$ in exponent comes from the localization process.

$$\begin{array}{rcl}
\multicolumn{3}{l}{A_{\text{nego}} \ \oplus\ \mathbf{G}_{\text{Trade}} \rightsquigarrow_N \mathsf{def}} \\
\mathbf{x_0} &=& \mathbf{x_A}^{\mathbf{x}_0} \\
\mathbf{x_A}^{\mathbf{x}_0} &=& \mathtt{S} \to \mathtt{B}\colon \mathit{Item}\langle \mathit{string}\rangle; \mathbf{x_1}^{\mathbf{x_0}} \\
\mathbf{x_1}^{\mathbf{x}_0} + \mathbf{x_6}^{\mathbf{x}_0} &=& \mathbf{x_2}^{\mathbf{x}_0} \\
\mathbf{x_2}^{\mathbf{x}_0} &=& \mathbf{x_3}^{\mathbf{x}_0} + \mathbf{x_4}^{\mathbf{x}_0} \\
\mathbf{x_3}^{\mathbf{x}_0} &=& \mathtt{B} \to \mathtt{C}\colon \mathit{Offer}\langle \mathit{nat}\rangle; \mathbf{x_5}^{\mathbf{x_0}} \\
\mathbf{x_5}^{\mathbf{x}_0} &=& \mathtt{C} \to \mathtt{B}\colon \mathit{Counter}\langle \mathit{nat}\rangle; \mathbf{x_6}^{\mathbf{x_0}} \\
\mathbf{x_4}^{\mathbf{x}_0} &=& \mathbf{x}_1 \\
\mathbf{x_1} &=& \mathbf{x_2} \mid \mathbf{x_3} \\
\mathbf{x_2} &=& \mathtt{B} \to \mathtt{S}\colon \mathit{Sale}\langle \mathit{boolean}\rangle; \mathbf{x_4} \\
\mathbf{x_3} &=& \mathtt{B} \to \mathtt{C}\colon \mathit{Purchase}\langle \mathit{boolean}\rangle; \mathbf{x_5} \\
\mathbf{x_4} \mid \mathbf{x_5} &=& \mathbf{x}_6 \\
\mathbf{x}_6 &=& \mathsf{end} \quad \mathsf{in}\ \mathbf{x}_0
\end{array}$$

Modulo congruence, this woven global type is the same as the trade session type with negotiation of [8], which is well formed. So it seems that naive aspect weaving provides a good way to enhance interactions while preserving well-formedness. However, the situation is not that simple, as shown by the following example.

Illustration 2. Consider the naive weaving of the authentication aspect on $\mathbf{G}_{\text{Trade}}$. The $\mathbf{x}_2$ and $\mathbf{x}_3$ in exponent come from two different applications of the localization process.

$$\begin{array}{rcl}
\multicolumn{3}{l}{A_{\text{auth}} \ \oplus\ \mathbf{G}_{\text{Trade}} \rightsquigarrow_N \mathsf{def}} \\
\mathbf{x_0} &=& \mathtt{S} \to \mathtt{B}\colon \mathit{Item}\langle \mathit{string}\rangle; \mathbf{x_1} \\
\mathbf{x_1} &=& \mathbf{x_2} \mid \mathbf{x_3} \\
\mathbf{x_2} &=& \mathbf{x_A}^{\mathbf{x_2}} \\
\mathbf{x_A}^{\mathbf{x_2}} + \mathbf{x_5}^{\mathbf{x_2}} &=& \mathbf{x_1}^{\mathbf{x_2}} \\
\mathbf{x_1}^{\mathbf{x_2}} &=& \mathtt{B} \to \mathtt{A}\colon \mathit{Auth}\langle \mathit{string}\rangle; \mathbf{x_2}^{\mathbf{x_2}} \\
\mathbf{x_2}^{\mathbf{x_2}} &=& \mathbf{x_3}^{\mathbf{x_2}} + \mathbf{x_4}^{\mathbf{x_2}} \\
\mathbf{x_3}^{\mathbf{x_2}} &=& \mathtt{A} \to \mathtt{B}\colon \mathit{Retry}; \mathbf{x_5}^{\mathbf{x_2}} \\
\mathbf{x_4}^{\mathbf{x_2}} &=& \mathtt{A} \to \mathtt{B}\colon \mathit{Ok}; \mathbf{x_6}^{\mathbf{x_2}} \\
\mathbf{x_6}^{\mathbf{x_2}} &=& \mathtt{B} \to \mathtt{S}\colon \mathit{Sale}\langle \mathit{boolean}\rangle; \mathbf{x_7}^{\mathbf{x_2}} \\
\mathbf{x_7}^{\mathbf{x_2}} &=& \mathbf{x}_4 \\
\mathbf{x_3} &=& \mathbf{x_A}^{\mathbf{x_3}} \\
\mathbf{x_A}^{\mathbf{x_3}} + \mathbf{x_5}^{\mathbf{x_3}} &=& \mathbf{x_1}^{\mathbf{x_3}} \\
\mathbf{x_1}^{\mathbf{x_3}} &=& \mathtt{B} \to \mathtt{A}\colon \mathit{Auth}\langle \mathit{string}\rangle; \mathbf{x_2}^{\mathbf{x_3}} \\
\mathbf{x_2}^{\mathbf{x_3}} &=& \mathbf{x_3}^{\mathbf{x_3}} + \mathbf{x_4}^{\mathbf{x_3}} \\
\mathbf{x_3}^{\mathbf{x_3}} &=& \mathtt{A} \to \mathtt{B}\colon \mathit{Retry}; \mathbf{x_5}^{\mathbf{x_3}} \\
\mathbf{x_4}^{\mathbf{x_3}} &=& \mathtt{A} \to \mathtt{B}\colon \mathit{Ok}; \mathbf{x_6}^{\mathbf{x_3}} \\
\mathbf{x_6}^{\mathbf{x_3}} &=& \mathtt{B} \to \mathtt{C}\colon \mathit{Purchase}\langle \mathit{boolean}\rangle; \mathbf{x_7}^{\mathbf{x_3}} \\
\mathbf{x_7}^{\mathbf{x_3}} &=& \mathbf{x}_5 \\
\mathbf{x_4} \mid \mathbf{x_5} &=& \mathbf{x}_6 \\
\mathbf{x}_6 &=& \mathsf{end} \quad \mathsf{in}\ \mathbf{x}_0
\end{array}$$

In this case, naive weaving produces an ill-formed type, because it breaks linearity on $\mathtt{B} \to \mathtt{A}\colon \mathit{Auth}\langle \mathit{string}\rangle$ (explained in Section 3.3). We will come back to this in Section 5 when studying properties of aspectual session types. Note already that the size and complexity of the woven type can grow quickly.

$\mathbf{T}_a$	$::=$	$\overline{A} \ \oplus\ \mathbf{T}$	Local Type with Aspects
A	$::=$	$\langle pc, adv\rangle$	Aspect
pc	$::=$	$!\langle \mathtt{p}, l_*\langle U_*\rangle\rangle \mid ?\langle \mathtt{p}, l_*\langle U_*\rangle\rangle$	Pointcut
	$\mid$	$pc + pc \mid 0$	
adv	$::=$	$\mathsf{def}\ \overline{T_a}\ \mathsf{in}\ \mathbf{x}$	Advice
T_a	$::=$	$\mathbf{x} = M_a; \mathbf{x}' \mid T$	Advice Transition
M_a	$::=$	$\mathsf{proceed} \mid M$	Advice Message

Figure 9: Syntax of Local Aspectual Session Types

$$\begin{array}{lcl}
\overline{A} \ \oplus\ \mathbf{G} \upharpoonright \mathtt{p} &=& \overline{A} \upharpoonright \mathtt{p} \ \oplus\ \mathbf{G} \upharpoonright \mathtt{p} \\
\langle pc, adv\rangle \upharpoonright \mathtt{p} &=& \langle pc \upharpoonright \mathtt{p}, adv \upharpoonright \mathtt{p}\rangle \\
pc + pc \upharpoonright \mathtt{p} &=& pc \upharpoonright \mathtt{p} + pc \upharpoonright \mathtt{p} \\
\mathtt{p} \to \mathtt{p}'\colon l_*\langle U_*\rangle \upharpoonright \mathtt{p} &=& !\langle \mathtt{p}', l_*\langle U_*\rangle\rangle \\
\mathtt{p} \to \mathtt{p}'\colon l_*\langle U_*\rangle \upharpoonright \mathtt{p}' &=& ?\langle \mathtt{p}, l_*\langle U_*\rangle\rangle \\
\mathtt{p} \to \mathtt{p}'\colon l_*\langle U_*\rangle \upharpoonright \mathtt{p}'' &=& 0\ (\mathtt{p}'' \notin \{\mathtt{p}, \mathtt{p}'\}) \\
\mathsf{proceed} \upharpoonright \mathtt{p} &=& \mathsf{proceed}
\end{array}$$

Figure 10: Projection Algorithm for Aspects

4.2 Aspects on Local Types

Syntax. The syntax of local aspectual session types is given in Figure 9. Similarly to global aspectual session types, one can specify a set of aspects to weave on a local type. Local join points now denote message sending and reception. In addition to disjunction, we also include the null pointcut 0, which can appear via projection. As before, an advice body can include proceed.

We extend congruence naturally on local types to aspects, in particular to account for 0 in pointcuts:

$$\begin{array}{rclcl}
\langle pc, adv\rangle &\equiv& \langle pc', adv'\rangle & \text{if } pc \equiv pc' \wedge adv \equiv adv' \\
pc + 0 &\equiv& 0 + pc &\equiv& pc
\end{array}$$

Projection. We extend the projection algorithm of Figure 4 to deal with the projection of aspects, shown in Figure 10. A message pointcut is projected as a message send pointcut on the sender, a message reception on the receiver, and 0 otherwise. The rest of the projection is straightforward. We call a projected aspect a *daemon aspect* when its projected pointcut is 0.

