Safe and live multiparty sessions as computational effects

Dylan McDermott
University of Oxford
United Kingdom
dylan@dylanm.org

Nobuko Yoshida
University of Oxford
United Kingdom
nobuko.yoshida@cs.ox.ac.uk

***Abstract*—We explore computational effects and denotational semantics of synchronous and asynchronous multiparty session communications, building a typed call-by-value λ-calculus, dubbed as $\lambda_{\texttt{SafeMP}}$, with concurrent message passing. The type system of $\lambda_{\texttt{SafeMP}}$ enforces not only *communication safety* (no type and label error), but also *deadlock-freedom* (communication never gets stuck). The key feature of our theory is tracking the computational effect of sending and receiving messages among multiple participants, regarding multiparty session types as *grades* of computations. We justify the close connection between sessions and effects by developing a denotational semantics for $\lambda_{\texttt{SafeMP}}$, involving a graded monad. While our calculus with general recursion is expressive, we can ensure *liveness* (communication eventually happens), in both synchronous and asynchronous semantics, by introducing a simple effect system which controls recursive behaviours. To our best knowledge, our work is the first which develops extensional denotational semantics for MPST with a guarantee of safety, deadlock-freedom and liveness, under both synchronous and asynchronous communications.**

***Index Terms*—computational effects, message passing, multiparty session types, denotational semantics, graded monad.**

I. Introduction

Multiparty session types (MPST) provide a typing discipline ensuring that multiple participants communicating via message-passing conform to a *multiparty protocol*, satisfying desired safety properties such as (1) **communication safety** (aka *type-safety*) (no participant will receive a message with a value of an unexpected type or unexpected label); and (2) **deadlock-freedom** (if a participant `p` wishes to receive a message from or send a message to a participant `q` then, unless the computation diverges, `q` will eventually send or receive that message). Deadlock-freedom implies that `p` and `q` cannot both be blocked waiting for messages from each other (wait-free), but it does not require that `q` will eventually communicate with `p` if the computation diverges. It is often desirable to prevent participants from being starved in this way, and thus, instead of settling for deadlock-freedom, to require the stronger property, **liveness** (aka *progress* [1]–[3]) (if one participant wishes to communicate with another, then that communication will *eventually* happen following the protocol).

The theory of MPST can guarantee those properties for multiple participants either by taking the *top-down approach* [4]–[6] (which uses a global type as a guidance) or *bottom-up approach* [7], [8] (which model-checks a set of local types to ensure a desired property). A vast amount of session type theories were proposed last two decades, but one simple question has been left open: while several semantic studies have been made for binary (2-party) session types based on, e.g., logical relations [9] and game semantics [10], to our best knowledge, no extensional denotational semantics for MPST which are derived from an expressive language (calculus) has been proposed yet (see § VIII). Our challenge is to find a simple but most effective way to construct semantics systematically, built on a computational calculus which subsumes core features of both synchronous and asynchronous MPST.

To meet this challenge, we start with a simple call-by-value λ-calculus, extended with constructs for message passing and parallel composition–an idealised programming language with message-passing concurrency. This calculus, by default, does not satisfy any of the three properties listed above.

We take a principled approach to designing our type system, by following a few key insights: firstly, in the context of a call-by-value semantics, sending and receiving a message can be seen as *computational effects*, analogous to mutating state or raising an exception. For enforcing deadlock-freedom, it therefore needs to track how computational effects are used. Recent work [11], [12] has established *grading* as the key technique of tracking computational effects compositionally, and thus we use a graded type system (aka an *effect system*).

A graded type system assigns both a type and a *grade* to each computation. The grade describes an abstract aspect of the behaviour of the computation. One of the main steps in describing a graded type system is choosing the set of grades. Our goal is to set grades to contain sufficient information about sending and receiving actions in order to ensure safety and liveness properties. Here we use the information associated with MPST. We call the resulting typed calculus $\lambda_{\texttt{SafeMP}}$.