Local Weaving. Weaving of local aspectual session types is defined in Figure 11. Compared to global weaving, the novelty here is that it is possible for pointcuts to be projected to 0. Intuitively, this corresponds to advice actions that do not involve either of the participants in the intercepted message. The weaving of such daemon aspects is expressed by Rule (T-Daemon). Rule (T-NoDaemon) corresponds to the non-0 local pointcut case, which, together with rules (T-StepT), (T-Skip) and (T-Weave), is similar to the global weaving rules.

Weaving daemons is challenging because locally, there is no way to know precisely when these advice actions have to be performed. Therefore, they must possibly be realizable at any point, in parallel with the participant's original activity. This is why Rule (T-Daemon) rewrites the local type to introduce a fork that branches between the original activity (which starts at $\mathbf{x}$) and the advice activity[2]. In addition, the advice activity must possibly be executed several times. Therefore, in the advice body, end is substituted with

[2] Note that new participants introduced by aspects are initialized with local types $\mathsf{def}\ \mathbf{x} = \mathsf{end}\ \mathsf{in}\ \mathbf{x}$, so that the local weaving of projected advice is always defined.

$$\text{(T-StepA)}\ \frac{A \vdash \mathbf{T} \rightsquigarrow \mathbf{T}' \qquad \overline{A} \vdash \mathbf{T}' \rightsquigarrow \mathbf{T}''}{A\overline{A} \ \oplus\ \mathbf{T} \rightsquigarrow \mathbf{T}''}$$

$$\text{(T-Daemon)}\ \frac{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_E \text{ fresh in } \overline{T}, \overline{T}_A}{\begin{array}{ll} \langle 0, \mathsf{def}\ \overline{T}_A \ \mathsf{in}\ \mathbf{x}_A\rangle \vdash & \\ & \mathsf{def}\ \mathbf{x}_0 = \mathbf{x} \mid \mathbf{x}_1 \\ \mathsf{def}\ \overline{T}\ \mathsf{in}\ \mathbf{x} \rightsquigarrow & \quad \mathbf{x}_1 + \mathbf{x}_E = \mathbf{x}_A \\ & \quad \overline{T}_A[\mathsf{proceed} \mapsto 0][\mathsf{end} \mapsto \mathbf{x}_E] \\ & \quad \overline{T} \ \ \mathsf{in}\ \mathbf{x}_0 \end{array}}$$

$$\text{(T-NoDaemon)}\ \frac{\langle pc, adv\rangle \vdash \overline{T} \rightsquigarrow \overline{T'} \qquad pc \neq 0}{\langle pc, adv\rangle \vdash \mathsf{def}\ \overline{T}\ \mathsf{in}\ \mathbf{x} \rightsquigarrow \mathsf{def}\ \overline{T'}\ \mathsf{in}\ \mathbf{x}}$$

$$\text{(T-StepT)}\ \frac{A \vdash T_1 \rightsquigarrow \overline{T'_1} \qquad A \vdash \overline{T_2} \rightsquigarrow \overline{T'_2}}{A \vdash T_1\overline{T_2} \rightsquigarrow \overline{T'_1T'_2}}$$

$$\text{(T-Skip)}\ \frac{T \neq \mathbf{x} = M.\mathbf{x}' \vee \neg match(pc, M)}{\langle pc, adv\rangle \vdash T \rightsquigarrow T}$$

$$\text{(T-Weave)}\ \frac{\overline{T}'_A = localize(\overline{T}_A, \mathbf{x}) \qquad match(pc, M)}{\begin{array}{l} \langle pc, \mathsf{def}\ \overline{T}_A\ \mathsf{in}\ \mathbf{x}_A\rangle \vdash \mathbf{x} = M.\mathbf{x}' \rightsquigarrow \\ \qquad \mathbf{x} = \mathbf{x}_A^{\mathbf{x}} \\ \qquad \overline{T}'_A[\mathsf{proceed} \mapsto M][\mathsf{end} \mapsto \mathbf{x}'] \end{array}}$$

Figure 11: Local Weaving

a newly-introduced merge point that connects to the start of the advice $\mathbf{x}_A$. In effect, this weaving strategy serializes all advice executions in a given participant. Finally, because this rule applies in participants that are not involved in the intercepted message, proceed is replaced by the null message 0.

Illustration. We illustrate the local weaving on the negotiation aspect on the client side. It produces a daemon negotiation aspect for C that can be executed infinitely often in parallel with the *Purchase* interaction ($'$s are for states dealing with the daemon).

$$\begin{array}{rcl} A_{\text{nego}} & \upharpoonright \mathsf{C} \ \oplus\ \mathbf{G}_{\text{Trade}} \upharpoonright \mathsf{C} \rightsquigarrow \mathsf{def} & \\ \mathbf{x}'_0 & = & \mathbf{x}_0 \mid \mathbf{x}'_1 \\ \mathbf{x}'_1 + \mathbf{x}'_E & = & \mathbf{x}'_A \\ \mathbf{x}'_A + \mathbf{x}'_6 & = & \mathbf{x}'_2 \\ \mathbf{x}'_2 & = & \mathbf{x}'_3 + \mathbf{x}'_4 \\ \mathbf{x}'_3 & = & ?\langle B, Offer\langle nat\rangle\rangle; \mathbf{x}'_5 \\ \mathbf{x}'_5 & = & !\langle B, Counter\langle nat\rangle\rangle; \mathbf{x}'_6 \\ \mathbf{x}'_4 & = & \mathbf{x}'_E \\ \mathbf{x}_0 & = & \mathbf{x}_2 \mid \mathbf{x}_3 \\ \mathbf{x}_2 & = & \mathbf{x}_4 \\ \mathbf{x}_3 & = & ?\langle B, Purchase\langle boolean\rangle\rangle; \mathbf{x}_5 \\ \mathbf{x}_4 \mid \mathbf{x}_5 & = & \mathbf{x}_6 \\ \mathbf{x}_6 & = & \mathsf{end} \quad \mathsf{in}\ \mathbf{x}'_0 \end{array}$$

Of course, in practice, the daemon will be executed only once for the global type $\mathbf{G}_{\text{Trade}}$, but this cannot be known locally.

Overall, the weaving of daemon advices introduces a mismatch between the global woven type and the local woven types. This means that weaving ($\rightsquigarrow$) and projection ($\upharpoonright$) do not commute: recalling Figure 5, the local types obtained by first weaving aspects in the global type and then projecting the woven type do not match the local types obtained by first projecting the aspects and global type and then locally weaving the aspects. In effect, the types obtained by local weaving describe a more parallel interaction than the interaction described by the global woven type.

This inevitable mismatch potentially breaks the connection between well-formedness of the global type and the desirable properties of local types. The most technical contribution of this work is to characterize sufficient conditions on aspectual session types for local aspect weaving to preserve the properties of the global woven type (Section 5), and to prove this proper relation between local and global weaving (Section 6).

5. Properties of Aspectual Session Types

Generalized multiparty session types come with a strong property: if a global type is well-formed, then the interaction of processes that realize the local projections of this type is free of deadlocks, free of orphan messages, and free of reception errors. Recall that well-formedness crucially relies on the local choice and linearity conditions (Section 3.3). To characterize sufficient conditions for aspect weaving to preserve the properties of the global woven type, we introduce two important conditions on aspects that deal with linearity and locality of choice:

- The *aspectual linearity* condition (Section 5.1) justifies that tagging makes sense.
- The *aspectual local choice* condition ensures that aspect weaving does not confuse participants by introducing opaque decisions (Section 5.2).

These two conditions are used to define well-formedness of aspectual session types in Section 5.4. We describe a correct version of global type weaving in Section 5.3, which refines naive weaving in order to address both conditions above. In Section 6, we formally prove that well-formedness of a global type with aspects effectively entails the same safety properties as that of global types.

5.1 Aspectual Linearity

Naive weaving produces a well-formed type in the negotiation example because the pointcut matches only one message. More generally, a sufficient condition to avoid race conditions for different advice executions is to rely directly on the linearity of global types. Namely, if the result of naive weaving is well-formed, two executions of the same advice can never run in parallel, and so there is no race condition between them. This restrictive but sufficient condition—a pointcut does not match two concurrent branches—is called *pointcut linearity*.

Definition 2 (Pointcut Linearity)**.** *Let A be an aspect and* $\mathbf{G}$ *be a well-formed base type. A is* pointcut-linear *wrt* $\mathbf{G}$ *iff* $A \oplus \mathbf{G} \rightsquigarrow_N \mathbf{G}'$ *for some well-formed* $\mathbf{G}'$.

Of course, more interesting aspects can match different messages. For instance, the logging and authentication pointcuts match non-linearly as they match two different messages occurring in potentially-concurrent branches. In those cases, naive global weaving can lead to ill-formed global types that break linearity, as illustrated in Illustration 2 of Section 4.2.

Relaxing linearity with tagged weaving. The authentication and logging aspects introduce identical behaviors in different threads, but seem intuitively correct. This conflict reflects the fact that the linearity requirement is too strong in most cases, because advice is typically defined in a general manner, meant to react regardless of specific thread/choice contexts.

Technically, we can sidetrack the linearity issue with advice through *tagged weaving*: deploying a different version of an advice for each message that is matched. These different versions are obtained by using fresh labels tagged with the matched message. For instance, tagged weaving modifies the woven type of the authentication aspect on $\mathbf{G}_{\mathrm{Trade}}$ (already presented in Illustration 2) by tagging each occurrence of *Auth*, *Ok* and *Retry* with the matched message:

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\quad A_{\mathrm{auth}} \oplus \mathbf{G}_{\mathrm{Trade}} \rightsquigarrow \mathsf{def}} \\
\ldots & & \text{(same definitions)} \\
\mathbf{x_1}^{\mathbf{x}_2} & = & \mathtt{B} \to \mathtt{A}\colon \mathit{Auth}^{\mathtt{B}\to\mathtt{S}\colon \mathit{Sale}\langle \mathit{boolean}\rangle}\langle \mathit{string}\rangle; \mathbf{x_2}^{\mathbf{x}_2} \\
\mathbf{x_3}^{\mathbf{x}_2} & = & \mathtt{A} \to \mathtt{B}\colon \mathit{Retry}^{\mathtt{B}\to\mathtt{S}\colon \mathit{Sale}\langle \mathit{boolean}\rangle}; \mathbf{x_5}^{\mathbf{x}_2} \\
\mathbf{x_4}^{\mathbf{x}_2} & = & \mathtt{A} \to \mathtt{B}\colon \mathit{Ok}^{\mathtt{B}\to\mathtt{S}\colon \mathit{Sale}\langle \mathit{boolean}\rangle}; \mathbf{x_6}^{\mathbf{x}_2} \\
\ldots & & \text{(same definitions)} \\
\mathbf{x_1}^{\mathbf{x}_3} & = & \mathtt{B} \to \mathtt{A}\colon \mathit{Auth}^{\mathtt{B}\to\mathtt{C}\colon \mathit{Purchase}\langle \mathit{boolean}\rangle}\langle \mathit{string}\rangle; \mathbf{x_2}^{\mathbf{x}_3} \\
\mathbf{x_3}^{\mathbf{x}_3} & = & \mathtt{A} \to \mathtt{B}\colon \mathit{Retry}^{\mathtt{B}\to\mathtt{C}\colon \mathit{Purchase}\langle \mathit{boolean}\rangle}; \mathbf{x_5}^{\mathbf{x}_3} \\
\mathbf{x_4}^{\mathbf{x}_3} & = & \mathtt{A} \to \mathtt{B}\colon \mathit{Ok}^{\mathtt{B}\to\mathtt{C}\colon \mathit{Purchase}\langle \mathit{boolean}\rangle}; \mathbf{x_6}^{\mathbf{x}_3} \\
\ldots & & \text{(same definitions)} \\
\mathbf{x}_6 & = & \mathsf{end} \quad \mathsf{in}\ \mathbf{x}_0
\end{array}
$$

By linearity of the initial global type, the same message cannot occur in two concurrent branches, so the same version of the advice cannot be executed twice in parallel. Therefore, well-formedness of the base type implies well-formedness of the tagged woven type. Note that tagging can only be done in the global type: in the local types, remote join points are not know, so it is impossible to relate messages properly.

Tagging is a technical device that sidetracks the strong linearity requirement but can obviously break the good properties of session types that are derived from well-formedness. For tagging to be compatible with linearity, the advice must satisfy an *advice linearity* condition. This condition ensures that even though tagging is not done locally, the locally-woven type *behaves as if* it were tagged. This property is proven in Section 6.4 by means of a simulation result.

Definition 3 (Advice Linearity)**.** *Let $A = \langle pc, adv\rangle$ be an aspect and $\mathbf{G}$ be a well-formed base type, such that $\mathbf{G} = \mathsf{def}\ \overline{G}\ \mathsf{in}\ \mathbf{x}_0$. A is* advice-linear *wrt $\mathbf{G}$ iff $\mathbf{G}, A \vdash \mathit{UniqueMsg}$ and for all M such that $\mathit{match}(pc, M)$, adv is single-threaded wrt M.*

Advice linearity relies on uniqueness of messages and single-threadedness. Both are formally defined in Appendix B. The first property ensures that two concurrent executions of a daemon can not interfere. An advice *adv* is said to be *single-threaded* wrt a message $M = \mathtt{p} \to \mathtt{p}'\colon l\langle U\rangle$ if, in the projections of $\mathit{adv}[\mathsf{proceed} \mapsto M]$ on $\mathtt{p}$ and $\mathtt{p}'$, there is only one thread communicating with daemon advices. That is $\mathit{adv}[\mathsf{proceed} \mapsto M] \upharpoonright \mathtt{p}, M \vdash \mathit{SgTh}(b_{\mathtt{p}})$ (resp. with $\mathtt{p}'$) with $b_{\mathtt{p}}$ or $b_{\mathtt{p}'}$ equal to false. The single-threaded condition ensures that the part of the advice that is not a daemon advice can not do a join on messages coming from different daemon advices, that are tagged with different labels.

Aspectual linearity follows from either pointcut linearity or advice linearity:

Definition 4 (Aspectual Linearity)**.** *An aspect A satisfies the* aspectual linearity *condition wrt a base type $\mathbf{G}$ iff A is either pointcut-linear or advice-linear wrt to $\mathbf{G}$.*

5.2 Aspectual Local Choice

Aspect weaving introduces an extra implicit choice (is the original message happening, or is an aspect applied?), so it should not break the local choice condition. Informally, this means that participants that are not directly informed of the decision (the source and target of the intercepted message) should either not depend on the choice

$$
\text{(G-Start)}\ \frac{\overline{A} \vdash \overline{G} \rightsquigarrow \overline{G'}}{\overline{A} \oplus \mathsf{def}\ \overline{G}\ \mathsf{in}\ \mathbf{x} \rightsquigarrow \mathsf{def}\ \overline{G'}\ \mathsf{in}\ \mathbf{x}}
$$

$$
\text{(G-StepA)}\ \frac{A,\ \overline{G} \vdash \overline{G} \rightsquigarrow \overline{G'} \qquad \overline{A} \vdash \overline{G'} \rightsquigarrow \overline{G''}}{A\overline{A} \vdash \overline{G} \rightsquigarrow \overline{G''}}
$$

$$
\text{(G-StepG)}\ \frac{A,\ \overline{G} \vdash G_1 \rightsquigarrow \overline{G'_1} \qquad A,\ \overline{G} \vdash \overline{G_2} \rightsquigarrow \overline{G'_2}}{A,\ \overline{G} \vdash G_1\overline{G_2} \rightsquigarrow \overline{G'_1 G'_2}}
$$

$$
\text{(G-Skip)}\ \frac{G \neq \mathbf{x} = M; \mathbf{x}' \vee \neg \mathit{match}(pc, M)}{\langle pc, adv\rangle,\ \overline{G} \vdash G \rightsquigarrow G}
$$

$$
\text{(G-Weave)}\ \frac{\begin{array}{c} \mathit{match}(pc, M) \qquad M = \mathtt{p} \to \mathtt{p}'\colon l\langle U\rangle \\ \overline{G}'_A = \mathit{tag}(\mathit{localize}(\overline{G}_A, \mathbf{x}), M, \overline{G}) \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, l' \text{ fresh in } \overline{G}, \overline{G}_A \end{array}}{\begin{array}{l} \langle pc, \mathsf{def}\ \overline{G}_A\ \mathsf{in}\ \mathbf{x}_A\rangle,\ \overline{G} \vdash \\ \mathbf{x} = M; \mathbf{x}' \rightsquigarrow \left[\begin{array}{l} \mathbf{x} = \mathbf{x}_1 + \mathbf{x}_A^{\mathbf{x}} \\ \mathbf{x}_1 = M\ [l \mapsto l']\ ; \mathbf{x}_2 \\ \overline{G}'_A\ [\mathsf{proceed} \mapsto M][\mathsf{end} \mapsto \mathbf{x}_3\ ] \\ \mathbf{x}_2 + \mathbf{x}_3 = \mathbf{x}' \end{array} \right. \end{array}}
$$

Figure 12: Global Weaving (tagging highlighted in blue with light gray boxes, local choice transformation highlighted in red with dark gray boxes)

taken or be explicitly informed of the choice.[3] To avoid introducing confusion by breaking locality of choice, aspect weaving should preserve the unique active senders and choice awareness conditions in the base type.

We could define aspectual local choice using modified definitions of *ASend* and *Rcv*. However, it is possible to check both properties by changing global weaving in a way that makes the implicit choice introduced by aspect weaving explicit, as discussed below. This implies that checking well-formedness on the global woven type ensures aspectual local choice.

5.3 Global Type Weaving Revisited

We now describe global type weaving (Figure 12) as a refinement and extension of naive global weaving, which includes both tagging and a transformation for aspectual local choice. The only rule that has to be changed wrt naive weaving is Rule (G-Weave).

The first change is the definition of $\overline{G}'_A$ (in blue), which deals with the tagging of labels. Advice tagging is defined in Figure 13. The second change in (G-Weave) is the "fake" choice introduced (in red) between the execution of the advice and the sending of a fresh version of M. Introducing this choice explicitly in the structure of the interaction ensures that well-formedness of the woven global type implies aspectual local choice.

We recursively extend the definition of aspectual linearity on global types with aspects. A global type with aspects $A\overline{A} \oplus \mathbf{G}$ satisfies the aspectual linearity condition if A satisfies the condition

[3] Note that our treatment of implicit choice would allow us to support pointcuts with dynamic conditions in a similar manner.

$$\begin{array}{rcl}
tag(\overline{G}_A, M, \overline{G}) & = & \overline{G}_A[l \mapsto l^M]\ \forall l \in F \\
 & & \text{where } F = labels(\overline{G}_A) \setminus labels(\overline{G}) \\
 & & \\
labels(\overline{G}) & = & \bigcup_{G \in \overline{G}} labels(G) \\
labels(\mathbf{x} = \mathtt{p} \to \mathtt{p}' : l\langle U\rangle; \mathbf{x}') & = & l \\
labels(G) & = & \emptyset \quad \text{if } G \neq \mathbf{x} = M; \mathbf{x}'
\end{array}$$

Figure 13: Advice Tagging

wrt $\mathbf{G}$ and $\overline{A} \oplus \mathbf{G}'$ satisfies the aspectual linearity condition (where $A \oplus \mathbf{G} \rightsquigarrow \mathbf{G}'$).