Integrating session types into a call-by-value λ-calculus with effects enables us to study session types from the perspective of computational effects, and in particular, to apply techniques from the computational effects literature to session types. We demonstrate this by developing a denotational semantics for $\lambda_{\texttt{SafeMP}}$, using techniques for modelling computational effects. Our denotational semantics is based on a *graded monad* [11], [13]–[15], which are the standard tool for modelling computations in a graded type system.

In this paper, we take the top-down approach which starts specifying a whole view of communications as a *global*

protocol, and each concurrent program is locally type-checked against its end-point projection. We investigate two semantics, **synchronous** and **asynchronous** semantics in $\lambda_{\texttt{SafeMP}}$. In synchronous semantics, sending a message blocks until that message is received; but in asynchronous, messaging is non-blocking but ordered through FIFO queues; the participant that sends the message can continue without waiting for the message to be received. Asynchronous message passing is more practical, but also more difficult to reason about. We show how to build asynchronous semantics in Section VII.

One of the technical challenges is ensuring **liveness** in the presence of *recursive behaviours*. We include general recursive functions in $\lambda_{\texttt{SafeMP}}$, so that computations can implement recursive protocols. As a consequence, $\lambda_{\texttt{SafeMP}}$ contains diverging terms that do not involve sending or receiving messages. This contrasts with the π-calculus, in which one can emulate recursive functions (the π-calculus is Turing-powerful), but only by repeatedly sending messages. Expressiveness by recursions in $\lambda_{\texttt{SafeMP}}$ affects liveness: in $\lambda_{\texttt{SafeMP}}$, one can starve a participant by simply diverging, and never sending a message. Due to this feature, session types alone are not enough to enforce liveness in the presence of recursion. We show how to rectify this in Section VI, using an extended effect system that control recursion.

Contributions and outline. This paper establishes the connection between MPST and computational effects, and uses this perspective to enforce safety and liveness properties in a call-by-value λ-calculus. **Section II** introduces a (untyped) concurrent λ-calculus with message passing and its (synchronous) semantics. **Section III** interprets multiparty session types as grades, and introduces the notion of *conformance* to $\lambda_{\texttt{SafeMP}}$. **Section IV** introduces our graded type system, and shows that it enforces **communication safety** and **deadlock-freedom**. **Section V** introduces a denotational semantics for our graded system, and hence a denotational interpretation of multiparty session types. We use the semantics to show that $\lambda_{\texttt{SafeMP}}$ has **no observable nondeterminism** (if some execution of a computation produces some result, then every execution of the computation produces that same result). **Section VI** shows how to modify our graded type system so that it also enforces **liveness**. **Section VII** introduces an asynchronous semantics for $\lambda_{\texttt{SafeMP}}$, and adapts our deadlock-freedom and liveness results to the asynchronous setting. The proofs are left to the appendix.

II. Syntax and operational semantics of $\lambda_{\texttt{SafeMP}}$

In this section, we introduce the syntax and operational semantics of our calculus $\lambda_{\texttt{SafeMP}}$. We base our calculus on Levy's *fine-grain call-by-value* [16] which has a simple operational semantics, while still being powerful enough to embed e.g. Plotkin's call-by-value calculus [17]. The calculus is extended with message sending and receiving, and parallel composition. We also give the examples we use throughout the paper, and discuss them with respect to the three desiderata listed in the introduction.

Computations. *Terms* are stratified into: *values* v, w and *computations*[1] t, u. $\mathsf{p}, \mathsf{q}, \mathsf{r}, \ldots$ range over a fixed set of *participants* $\mathcal{P}$ and $\ell, \ell', \ldots$ range over a fixed set of *labels* $\mathcal{L}$.