Also, with the final definition of global type weaving, we can now formally define the aspectual local choice condition:

Definition 5 (Aspectual Local Choice)**.** *An aspect A satisfies the* aspectual local choice *condition wrt a base type $\mathbf{G}$ iff $A \oplus \mathbf{G} \rightsquigarrow \mathbf{G}'$ and $\mathbf{G}'$ is well-formed*

This definition is naturally extended to an aspectual session type $\overline{A} \oplus \mathbf{G}$.

5.4 Well-formed Aspectual Session Types

To express well-formedness of aspectual session types, we introduce two sanity conditions, in addition to aspectual linearity and aspectual local choice. First, every advice must have an end so as to ensure that the constructions of (G-Weave), (T-Weave) and (T-Daemon) satisfy the sanity check for session types. Second, every message from/to a daemon advice must be fresh with respect to the base session type $\mathbf{G}$. Intuitively, a daemon adds an interaction that was not present and if it uses messages that are already in $\mathbf{G}$, it may break local choice or linearity.

Definition 6 (Well-formedness)**.** *An aspectual session type $\overline{A} \oplus \mathbf{G}$ is* well-formed *when*

- $\mathbf{G}$ *is well-formed,*
- *every advice has an* end*,*
- *every message from/to a daemon advice is fresh wrt* $\mathbf{G}$*,*
- $\overline{A} \oplus \mathbf{G}$ *satisfies the aspectual linearity condition,*
- $\overline{A} \oplus \mathbf{G}$ *satisfies the aspectual local choice condition.*

6. Relating Local and Global Weaving

To compare local and global weaving, we need to define a formal semantics for local and global types. Following [8], we use the notion of Communicating Finite State Machines (CFSMs) to interpret local types and projections of global types.

In this section, we show that the interpretation of local woven types is simulated by the interpretation of the global woven type. This allows us to preserve all the properties satisfied by global types on a realization that is less sequential and that uses less labels.

Note that Sections 6.1 and 6.2 present background material directly based on [8]. Our contribution starts in Section 6.3.

6.1 Communicating Finite State Machines

Definition 7 (CFSM)**.** *A communicating finite state machine is a finite state transition system $M = (\mathcal{Q}, C, q_0, \mathbb{L}, \delta)$ where $\mathcal{Q}$ is a finite set of states, $C = \{\mathtt{pq} \in \mathcal{P}^2 | \ \mathtt{p} \neq \mathtt{q}\}$, q_0 is the initial state, $\mathbb{L}$ is a finite set of labels, $\delta \subseteq \mathcal{Q} \times (C \times \{!, ?\} \times \mathbb{L}) \times \mathcal{Q}$ is a finite set of transitions.*

A transition of the form $\mathtt{pq}!l$ is called a *sending action* from participant $\mathtt{p}$ to participant $\mathtt{q}$, and a transition of the form $\mathtt{pq}?l$ is called a *receiving action* from $\mathtt{p}$ by $\mathtt{q}$. When the kind of action is not relevant, we use a, a' to range over sending or receiving actions.

A path in M is a finite sequence of states $q_0 \cdots q_n$ starting from the initial state q_0 such that $(q_i, a, q_{i+1}) \in \delta$.

Definition 8 (CS)**.** *A (communicating) system $\mathcal{S} = (M_\mathtt{p})_{\mathtt{p}\in\mathcal{P}}$ is a tuple of CFSMs $M_\mathtt{p} = (\mathcal{Q}_\mathtt{p}, C, q_{0\mathtt{p}}, \mathbb{L}, \delta_\mathtt{p})$ that share the same channels .*

Given a CS $\mathcal{S}$, we note $\delta = \biguplus_{\mathtt{p}\in\mathcal{P}} \delta_\mathtt{p}$ the resulting transition on the system. A *configuration* of $\mathcal{S}$ is a couple $c = ((q_\mathtt{p})_{\mathtt{p}\in\mathcal{P}}, (\mathbf{w}_{\mathtt{pq}})_{\mathtt{p}\neq\mathtt{q}\in\mathcal{P}})$ where $q_p \in \mathcal{Q}_\mathtt{p}$ is a state and $\mathbf{w}_{\mathtt{pq}} \in \mathbb{L}^*$ is a queue of messages. The initial configuration of a system is $c_0 = ((q_{0\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}, \overline{\varepsilon})$. There are two kinds of transitions t on a communicating system $\mathcal{S}$ from c to c', written $c \xrightarrow{t} c'$, that evolve in only one CFSM:

send. $t = (q_\mathtt{p}, \mathtt{pq}!l, q'_\mathtt{p}) \in \delta_\mathtt{p}$ with $q'_{\mathtt{p}'} = q_{\mathtt{p}'}$ for all $\mathtt{p}' \neq \mathtt{p}$ and $\mathbf{w}'_{\mathtt{pq}} = \mathbf{w}_{\mathtt{pq}}\ l$ and $\mathbf{w}'_{\mathtt{p}'\mathtt{q}'} = \mathbf{w}_{\mathtt{p}'\mathtt{q}'}$ for all $\mathtt{p}'\mathtt{q}' \neq \mathtt{pq}$.

receive. $t = (q_\mathtt{p}, \mathtt{pq}?l, q'_\mathtt{p}) \in \delta_\mathtt{p}$ with $q'_{\mathtt{p}'} = q_{\mathtt{p}'}$ for all $\mathtt{p}' \neq \mathtt{p}$ and $\mathbf{w}_{\mathtt{pq}} = l\ \mathbf{w}'_{\mathtt{pq}}$ and $\mathbf{w}'_{\mathtt{p}'\mathtt{q}'} = \mathbf{w}_{\mathtt{p}'\mathtt{q}'}$ for all $\mathtt{p}'\mathtt{q}' \neq \mathtt{pq}$.

The conditions on $\mathbf{w}_{\mathtt{pq}}$ express that a send performs an enqueue operation, and a receive performs a dequeue operation.

An *execution* of a CS is a sequence of transitions starting from the initial configuration:

$$c_0 \xrightarrow{t_0} c_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} c_n.$$

A configuration is *reachable* when there is an execution that leads to it. We note $RS(\mathcal{S})$ the set of reachable configurations of $\mathcal{S}$. A configuration $(\overline{q}, \overline{\varepsilon})$ is *final* when every state of $\overline{q}$ is final.

Property 1. *Let $\mathcal{S}$ be a communicating system and $c = (\overline{q}, \overline{\mathbf{w}})$ a configuration of $\mathcal{S}$. The following definitions follow [5]:*

- *c is a* deadlock configuration *if $\overline{\mathbf{w}} = \varepsilon$ and each state of $\overline{q}$ has only outgoing receiving transitions, i.e., all machine are blocked waiting for messages.*
- *c is an* orphan message configuration *if all states of $\overline{q}$ are final but the queue $\overline{\mathbf{w}}$ of messages is not empty.*
- *c is an* reception error configuration *if there exists a participant $\mathtt{p}$ which has only outgoing receiving transitions $(q_\mathtt{p}, \mathtt{qp}?l, q'_\mathtt{p}$ with $\mathbf{w}_{\mathtt{qp}} = l'\mathbf{w}'_{\mathtt{qp}}$ and $l \neq l'$.*

Property 2. *A communicating system $\mathcal{S}$ satisfies the* progress property *when for all $c \in RS(\mathcal{S})$, either c is final or $c \to c'$ for some configuration c'. It satisfies the* liveness property *when a final configuration can always be reached from any reachable configuration.*

6.2 Multiparty session automata

This section sums up the interpretation of generalized multiparty session types in terms of CFSMs called *multiparty session automata* (MSA), and the main properties satisfied by MSA.

In what follows, $\mathbf{X}$ denotes a collection of states of a local type connected by $|$ and $\mathbf{X}[-]$ a context with a hole. Given a list of local type $\overline{T}$, $\mathbf{X}$ comes with a notion of congruence $\equiv_{\overline{T}}$ defined in [8, page 11].

Given a local type $\mathbf{T} = \mathsf{def}\ \overline{T}\ \mathsf{in}\ \mathbf{x}_0$ obtained by projecting a global type $\mathbf{G}$ on a participant $\mathtt{p}$, we define the CFSM $\mathcal{A}(\mathbf{T}) = (\mathcal{Q}, C, \mathbf{x}_0, labels(\mathbf{G}), \delta)$ as follows:

- $\mathcal{Q}$ is defined as the set of $\mathbf{X}$ build over states of $\mathbf{T}$, up to the congruence $\equiv_{\overline{T}}$,
- $C = \{\mathtt{pq} \mid \mathtt{p}, \mathtt{q} \in \mathbf{G}\}$,
- $(\mathbf{X}[\mathbf{x}], \mathtt{pp}'!l, \mathbf{X}[\mathbf{x}']) \in \delta$ iff $\mathbf{x} = !\langle \mathtt{p}', l\langle U\rangle\rangle; \mathbf{x}' \in \overline{T}$,
- $(\mathbf{X}[\mathbf{x}], \mathtt{p}'\mathtt{p}?l, \mathbf{X}[\mathbf{x}']) \in \delta$ iff $\mathbf{x} = ?\langle \mathtt{p}', l\langle U\rangle\rangle; \mathbf{x}' \in \overline{T}$.

A *multiparty session automaton* (MSA) is a communicating system of the form $(\mathcal{A}(\mathbf{G} \upharpoonright \mathtt{p}))_{\mathtt{p} \in \mathbf{G}}$ for a well-formed global type $\mathbf{G}$.

Below are the main properties satisfied by MSAs, proven in [8], which we want to extend to aspectual session types:

Theorem 1 (from [8])**.** *A multiparty session automaton is free from deadlock, orphan message, and reception error configurations. It satisfies the progress property, and when the global type that generated it contains* end*, it satisfies the liveness property.*

6.3 Aspectual multiparty session automata

Aspectual session types give rise to two kinds of communication systems, depending on whether we consider global or local weaving. The goal of this section and of Section 6.4 is to show that the system coming from global weaving, which inherits directly the properties of MSAs, can simulate the system coming from local weaving.

An *aspectual multiparty session automaton*, noted $\mathcal{A}(\overline{A}, \mathbf{G})$, is a MSA of a well-formed woven global type $\mathbf{G}'$, with $\overline{A} \oplus \mathbf{G} \rightsquigarrow \mathbf{G}'$ and $\overline{A} \oplus \mathbf{G}$ is well-formed.

A *locally woven multiparty session automaton* is a communicating system of the form $(\mathcal{A}(\mathbf{T}_\mathtt{p}))_{\mathtt{p} \in \mathbf{G}}$, where $(\overline{A} \oplus \mathbf{G} \upharpoonright \mathtt{p} \rightsquigarrow \mathbf{T}_\mathtt{p})_{\mathtt{p} \in \mathbf{G}}$ is a set of woven local types coming from a well-formed aspectual session type $\overline{A} \oplus \mathbf{G}$.

Advice atomic executions. We want to relate configurations in $(\mathcal{A}(\mathbf{T}_\mathtt{p}))_{\mathtt{p} \in \mathbf{G}}$ to configurations in $\mathcal{A}(\overline{A}, \mathbf{G})$. But configurations in $(\mathcal{A}(\mathbf{T}_\mathtt{p}))_{\mathtt{p} \in \mathbf{G}}$ are less sequential because daemon advices execute in parallel with the initial interaction. We first need to restrict the kind of configurations we have to deal with, by adding an atomicity condition that forces an interaction on an advice to be completed before another one starts.

Definition 9 (Advice atomic execution)**.** *An execution e in $(\mathcal{A}(\mathbf{T}_\mathtt{p}))_{\mathtt{p} \in \mathbf{G}}$ is* advice atomic *when for any transition $c \xrightarrow{t} c'$ in e, with $t = (\mathbf{X}[\mathbf{x}], \mathtt{pq}!l, \mathbf{X}[\mathbf{x}'])$ between two states $\mathbf{x}$ and $\mathbf{x}'$ defined in $\mathbf{G}$, states present in c and c' are not localized states (of the form $\mathbf{x}_2^{\mathbf{x}_1}$, generated by Rule (T-Weave)), and states present in c that deal with daemon advice are all equal to their corresponding $\mathbf{x}_E$ (generated by Rule (T-Daemon)). By extension, we say that a configuration is* advice atomic *when it can be reached by an advice atomic execution.*

Of course, every configuration is not advice atomic, but using aspectual linearity, we can show that every configuration can be extended to a configuration that satisfies this atomicity.

Property 3. *Suppose $c \in RS((\mathcal{A}(\mathbf{T}_\mathtt{p}))_{\mathtt{p} \in \mathbf{G}})$. Then, c can be extended to $c \to^* c'$ such that c' can be reached by an advice atomic execution.*

Proof. We first show that two parallel executions of the same advice cannot interfere with each other.
Case of advice linearity. Using the unique message property, an execution of a daemon advice cannot be perturbed by message coming from other threads. Using the single-threaded condition, only one non-daemon advice (that is, the projection of an advice on one of the two participants of the matched message) can be waiting for a message coming from only one daemon at a time. It follows that two daemons of the same advice cannot interact in different threads at the same time.
Case of pointcut linearity. Using pointcut linearity, we know that two executions of the same advice cannot be done in parallel, because there is no race condition, even without tagging.

Using well-threadedness, we can reach the end state of every pending advice execution. Then, because every message of an advice is fresh (extra sanity condition), it can be commuted with the remaining interaction in the initial global type $\mathbf{G}$. □

6.4 Simulation of locally woven MSAs.

Intuitively, a communicating system $\mathcal{S}$ is simulated by a communicating system $\mathcal{S}'$ when every action performed in $\mathcal{S}$ can be replicated in some way in $\mathcal{S}'$. More formally, in our setting, a simulation $\mathcal{R}$ from $\mathcal{S}$ to $\mathcal{S}'$ is a relation between configurations of $\mathcal{S}$ and $\mathcal{S}'$ such that the initial configurations are related (i.e. $c_0 \,\mathcal{R}\, c'_0$) and when $c_1 \,\mathcal{R}\, c'_1$ and c_1 reduces in one step to c_2, there exists c'_2 such that $c_2 \,\mathcal{R}\, c'_2$ and c'_1 reduces (in 0, 1 or more steps) to c'_2. We illustrate this with the following diagrams:

$$\begin{tikzcd} c_0 \arrow[r, dash, "\mathcal{R}"] & c'_0 \end{tikzcd} \qquad \begin{tikzcd} c_1 \arrow[r, dash, "\mathcal{R}"] \arrow[d] & c'_1 \arrow[d, dashed, "*"] \\ c_2 \arrow[r, dash, dashed, "\mathcal{R}"] & c'_2 \end{tikzcd}$$

For any participant $\mathtt{p}$ and aspect $A \in \overline{A}$, we define a partial function $\sigma^A_\mathtt{p}$ from states of the CFSM of $\mathcal{A}(A, \mathbf{G})$ at $\mathtt{p}$ to states of the CFSM of $\mathcal{A}(\mathbf{T}_\mathtt{p})$ defined as the identity when $A \upharpoonright \mathtt{p}$ is not a daemon aspect, and—when it is a daemon aspect—defined as:

$$\begin{array}{rcll} \underline{\sigma}^A_\mathtt{p}([-]) & = & [-] & \\ \underline{\sigma}^A_\mathtt{p}(\mathbf{x}) & = & \mathbf{x} & \text{when } \mathbf{x} \in \mathbf{G} \\ \underline{\sigma}^A_\mathtt{p}(\mathbf{x}_1^{\mathbf{x}_2}) & = & \mathbf{x}_1 | \mathbf{x}_2 & \text{when } \mathbf{x}_1 \in A, \mathbf{x}_2 \in \mathbf{G} \\ \underline{\sigma}^A_\mathtt{p}(\mathbf{X}|\mathbf{X}') & = & \underline{\sigma}^A_\mathtt{p}(\mathbf{X}) | \underline{\sigma}^A_\mathtt{p}(\mathbf{X}') & \text{when at most one state of} \\ & & & \underline{\sigma}^A_\mathtt{p}(\mathbf{X}) \text{ or } \underline{\sigma}^A_\mathtt{p}(\mathbf{X}') \text{ is in } A \\ \underline{\sigma}^A_\mathtt{p}(\mathbf{X}) & & \multicolumn{2}{l}{\text{is otherwise undefined}} \end{array}$$

We then set $\sigma^A_\mathtt{p}(\mathbf{X}) = \underline{\sigma}^A_\mathtt{p}(\mathbf{X}) | \mathbf{x}_E$ when $\underline{\sigma}^A_\mathtt{p}(\mathbf{X})$ contains no state in A, with $\mathbf{x}_A$ is the initial state of A. Otherwise, $\sigma^A_\mathtt{p}(\mathbf{X}) = \underline{\sigma}^A_\mathtt{p}(\mathbf{X})$. Note that $\sigma^A_\mathtt{p}$ is defined in two times using the auxiliary function $\underline{\sigma}^A_\mathtt{p}$ because a state of $\mathcal{A}(A, \mathbf{G})$ at $\mathtt{p}$ may contain no state of A, whereas every state of $\mathcal{A}(\mathbf{T}_\mathtt{p})$ contains exactly one state of A (even if just $\mathbf{x}_E$) because a daemon advice always executes in parallel with the rest of the local type.

This means that $\sigma^A_\mathtt{p}(\mathbf{X})$ is undefined when there is more than one execution of the daemon aspect in parallel because parallel executions of the daemon aspect are not possible in the locally woven type. It is also undefined when we are in a fresh (red) state of Rule (G-Weave), because such a state has been added to check the aspectual local choice condition, but it has no computational meaning and is not present in the locally woven type.

We extend the definition of $\sigma_\mathtt{p}$ to a list of aspects

$$\sigma^\varepsilon_\mathtt{p} = id \qquad \text{and} \qquad \sigma^{A\overline{A}}_\mathtt{p} = \sigma^{\overline{A}}_\mathtt{p} \circ \sigma^A_\mathtt{p}.$$

We now use $\sigma_\mathtt{p}$ to define the relation $\mathcal{R}_{Tag}$ between configurations of $(\mathcal{A}(\mathbf{T}_\mathtt{p}))_{\mathtt{p} \in \mathbf{G}}$ and configurations of $\mathcal{A}(\overline{A}, \mathbf{G})$ as

$$((\mathbf{X}_\mathtt{p})_{\mathtt{p} \in \mathcal{P}}, (\mathbf{w}_{\mathtt{pq}})_{\mathtt{p} \neq \mathtt{q} \in \mathcal{P}}) \,\mathcal{R}_{Tag} ((\mathbf{X}'_\mathtt{p})_{\mathtt{p} \in \mathcal{P}}, (\mathbf{w}'_{\mathtt{pq}})_{\mathtt{p} \neq \mathtt{q} \in \mathcal{P}})$$

exactly when

$$\forall \mathtt{p} \neq \mathtt{q} \in \mathcal{P},\ \mathbf{X}_\mathtt{p} = \sigma^{\overline{A}}_\mathtt{p}(\mathbf{X}'_\mathtt{p}) \quad \text{and} \quad \mathbf{w}_{\mathtt{pq}} = untag(\mathbf{w}'_{\mathtt{pq}}, \mathbf{G}).$$

where $untag(\mathbf{w}'_{\mathtt{pq}}, \mathbf{G})$ is the function that removes in $\mathbf{w}'_{\mathtt{pq}}$ the tags introduced on labels not present in $\mathbf{G}$.