The following is the grammar used to generate the syntax.

$$
\begin{aligned}
v, w &::= x \mid () \mid n \mid \textbf{true} \mid \textbf{false} \mid \textbf{rec}\ f\, x.\, t \\
t, u &::= \textbf{return}\ v \mid \textbf{let}\ x = t\ \textbf{in}\ u \\
&\quad \mid v - w \mid v < w \mid \textbf{if}\ v\ \textbf{then}\ t_1\ \textbf{else}\ t_2 \mid w\, v \\
&\quad \mid \textbf{send}\,(\ell, v)\ \textbf{to}\ \mathsf{p}\ \textbf{then}\ t \mid \textbf{recv}\ \mathsf{p}\,\{(\ell_i, x_i).\, t_i\}_{i \in I} \\
&\quad \mid \textbf{let}\ x_1 = \mathsf{r}_1 \lhd t_1 \parallel \cdots \parallel x_n = \mathsf{r}_n \lhd t_n\ \textbf{in}\ u
\end{aligned}
$$

Values include variables x, constants (unit $()$, integer n and the booleans **true** and **false**), and anonymous recursive functions **rec** $f\, x.\, t$. A *ground value* is a value v that is not a variable, and does not have the form **rec** $f\, x.\, t$. In the latter, the variable f stands for the recursive function, so that applying f to an argument makes a recursive call. The value **rec** $f\, x.\, t$ is itself unnamed. We recover ordinary non-recursive λ-abstractions by defining

$$\lambda x.\, t = \textbf{rec}\ f\, x.\, t \text{ where } f \text{ is not free in } t$$

For computations, we explain the constructs above along with the (synchronous) *reduction relation* $t \overset{\alpha}{\rightsquigarrow} t'$, which is defined in Figure 1. First we explain the notions of *action* we label our reductions with. *Local actions* α and *global actions* β are generated by the following, where we require v to be a ground value.

$$
\begin{aligned}
\alpha &::= \tau \mid \mathsf{p}!(\ell, v) \mid \mathsf{p}?(\ell, v) \\
\beta &::= \tau \mid \mathsf{p} \to \mathsf{q} : (\ell, v) \qquad \text{where } \mathsf{p} \neq \mathsf{q}
\end{aligned}
$$

We restrict message payloads to ground values to ensure that payloards do not contain participant names as values (following the standard MPST). We will also write $t \overset{\alpha}{\rightsquigarrow}{}^{*}\ t'$ for a finite sequence of reductions ending in action α, where all the reductions before the last have action τ. For $\alpha = \tau$ we permit the sequence of reductions to be empty (with $t = t'$), while for any other action we require there to be at least one reduction. A global action β can be *projected* onto a set of participants R, the result being a local action $\beta \upharpoonright \mathsf{R}$; this is defined by $\tau \upharpoonright \mathsf{R} = \tau$ and

$$
(\mathsf{p} \to \mathsf{q} : (\ell, v)) \upharpoonright \mathsf{R} = \begin{cases} \mathsf{q}!(\ell, v) & \text{if } \mathsf{p} \notin \mathsf{R} \wedge \mathsf{q} \in \mathsf{R} \\ \mathsf{p}?(\ell, v) & \text{if } \mathsf{p} \in \mathsf{R} \wedge \mathsf{q} \notin \mathsf{R} \\ \tau & \text{otherwise} \end{cases}
$$

The first part of the grammar of computations is the core of fine-grain call-by-value: a computation can *return* a value v, or it can be a sequence **let** $x = t$ **in** u of computations. This first evaluates t, then, if t returns a result, evaluates u with x bound to the result of t, by rules [LetR] and [Let]. Rules [IfT], [IfF], [Sub] and [Eq] are standard. All of these operate on values; thus to e.g. compare two computations that return integers, it is necessary to explicitly use **let** to evaluate the operands.

The computation **send** (ℓ, v) **to** p **then** t sends a message (ℓ, v) to a participant p, and then continues as the computation t. Participants $\mathsf{p}, \mathsf{q}, \mathsf{r}, \ldots$ are analogous to variable names x;

[1] Levy et al. [16] call these *producers*.

they are bound by the parallel composition construct explained below. The payload is required to be a value; to use the result of a computation as the payload, one again has to use **let**.

The computation **recv** $\mathsf{p}\,\{(\ell_1, x_1).\, t_1, \ldots, (\ell_n, x_n).\, t_n\}$ receives one message from the participant p; if ℓ_i is selected, it continues as t_i, with x_i bound to the message payload. We require the message labels ℓ_i to be distinct from each other, and we require $n > 0$. We typically write **recv** $\mathsf{p}\,\{(\ell_i, x_i).\, t_i\}_{i \in I}$ where I is a finite nonempty set, but we do not consider I to be part of the syntax.

The final **let** construct, evaluates the computations $t_1, \ldots, t_n$ in parallel (rule [Par]), and, once all of them return values ([TRet]), evaluates u with the variables x_i bound to the results. This is the parallel composition construct we include in $\lambda_{\texttt{SafeMP}}$, and it is the only construct that introduces new participant names; for each i, the participant names $\mathsf{r}_1, \ldots, \mathsf{r}_n$ are bound in the computation t_i. Thus t_i, acting as participant r_i, can send a message to or receive a message from another participant r_j. We will refer to the computations $t_1, \ldots, t_n$, running in parallel, as a (multiparty) session; in general a session has the following form.

$$\mathcal{M} \;::=\; \mathsf{r}_1 \lhd t_1 \parallel \mathsf{r}_2 \lhd t_2 \parallel \cdots \parallel \mathsf{r}_n \lhd t_n$$

As a special case of our parallel let construct, we can *run* a session, ignoring its results:

$$\begin{aligned}&\textbf{run}\,(\mathsf{r}_1 \lhd t_1 \parallel \cdots \parallel \mathsf{r}_n \lhd t_n)\\ &= \textbf{let}\;\; x_1 = \mathsf{r}_1 \lhd t_1 \parallel \cdots \parallel x_n = \mathsf{r}_n \lhd t_n \;\;\textbf{in}\;\; \textbf{return}\,()\end{aligned}$$

Examples. We use the following running examples.

Example II.1 (Two buyers) Our first example is a variant of a classic example from the MPST literature [18]. There are three participants, a store s, and two buyers b_1 and b_2. Buyer b_1 sends a `query` to s, asking for the `price` of an item. Once s sends the price to b_1, b_1 then asks b_2 to contribute some amount to the cost, by sending a `price` (presumably no more than the price b_1 received). Then b_2 sends a `yes` or `no` response to b_1, and b_1 tells s whether it wishes to `buy` or `cancel`.

For instance, in the following s gives the price 10 for every item; b_1 requests the price for item 7 and asks b_2 to contribute at most 4 towards the cost, and b_1 agrees to any price.

$$\begin{aligned}t_{\mathsf{s}} = {}&\textbf{recv}\;\mathsf{b}_1\,\{(\texttt{query}, x).\\ &\quad\textbf{send}\,(\texttt{price}, 10)\;\textbf{to}\;\mathsf{b}_1\;\textbf{then}\\ &\quad\textbf{recv}\;\mathsf{b}_1\,\{\begin{array}{l}(\texttt{buy}, y).\;\textbf{return true},\\ (\texttt{cancel}, y).\;\textbf{return false}\end{array}\}\}\end{aligned}$$

$$\begin{aligned}t_{\mathsf{b}_1} = {}&\textbf{send}\,(\texttt{query}, 7)\;\textbf{to}\;\mathsf{s}\;\textbf{then}\\ &\textbf{recv}\;\mathsf{s}\,\{(\texttt{price}, x).\\ &\textbf{let}\;\; y = 3 < x\;\;\textbf{in}\\ &\textbf{let}\;\; z = \textbf{if}\;\; y\;\;\textbf{then return}\;4\;\;\textbf{else return}\;x\;\;\textbf{in}\\ &\textbf{send}\,(\texttt{price}, z)\;\textbf{to}\;\mathsf{b}_2\;\textbf{then}\\ &\textbf{recv}\;\mathsf{b}_2\,\{\begin{array}{l}(\texttt{yes}, w).\;\textbf{send}\,(\texttt{buy}, ())\;\textbf{to}\;\mathsf{s}\;\textbf{then return}\,()\\ (\texttt{no}, w).\;\textbf{send}\,(\texttt{cancel}, ())\;\textbf{to}\;\mathsf{s}\;\textbf{then return}\,()\end{array}\}\}\end{aligned}$$

$$t_{\mathsf{b}_2} = \textbf{recv}\;\mathsf{b}_1\,\{(\texttt{price}, z).\,\textbf{send}\,(\texttt{yes}, ())\;\textbf{to}\;\mathsf{b}_1\;\textbf{then return}\;z\}$$