Theorem 2. *The relation $\mathcal{R}_{Tag}$ defines a simulation from advice atomic configurations of $(\mathcal{A}(\mathbf{T}_\mathtt{p}))_{\mathtt{p} \in \mathbf{G}}$ to configurations of $\mathcal{A}(\overline{A}, \mathbf{G})$.*

Proof. Initial configurations are in $\mathcal{R}_{Tag}$ because queues are empty and $\sigma^{\overline{A}}_\mathtt{p}$ is the identity on the initial state.

Suppose $c_1 \mathcal{R}_{Tag} c'_1$ and c_1 reduces in one step to c_2 with transition $t = (\mathbf{X}[\mathbf{x}_{1\mathrm{p}}], M, \mathbf{X}[\mathbf{x}_{2\mathrm{p}}]) \in \delta_{\mathrm{p}}$. By definition of $\mathcal{R}_{Tag}$, the configuration c'_n at p, say $\mathbf{X}'$, satisfies $\sigma_{\mathrm{p}}^{\overline{A}}(\mathbf{X}') = \mathbf{X}[\mathbf{x}_{1\mathrm{p}}]$. We proceed by case analysis on $\mathbf{X}[\mathbf{x}_{1\mathrm{p}}]$.
Case $\mathbf{x}_{1\mathrm{p}} = M; \mathbf{x}_{2\mathrm{p}} \in A \upharpoonright \mathrm{p}$, A is not a daemon at p.
M has been inserted in the interaction by Rule (T-Weave) for a certain matched message that is the projection of a global message, say M_G. Because the configuration is advice atomic, M_G is the only currently matched message in the interaction. By definition of σ_{p}, $\mathbf{X}'$ is of the form $\mathbf{X}'[\mathbf{x}_{1\mathrm{p}}]$. It is not difficult to check that

$$t' = (\mathbf{X}'[\mathbf{x}_{1\mathrm{p}}], tag(M, M_G, \overline{G}), \mathbf{X}'[\mathbf{x}_{2\mathrm{p}}]) \in \delta'_{\mathrm{p}}$$

and $c_2 \mathcal{R}_{Tag} c'_2$ with $c'_1 \xrightarrow{t'} c'_2$.
Case $\mathbf{x}_{1\mathrm{p}} = M; \mathbf{x}_{2\mathrm{p}} \in A \upharpoonright \mathrm{p}$, A is a daemon at p.
M has been inserted in the interaction by Rule (T-Daemon). By aspectual local choice, the daemon cannot be an active sender, so the interaction in the advice has already started on the sender of the matched message. Thus, $\mathbf{X}'$ is of the form $\mathbf{X}'[\mathbf{x}^{\mathbf{x}}_{1\mathrm{p}}]$. The state $\mathbf{x}^{\mathbf{x}}_{1\mathrm{p}}$ has been introduced by the localization process of Rule (G-Weave), for a matched pattern M_G (unique by advice atomicity). Again, it is easy to check that

$$t' = (\mathbf{X}'[\mathbf{x}^{\mathbf{x}}_{1\mathrm{p}}], tag(M, M_G, \overline{G}), \mathbf{X}'[\mathbf{x}^{\mathbf{x}}_{2\mathrm{p}}]) \in \delta'_{\mathrm{p}}$$

and $c_2 \mathcal{R}_{Tag} c'_2$ with $c'_1 \xrightarrow{t'} c'_2$.
Case $\mathbf{x}_{1\mathrm{p}} = M; \mathbf{x}_{2\mathrm{p}} \in G \upharpoonright \mathrm{p}$.
In this case, both interpretations do not coincide as local weaving may introduce daemons in parallel with the initial interaction. But c_1 is advice atomic, so there is no pending interaction in daemons advices, which means that $\mathbf{X}'$ is of the form $\mathbf{X}'[\mathbf{x}_{1\mathrm{p}}]$ and

$$t' = (\mathbf{X}'[\mathbf{x}_{1\mathrm{p}}], M, \mathbf{X}'[\mathbf{x}_{2\mathrm{p}}]) \in \delta'_{\mathrm{p}}$$

and $c_2 \mathcal{R}_{Tag} c'_2$ with $c'_1 \xrightarrow{t'} c'_2$.

□

6.5 Properties of locally woven MSAs

Aspectual multiparty session automata satisfy properties of Theorem 1 because they are just a particular kind of MSAs. But locally woven multiparty session automata are not proper MSAs because they may break the linearity property as they allow race conditions. In Section 6.3, we have shown that although more liberal, every configuration of a locally woven MSA can reach an advice atomic configuration that is a configuration in which all executions of advices have been done sequentially. Then, in Section 6.4, we have shown that advice atomic configurations of a locally woven MSA can be simulated by configurations of the corresponding aspectual MSA. As all properties of Theorem 1 are closed by reduction, we can directly deduce the main theorem of this paper.

Theorem 3 (Property of locally woven MSAs.)**.** *A locally woven MSA is free from deadlock, orphan message, or reception error configurations. It satisfies the progress property, and when the global that generated it contains* end*, it satisfies the liveness property.*

7. Application: Data Parallel Programming Patterns and Mutualized Infrastructures

Data-parallel computations, notably the map-reduce paradigm, are mainly expressed in terms of the application of regular communication and computation patterns. Such computations are frequently performed using mutualized execution environments, such as private and public clouds.

Consider an interaction (Fig. 14a) where a client C may use different external services $\mathtt{S_e}$, $\mathtt{S'_e}$ depending on some internal choice.

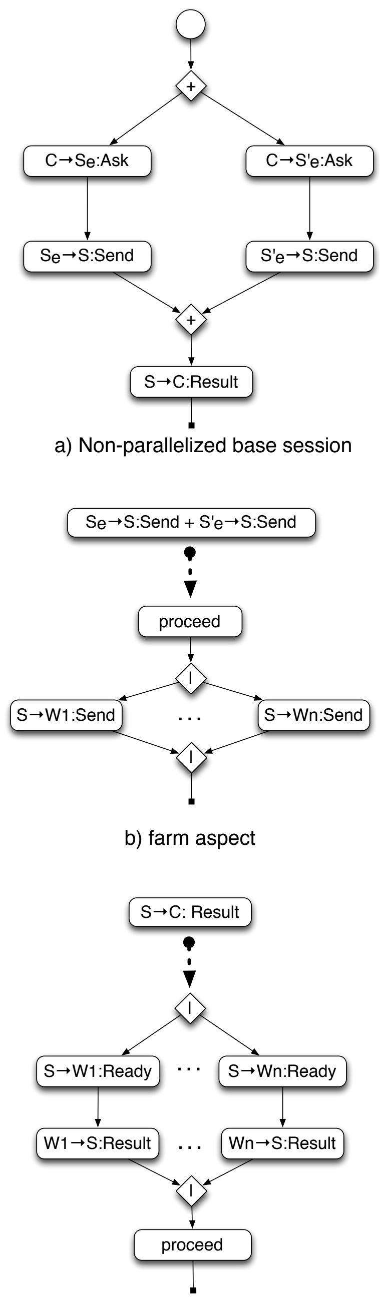

Figure 14: Managing data parallel computations

Then either of the two external service delegates the computation to the same, mutualized, service S. Finally, S sends the result to the client C. If the work can be performed by applying the same computation to small chunks of a large data set it is frequently performed in a data-parallel manner on many worker nodes. In this case, the first part of the computation (the initiation from the service S to workers Ws) can be performed using a "farm" pattern, while the second (involving the service S, the workers Ws and the client C) is termed a "gather" pattern.

Introducing such a data-parallel pattern directly in the session type greatly complicates its definition. With aspectual session types, the parallel interaction pattern is generated automatically using general-purpose aspects.

We can use the farm and gather aspect depicted in Figs 14b and c to introduce the farm/gather pattern without having to modify the base session. The farm aspect intercepts a send message to S from either the external service S_e or S'_e and transmits it to the worker nodes W1–Wn. The gather aspect intercepts the result message from S to C and informs all the worker nodes (by means of the interactions $S \to Wi$: *Ready*) and then gathers the partial results at C. Note that it is not possible to modify the gather aspect by omitting the interactions $S \to Wi$: *Ready* because the resulting gather aspect would not satisfy the aspectual local choice property. Indeed, daemons (here workers) can not initiate an advice interaction.

The naive weaving of these two aspects does not satisfy the traditional linearity condition on sessions (Section 3.3) because there are concurrent identical messages emissions on worker nodes on both branches (involving S_e or S'_e). This is because a worker node is not informed of the choice made by the client C. But with global weaving, those two branches are tagged with different names so linearity is preserved. We then need to check that concurrent invocations of a daemon worker remain separated. This is enforced by the message $S \to Wi$: *Ready* which forces the gather advice to be advice linear (Definition 3) by adding an explicit synchronization between S and Wi before the result is send.

8. Related Work

Our work is, to the best of our knowledge, the first approach investigating the benefits and limits of an integration of aspects with session types. Gay et al. [11] have presented a notion of modular session types: they aim at a seamless integration of object-oriented abstractions and the corresponding typing scheme with session types. In contrast to our work, their approach does not provide any support for the modular definition of crosscutting functionalities. Furthermore, they do not consider the extension of session types by harmless race conditions as we do.