The following computation is safe, deadlock-free, and live.

$$\begin{aligned}&\textbf{let}\;\; \mathsf{s} = \mathsf{x} \lhd t_{\mathsf{s}} \parallel \mathsf{b}_1 = \mathsf{y}_1 \lhd t_{\mathsf{b}_1} \parallel \mathsf{b}_2 = \mathsf{y}_2 \lhd t_{\mathsf{b}_2}\;\;\textbf{in}\;\; \textbf{return}\;y_2\\ &\overset{\tau}{\rightsquigarrow}{}^{*}\;\textbf{return}\;4\end{aligned}$$

Example II.2 (State with two participants) The second example shows that we can emulate other computational effects by using message passing, in a manner reminiscent of *effect handlers* [19]. Specifically, we focus on state, where the state is a single integer.

Initially, we consider two participants `state` and `consumer`:

$$\textbf{run}\,(\mathsf{state} \lhd t_{\mathsf{state}} \parallel \mathsf{consumer} \lhd t_{\mathsf{consumer}})$$

The participant `state` keeps track of the current state, while `consumer` asks `state` either (`get`) to send the current value (`st`), or to update the state (`put`). For instance, one possible implementation of the state is the following computation.

$$\begin{aligned}t_{\mathsf{state}} = (&\textbf{rec}\;f\,x.\;\textbf{recv}\;\mathsf{consumer}\,\{\\ &(\texttt{get}, y).\;\textbf{send}\,(\texttt{st}, x)\;\textbf{to}\;\mathsf{consumer}\;\textbf{then}\;f\,x,\\ &(\texttt{put}, y).\,f\,y\}\,0\end{aligned}$$

Here the variable x tracks the current value of the state, which is initialized to 0, and changes to the value sent by `consumer` in the recursive call in the `put` case. A basic example of t_{consumer} is the computation $\texttt{decr}\,1$, which repeatedly decreases the state by 1.

$$\begin{aligned}\texttt{decr} = \textbf{rec}\;f\,x.\;\;&\textbf{send}\,(\texttt{get}, ())\;\textbf{to}\;\mathsf{state}\;\textbf{then}\\ &\textbf{recv}\;\mathsf{state}\,\{(\texttt{st}, y).\\ &\textbf{let}\;\; z = y - x\;\;\textbf{in}\\ &\textbf{send}\,(\texttt{put}, z)\;\textbf{to}\;\mathsf{state}\;\textbf{then}\\ &f\,x\}\end{aligned}$$

Here x is the amount to decrease the state by, and y is the value received from `state`. If we run this in parallel with t_{state}, none of our desiderata from the introduction are violated: there are no issues with unexpected messages, nor with deadlocks, nor with liveness (the computation would not terminate, but there are no starved participants).

Example II.3 (State with multiple participants) We make our state example more interesting by noting that, since we allow parallel compositions to be nested, `consumer` can actually be multiple computations running in parallel:

$$t_{\mathsf{consumer}} = \textbf{run}\,(\mathsf{c}_1 \lhd t_{\mathsf{c}_1} \parallel \mathsf{c}_2 \lhd t_{\mathsf{c}_2})$$

(Here we consider two, but we could have more.) The computations t_{c_i} have four participants in scope, namely `state`, `consumer`, c_1, and c_2. In particular, they can communicate with each other, and they share access to the state.

Suppose that we define $t_{c_1} = \texttt{decr}\,1$ and $t_{c_2} = \texttt{decr}\,2$. Every time `state` sends a `st` to `consumer`, this message will go to one of c_1 or c_2 (using the rule [SPar]), whichever one is waiting for the result.[2] Note however that the two computations can race for access to the state. In $\lambda_{\texttt{SafeMP}}$, we do not permit such races, and in this case t_{consumer} is not typable using the rules we give in Section IV.