There is also work that shares different specific features with our approach. In the following, we discuss work related to the expressivity of session types, previous application of aspects to (other) kinds of protocol, and studies on the relationship between session types and non-functional properties of interacting systems.

Expressivity of session types. Session types have been originally developed in order to precisely define typed interaction protocols between two partners [12]. Recently, their extension to multiparty protocols [8, 13] have significantly increased their expressivity. Furthermore, a kind of quantification in the form of roles has also been proposed [7]. However, none of these approaches provides the main benefits of aspectual session types: the definition of functionalities separated from a global session type in a modular way and the extension of the expressivity of session types by accepting certain interaction protocols that include race conditions.

Aspects over protocols. Several researchers have investigated the formal definition of aspects and their properties over different kinds of protocols, principally regular protocols (for instance [1, 10]). Since standard regular protocols do not obey the strong properties that session types enjoy, the integration of aspects is much simpler but may change the overall interaction structure in much less predictable ways.

There is also a significant body of work on programming extensions for protocol-like structures, notably in the context of web services and service compositions (for example [2, 6]). These approaches may modify (regular) service compositions in even more general ways and therefore do not, in general, preserve most correctness properties of the underlying protocols.

Session types and specific functionalities. Finally, some authors have explored the definition and properties over session types of functionalities that are not directly linked to the basic interaction structure. Capecchi et al. [3], for instance, have investigated security properties, notably information flow properties, over session types. Carbone et al. [4] consider the application of session types to the definition of exceptional behaviors. In contrast to our work, these approaches are specific to the functionalities they are interested in. Furthermore, they apply existing variants of session types and do not enhance their expressivity.

9. Conclusion

Session types provide clear benefits for the definition of protocol-like interactions in distributed systems, in particular, the static typability of multiparty interactions and a notion of projection that ensures the correct implementation of global interactions solely in terms of local processes.

We have proposed aspectual session types that augment the expressivity of generalized multiparty session types by allowing race conditions that are, as we show, harmless. Second, we support modular extensions of sessions, making it possible to bring the benefits of aspect orientation to the definition of distributed processes with strong communication safety properties.

As future work, we will study how to enhance quantification with wildcards on participants and dynamic conditions, and develop local aspect weaving at the level of processes.

References

[1] C. Allan, P. Avgustinov, et al. Adding trace matching with free variables to AspectJ. In Richard P. Gabriel, editor, *ACM Conference on Object-Oriented Programming, Systems and Languages (OOPSLA)*. ACM Press, 2005.

[2] F. Baligand, N. Rivierre, and T. Ledoux. A declarative approach for QoS-aware web service compositions. In *5th International Conference on Service-Oriented Computing, ICSOC 2007*, volume LNCS 4749. Springer, 2007.

[3] S. Capecchi, I. Castellani, M. Dezani-Ciancaglini, and T. Rezk. Session types for access and information flow control. In *CONCUR - Concurrency Theory, 21th International Conference*, LNCS 6269. Springer, 2010.

[4] M. Carbone, K. Honda, and N. Yoshida. Structured interactional exceptions in session types. In *CONCUR*, LNCS 5201. Springer, 2008.

[5] G. Cécé and A. Finkel. Verification of programs with half-duplex communication. *Information and Computation*, 202(2):166–190, 2005.

[6] A. Charfi and M. Mezini. Using aspects for security engineering of web service compositions. In *IEEE International Conference on Web Services, ICWS 2005*, 2005.

[7] P.-M. Deniélou and N. Yoshida. Dynamic multirole session types. In *Symposium on Principles of Programming Languages (POPL)*. ACM, 2011.

[8] P.M. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *21st European Symposium on Programming (ESOP)*, LNCS 7211, page 194. Springer, 2012.

[9] P.M. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. Technical report, Imperial College, London, 2012. http://www.doc.ic.ac.uk/ malo/msa/.

[10] R. Douence, P. Fradet, and M. Südholt. Composition, reuse and interaction analysis of stateful aspects. In *Proc. of 3rd Int. Conf. on Aspect-Oriented Software Development (AOSD)*. ACM, 2004.

[11] S. J. Gay, V. T. Vasconcelos, et al. Modular session types for distributed object-oriented programming. In *POPL'10*. ACM, 2010.

[12] K. Honda, Vasco T. Vasconcelos, and M. Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP'98*, LNCS 1381, pages 22–38. Springer, 1998.

[13] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *35th Symposium on Principles of Programming Languages (POPL)*. ACM, 2008.

$$
\begin{array}{rcl}
\mathbf{G}_{\text{Trade}} & = & \mathsf{def} \\
\mathbf{x_0} & = & \mathtt{S} \to \mathtt{B}\colon \mathit{Item}\langle \mathit{string}\rangle; \mathbf{x_1} \\
\mathbf{x_1} & = & \mathbf{x_2} \mid \mathbf{x_3} \\
\mathbf{x_2} & = & \mathtt{B} \to \mathtt{S}\colon \mathit{Sale}\langle \mathit{boolean}\rangle; \mathbf{x_4} \\
\mathbf{x_3} & = & \mathtt{B} \to \mathtt{C}\colon \mathit{Purchase}\langle \mathit{boolean}\rangle; \mathbf{x_5} \\
\mathbf{x_4} \mid \mathbf{x_5} & = & \mathbf{x}_6 \\
\mathbf{x}_6 & = & \mathsf{end} \quad \mathsf{in}\ \mathbf{x}_0
\end{array}
$$

a) Global type for the trade session

$$
\begin{array}{l}
A_{\text{nego}} = \langle pc_{\text{nego}}, ad_{\text{nego}} \rangle \\
\textit{where} \\
\quad pc_{\text{nego}} = \mathtt{S} \to \mathtt{B}\colon \mathit{Item}\langle \mathit{string}\rangle \\
\quad ad_{\text{nego}} = \mathsf{def} \\
\begin{array}{rcl}
\mathbf{x_A} & = & \mathsf{proceed}; \mathbf{x_1} \\
\mathbf{x_1} + \mathbf{x_6} & = & \mathbf{x_2} \\
\mathbf{x_2} & = & \mathbf{x_3} + \mathbf{x}_4 \\
\mathbf{x_3} & = & \mathtt{B} \to \mathtt{C}\colon \mathit{Offer}\langle \mathit{nat}\rangle; \mathbf{x_5} \\
\mathbf{x_5} & = & \mathtt{C} \to \mathtt{B}\colon \mathit{Counter}\langle \mathit{nat}\rangle; \mathbf{x_6} \\
\mathbf{x}_4 & = & \mathsf{end} \quad \mathsf{in}\ \mathbf{x}_A
\end{array}
\end{array}
$$

b) Negotiation aspect

$$
\begin{array}{l}
A_{\text{log}} = \langle pc_{\text{log}}, ad_{\text{log}} \rangle \\
\textit{where} \\
\quad pc_{\text{log}} = \mathtt{B} \to \mathtt{S}\colon * + \mathtt{B} \to \mathtt{C}\colon * \\
\quad ad_{\text{log}} = \mathsf{def} \\
\begin{array}{rcl}
\mathbf{x_A} & = & \mathsf{proceed}; \mathbf{x_1} \\
\mathbf{x_1} & = & \mathtt{B} \to \mathtt{L}\colon \mathit{LogData}\langle \mathit{string}\rangle; \mathbf{x_2} \\
\mathbf{x}_2 & = & \mathsf{end} \quad \mathsf{in}\ \mathbf{x}_A
\end{array}
\end{array}
$$

c) Logging aspect

$$
\begin{array}{l}
A_{\text{auth}} = \langle pc_{\text{auth}}, ad_{\text{auth}} \rangle \\
\textit{where} \\
\quad pc_{\text{auth}} = \mathtt{B} \to \mathtt{S}\colon * + \mathtt{B} \to \mathtt{C}\colon * \\
\quad ad_{\text{auth}} = \mathsf{def} \\
\begin{array}{rcl}
\mathbf{x_A} + \mathbf{x_5} & = & \mathbf{x_1} \\
\mathbf{x_1} & = & \mathtt{B} \to \mathtt{A}\colon \mathit{Auth}\langle \mathit{string}\rangle; \mathbf{x_2} \\
\mathbf{x_2} & = & \mathbf{x_3} + \mathbf{x_4} \\
\mathbf{x_3} & = & \mathtt{A} \to \mathtt{B}\colon \mathit{Retry}; \mathbf{x_5} \\
\mathbf{x_4} & = & \mathtt{A} \to \mathtt{B}\colon \mathit{Ok}; \mathbf{x_6} \\
\mathbf{x_6} & = & \mathsf{proceed}; \mathbf{x}_7 \\
\mathbf{x}_7 & = & \mathsf{end} \quad \mathsf{in}\ \mathbf{x}_A
\end{array}
\end{array}
$$

d) Authentication aspect

Figure 15: Formal definition of the trade example with aspects

A. Formal definition of the trade example

Figure 15 presents the formal definition of the trade example introduced in Section 2.