To give an example that is typable, we instead ensure that only one of c_1 and c_2 is actively communicating with `state` by making each of them wait for a `yield` message from the

[2]Since the reduction relation defined above is for synchronous message passing, only one of the two will be waiting for a result at each point in the reduction.

Computation reduction $\boxed{t \overset{\alpha}{\rightsquigarrow} t'}$

$$[\textsc{LetR}]\ \frac{}{\textbf{let}\ x = \textbf{return}\ v\ \textbf{in}\ u \overset{\tau}{\rightsquigarrow} u[x \mapsto v]} \qquad [\textsc{Let}]\ \frac{t \overset{\alpha}{\rightsquigarrow} t'}{\textbf{let}\ x = t\ \textbf{in}\ u \overset{\alpha}{\rightsquigarrow} \textbf{let}\ x = t'\ \textbf{in}\ u} \qquad [\textsc{IfT}]\ \frac{}{\textbf{if}\ \textbf{true}\ \textbf{then}\ t_1\ \textbf{else}\ t_2 \overset{\tau}{\rightsquigarrow} t_1}$$

$$[\textsc{Sub}]\ \frac{}{(n_1 - n_2) \overset{\tau}{\rightsquigarrow} \textbf{return}\ m}\ \text{where } m = n_1 - n_2 \qquad [\textsc{Eq}]\ \frac{}{(n_1 < n_2) \overset{\tau}{\rightsquigarrow} \textbf{return}\ b}\ \text{where } b = \begin{cases} \textbf{true} & \text{if } n_1 < n_2 \\ \textbf{false} & \text{otherwise} \end{cases}$$

$$[\textsc{Rec}]\ \frac{}{(\textbf{rec}\ f\, x.\, t)\, v \overset{\tau}{\rightsquigarrow} t[f \mapsto (\textbf{rec}\ f\, x.\, t), x \mapsto v]}$$

$$[\textsc{Send}]\ \frac{}{\textbf{send}\,(\ell, v)\ \textbf{to}\ \mathsf{p}\ \textbf{then}\ t \overset{\mathsf{p}!(\ell,v)}{\rightsquigarrow} t} \qquad [\textsc{Recv}]\ \frac{}{\textbf{recv}\ \mathsf{p}\,\{(\ell_i, x_i).\, t_i\}_{i\in I} \overset{\mathsf{p}?(\ell_j,v)}{\rightsquigarrow} t_j[x_j \mapsto v]}$$

$$[\textsc{TRet}]\ \frac{}{\textbf{let}\ x_1 = \mathsf{r}_1 \lhd \textbf{return}\ v_1 \parallel \cdots \parallel x_n = \mathsf{r}_n \lhd \textbf{return}\ v_n\ \textbf{in}\ u \overset{\tau}{\rightsquigarrow} u[x_i \mapsto v_i]_i}$$

$$[\textsc{Par}]\ \frac{(\mathsf{r}_1 \lhd t_1 \parallel \cdots \parallel \mathsf{r}_n \lhd t_n) \overset{\beta}{\rightsquigarrow} (\mathsf{r}_1 \lhd t'_1 \parallel \cdots \parallel \mathsf{r}_n \lhd t'_n)}{\textbf{let}\ x_1 = \mathsf{r}_1 \lhd t_1 \parallel \cdots \parallel x_n = \mathsf{r}_n \lhd t_n\ \textbf{in}\ u \overset{\beta \upharpoonright \{\mathsf{r}_1,\ldots,\mathsf{r}_n\}}{\rightsquigarrow} \textbf{let}\ x_1 = \mathsf{r}_1 \lhd t'_1 \parallel \cdots \parallel x_n = \mathsf{r}_n \lhd t'_n\ \textbf{in}\ u}$$

Session reduction $\boxed{\mathcal{M} \overset{\beta}{\rightsquigarrow} \mathcal{M}'}$

$$[\textsc{Com}]\ \frac{t_j \overset{\mathsf{r}_k!(\ell,v)}{\rightsquigarrow} t'_j \qquad t_k \overset{\mathsf{r}_j?(\ell,v)}{\rightsquigarrow} t'_k \qquad j \neq k}{(\mathsf{r}_1 \lhd t_1 \parallel \cdots \parallel \mathsf{r}_n \lhd t_n) \overset{\mathsf{r}_j \to \mathsf{r}_k:(\ell,v)}{\rightsquigarrow} (\mathsf{r}_1 \lhd t'_1 \parallel \cdots \parallel \mathsf{r}_n \lhd t'_n)}\ \text{where } t'_i = t_i \text{ for all } i \notin \{j, k\}$$

$$[\textsc{SPar}]\ \frac{t_j \overset{\alpha}{\rightsquigarrow} t'_j \qquad \alpha \text{ does not contain any } \mathsf{r}_i}{(\mathsf{r}_1 \lhd t_1 \parallel \cdots \parallel \mathsf{r}_n \lhd t_n) \overset{\beta}{\rightsquigarrow} (\mathsf{r}_1 \lhd t'_1 \parallel \cdots \parallel \mathsf{r}_n \lhd t'_n)}\ \text{where } t'_i = t_i \text{ for all } i \neq j \text{ and } \beta = \begin{cases} \tau & \text{if } \alpha = \tau \\ \mathsf{r}_j \to \mathsf{p} : (\ell, v) & \text{if } \alpha = \mathsf{p}!(\ell, v) \\ \mathsf{p} \to \mathsf{r}_j : (\ell, v) & \text{if } \alpha = \mathsf{p}?(\ell, v) \end{cases}$$

Fig. 1. Operational semantics of $\lambda_{\texttt{SafeMP}}$. We omit [IfF].

other before accessing the state.

$$\begin{aligned}
\texttt{waitForYield}_{\mathsf{p}} &= \textbf{rec}\ f\, x.\, \textbf{recv}\ \mathsf{p}\,\{(\texttt{noYield}, y).\, f\, x,\\
&\qquad\qquad (\texttt{yield}, y).\, \textbf{return}\,()\}\\
\texttt{decrYield}_{\mathsf{p}} &= \textbf{rec}\ f\, x.\ \ \textbf{let}\ y = \texttt{waitForYield}_{\mathsf{p}}\,()\ \textbf{in}\\
&\qquad\qquad \textbf{send}\,(\texttt{get}, ())\ \textbf{to}\ \mathsf{state}\ \textbf{then}\\
&\qquad\qquad \textbf{recv}\ \mathsf{state}\,\{(\texttt{st}, y).\\
&\qquad\qquad \textbf{send}\,(\texttt{noYield}, ())\ \textbf{to}\ \mathsf{p}\ \textbf{then}\\
&\qquad\qquad \textbf{let}\ z = y - x\ \textbf{in}\\
&\qquad\qquad \textbf{send}\,(\texttt{put}, z)\ \textbf{to}\ \mathsf{state}\ \textbf{then}\\
&\qquad\qquad \textbf{send}\,(\texttt{yield}, ())\ \textbf{to}\ \mathsf{p}\ \textbf{then}\\
&\qquad\qquad f\, x\}\\
t_{\mathsf{c}_1} &= \texttt{decrYield}_{\mathsf{c}_2}\, 1\\
t_{\mathsf{c}_2} &= \textbf{send}\,(\texttt{yield}, ())\ \textbf{to}\ \mathsf{c}_1\ \textbf{then}\ \texttt{decrYield}_{\mathsf{c}_1}\, 2
\end{aligned}$$

Here each c_i decreases the state once before passing control to the other, so the state will decrease by 1, then 2, then 1, and so on. There are no races, and again communication is safe, deadlock-free, and live. We discuss why this example is typable below.

III. Grades, session types, and conformance

A. Local types

As explained in the introduction, we give a graded type system for $\lambda_{\texttt{SafeMP}}$, in which we assign a *grade* g to each computation (along with the type). The grades here are the closed (local) *session types* used in the MPST literature; each of these expresses the behaviour of a single participant.