B. Formal definition of advice linearity

Figure 16 defines the uniqueness of messages. Figure 17 defines the computation of single threaded advice. Both definitions are expressed using auxiliary functions (respectively *Msg*, *STh*) that compute the result in their third argument and keep track of the protocol structure in the first two arguments. In Rule (STh-Join), the # operator denotes disjunction but is undefined when both arguments are true. This is to ensure that two branches that join can not both have an interaction with a daemon.

$$
\text{(Msg-Def)}\ \frac{\begin{array}{c}\mathbf{G} = \mathsf{def}\ \overline{G}\ \mathsf{in}\ \mathbf{x} \qquad A = \langle pc, \mathsf{def}\ \overline{G_A}\ \mathsf{in}\ \mathbf{x}_A \rangle \\ \overline{G}, pc, \overline{G}_A \vdash \mathit{Msg}(\mathbf{x}_A, \varepsilon, \mathsf{M})\end{array}}{\mathbf{G}, A \vdash \mathit{UniqueMsg}}
$$

$$
\text{(Msg-Message)}\ \frac{\begin{array}{c}\mathbf{x} = M; \mathbf{x}' \in \overline{G}_A \qquad M \neq \mathsf{proceed} \\ \overline{G}, pc, \overline{G}_A \vdash \mathit{Msg}(\mathbf{x}', \overline{\mathbf{x}}, \mathsf{M}')\end{array}}{\overline{G}, pc, \overline{G}_A \vdash \mathit{Msg}(\mathbf{x}, \overline{\mathbf{x}}, \{M\} \uplus \mathsf{M}')}
$$

$$
\text{(Msg-Proceed)}\ \frac{\begin{array}{c}\mathbf{x} = \mathsf{proceed}; \mathbf{x}' \in \overline{G}_A \\ \overline{G}, pc, \overline{G}_A \vdash \mathit{Msg}(\mathbf{x}', \overline{\mathbf{x}}, \mathsf{M}') \\ \mathsf{M} = \{M \mid \mathbf{x} = M; \mathbf{x}' \in \overline{G} \wedge \mathit{match}(pc, M)\}\end{array}}{\overline{G}, pc, \overline{G}_A \vdash \mathit{Msg}(\mathbf{x}, \overline{\mathbf{x}}, \mathsf{M} \uplus \mathsf{M}')}
$$

$$
\text{(Msg-Choice)}\ \frac{\begin{array}{c}\mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2 \in \overline{G}_A \qquad \mathbf{x} \notin \overline{\mathbf{x}} \\ \overline{G}, pc, \overline{G}_A \vdash \mathit{Msg}(\mathbf{x}_1, \mathbf{x}\overline{\mathbf{x}}, \mathsf{M}_1) \\ \overline{G}, pc, \overline{G}_A \vdash \mathit{Msg}(\mathbf{x}_2, \mathbf{x}\overline{\mathbf{x}}, \mathsf{M}_2)\end{array}}{\overline{G}, pc, \overline{G}_A \vdash \mathit{Msg}(\mathbf{x}, \overline{\mathbf{x}}, \mathsf{M}_1 \cup \mathsf{M}_2)}
$$

$$
\text{(Msg-Fork)}\ \frac{\begin{array}{c}\mathbf{x} = \mathbf{x}_1 \mid \mathbf{x}_2 \in \overline{G}_A \qquad \mathbf{x} \notin \overline{\mathbf{x}} \\ \overline{G}, pc, \overline{G}_A \vdash \mathit{Msg}(\mathbf{x}_1, \mathbf{x}\overline{\mathbf{x}}, \mathsf{M}_1) \\ \overline{G}, pc, \overline{G}_A \vdash \mathit{Msg}(\mathbf{x}_2, \mathbf{x}\overline{\mathbf{x}}, \mathsf{M}_2)\end{array}}{\overline{G}, pc, \overline{G}_A \vdash \mathit{Msg}(\mathbf{x}, \overline{\mathbf{x}}, \mathsf{M}_1 \uplus \mathsf{M}_2)}
$$

$$
\text{(Msg-Merge/Join)}\ \frac{\begin{array}{c}\mathbf{x}_1 \mid \mathbf{x}_2 = \mathbf{x} \in \overline{G}_A \vee \mathbf{x}_1 + \mathbf{x}_2 = \mathbf{x} \in \overline{G}_A \\ \overline{G}, pc, \overline{G}_A \vdash \mathit{Msg}(\mathbf{x}, \overline{\mathbf{x}}, \mathsf{M})\end{array}}{\overline{G}, pc, \overline{G}_A \vdash \mathit{Msg}(\mathbf{x}_i, \overline{\mathbf{x}}, \mathsf{M}) \quad i \in \{1, 2\}}
$$

$$
\text{(Msg-Choice/Fork-Stop)}\ \frac{\begin{array}{c}\mathbf{x} = \mathbf{x}_1 \mid \mathbf{x}_2 \in \overline{G}_A \vee \mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2 \in \overline{G}_A \\ \mathbf{x} \in \overline{\mathbf{x}}\end{array}}{\overline{G}, pc, \overline{G}_A \vdash \mathit{Msg}(\mathbf{x}, \overline{\mathbf{x}}, \varepsilon)}
$$

$$
\text{(Msg-End)}\ \frac{\mathbf{x} = \mathsf{end} \in \overline{G}_A}{\overline{G}, pc, \overline{G}_A \vdash \mathit{Msg}(\mathbf{x}, \overline{\mathbf{x}}, \varepsilon)}
$$

Figure 16: Unique messages

$$
\text{(STh-Def)}\ \frac{\overline{T_A}, \mathbf{x}_A, M \vdash \mathit{STh}(\mathsf{end}, \varepsilon, b)}{\mathsf{def}\ \overline{G_A}\ \mathsf{in}\ \mathbf{x}_A, M \vdash \mathit{SgTh}(b)}
$$

$$
\text{(STh-Message)}\ \frac{\mathbf{x} = ?\langle q, l\langle U\rangle\rangle; \mathbf{x}' \in \overline{T}_A \qquad \mathsf{q} \neq \mathsf{p} \text{ or } \mathsf{p}'}{\overline{T_A}, \mathbf{x}_A, M \vdash \mathit{STh}(\mathbf{x}, \overline{\mathbf{x}}, \mathbb{T})}
$$

$$
\text{(STh-Message2)}\ \frac{\begin{array}{c}\mathbf{x} = ?\langle q, l\langle U\rangle\rangle; \mathbf{x}' \in \overline{T}_A \qquad \mathsf{q} = \mathsf{p} \text{ or } \mathsf{p}' \\ \overline{T}_A, \mathbf{x}_A, \mathsf{p} \to \mathsf{p}'\colon l\langle U\rangle \vdash \mathit{STh}(\mathbf{x}', \overline{\mathbf{x}}, b)\end{array}}{\overline{T_A}, \mathbf{x}_A, M \vdash \mathit{STh}(\mathbf{x}, \overline{\mathbf{x}}, b)}
$$

$$
\text{(STh-Merge)}\ \frac{\begin{array}{c}\mathbf{x}_1 + \mathbf{x}_2 = \mathbf{x} \in \overline{T}_A \qquad \mathbf{x} \notin \overline{\mathbf{x}} \\ \overline{T_A}, \mathbf{x}_A, M \vdash \mathit{STh}(\mathbf{x}_1, \mathbf{x}\overline{\mathbf{x}}, b_1) \\ \overline{T_A}, \mathbf{x}_A, M \vdash \mathit{STh}(\mathbf{x}_2, \mathbf{x}\overline{\mathbf{x}}, b_2)\end{array}}{\overline{T_A}, \mathbf{x}_A, M \vdash \mathit{STh}(\mathbf{x}, \overline{\mathbf{x}}, b_1 \vee b_2)}
$$

$$
\text{(STh-Join)}\ \frac{\begin{array}{c}\mathbf{x}_1 \mid \mathbf{x}_2 = \mathbf{x} \in \overline{T}_A \qquad \mathbf{x} \notin \overline{\mathbf{x}} \\ \overline{T_A}, \mathbf{x}_A, M \vdash \mathit{STh}(\mathbf{x}_1, \mathbf{x}\overline{\mathbf{x}}, b_1) \\ \overline{T_A}, \mathbf{x}_A, M \vdash \mathit{STh}(\mathbf{x}_2, \mathbf{x}\overline{\mathbf{x}}, b_2)\end{array}}{\overline{T_A}, \mathbf{x}_A, M \vdash \mathit{STh}(\mathbf{x}, \overline{\mathbf{x}}, b_1 \mathbin{\#} b_2)}
$$

$$
\text{(STh-Fork/Choice)}\ \frac{\begin{array}{c}\mathbf{x} = \mathbf{x}_1 \mid \mathbf{x}_2 \in \overline{T}_A \vee \mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2 \in \overline{T}_A \\ \overline{T_A}, \mathbf{x}_A, M \vdash \mathit{STh}(\mathbf{x}, \overline{\mathbf{x}}, b)\end{array}}{\overline{T_A}, \mathbf{x}_A, M \vdash \mathit{STh}(\mathbf{x}_i, \overline{\mathbf{x}}, b) \quad i \in \{1, 2\}}
$$

$$
\text{(STh-Merge/Join-Stop)}\ \frac{\mathbf{x}_1 \mid \mathbf{x}_2 = \mathbf{x} \in \overline{T}_A \vee \mathbf{x}_1 + \mathbf{x}_2 = \mathbf{x} \in \overline{T}_A \qquad \mathbf{x} \in \overline{\mathbf{x}}}{\overline{T_A}, \mathbf{x}_A, M \vdash \mathit{STh}(\mathbf{x}, \overline{\mathbf{x}}, \mathbb{F})}
$$

$$
\text{(STh-Init)}\ \frac{}{\overline{T_A}, \mathbf{x}_A, M \vdash \mathit{STh}(\mathbf{x}_A, \overline{\mathbf{x}}, \mathbb{F})}
$$

$$
\text{(STh-End)}\ \frac{\mathbf{x} = \mathsf{end} \in \overline{T}_A \qquad \overline{T_A}, \mathbf{x}_A, M \vdash \mathit{STh}(\mathbf{x}, \overline{\mathbf{x}}, b)}{\overline{T_A}, \mathbf{x}_A, M \vdash \mathit{STh}(\mathsf{end}, \overline{\mathbf{x}}, b)}
$$

Figure 17: Check for single-threadedness