Session types $(g, h, ...)$, possibly with free session type variables $\mathbf{t}$, are generated as follows.

$$\mathsf{b} \ ::= \ \textbf{unit} \mid \textbf{bool} \mid \textbf{int}$$

$$g \ ::= \ \textbf{end} \mid \oplus_{i\in I}\, \mathsf{p}!\ell_i\langle \mathsf{b}_i\rangle.\, g_i \mid \&_{i\in I}\mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, g_i \mid \mathbf{t} \mid \mu\mathbf{t}.\, g$$

Above b ranges over *ground types*, which specify the type of the payloads of messages. We write $v : \mathsf{b}$ to indicate that ground value v has ground type b. A session type is *closed* when there are no free session type variables $\mathbf{t}$; the *recursive* session type $\mu\mathbf{t}.\, g$ binds $\mathbf{t}$ in g. We require recursion to be *guarded*, meaning that g in above cannot be **end** or a type variable. The meaning of each closed session type g, ignoring the possibility of divergence, is as: **end** returns a result,

without sending or receiving any messages; $\oplus_{i\in I}\mathsf{p}!\ell_i\langle \mathsf{b}_i\rangle.\, g_i$ sends one message to p, with label ℓ_i and payload of type b_i, and then continues as described in g_i; $\&_{i\in I}\mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, g_i$ receives one message from p, and if that message has label ℓ_i and payload of type b_i, then continues as described in g_i; and $\mu\mathbf{t}.\, g$ has the same meaning as $g[\mathbf{t}\mapsto \mu\mathbf{t}.\, g]$.

Until Section VI, we do not track the possibility of divergence, so if a computation t has grade g, it may diverge instead of interacting as described above. (This is different from the rest of the MPST literature.)

Similar syntactic restrictions to those for **recv** computations apply to both $\oplus$ and $\&$, in particular, we require the labels ℓ_i to be distinct, and we require I to be finite and non-empty. When I is a singleton, we write $\mathsf{p}!\ell\langle \mathsf{b}\rangle.\, g$ and $\mathsf{p}?\ell\langle \mathsf{b}\rangle.\, g$.

We use the standard subtyping relation $<:$ for closed session types [6] where $\oplus$ is covariant and $\&$ is contravariant. This preorder permits the set of labels appearing in send and receive types to be changed. It is defined coinductively by the following rules.

$$[\textsc{LeqSend}]\ \frac{I\subseteq J \qquad \{g_i <: h_i\}_{i\in I}}{\oplus_{i\in I}\mathsf{p}!\ell_i\langle \mathsf{b}_i\rangle.\, g_i <: \oplus_{j\in J}\mathsf{p}!\ell_j\langle \mathsf{b}_j\rangle.\, h_j}$$

$$[\textsc{LeqRecv}]\ \frac{J\subseteq I \qquad \{g_j <: h_j\}_{j\in J}}{\&_{i\in I}\mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, g_i <: \&_{j\in J}\mathsf{p}?\ell_j\langle \mathsf{b}_j\rangle.\, h_j}$$

$$[\textsc{End}]\quad \mathbf{end} <: \mathbf{end}$$

$$[\mu\mathrm{L}]\ \frac{g[\mathbf{t}\mapsto\mu\mathbf{t}.\, g] <: h}{\mu\mathbf{t}.\, g <: h} \qquad [\mu\mathrm{R}]\ \frac{g <: h[\mathbf{t}\mapsto\mu\mathbf{t}.\, h]}{g <: \mu\mathbf{t}.\, h}$$

To give a graded type system, we actually need an ordered *monoid* of grades. The multiplication $g\cdot g'$ of two grades is the grade of a computation that first does g, and then does g'. In our case, we define it as follows.

$$(\oplus_{i\in I}\mathsf{p}!\ell_i\langle \mathsf{b}_i\rangle.\, g_i)\cdot h = \oplus_{i\in I}\mathsf{p}!\ell_i\langle \mathsf{b}_i\rangle.\, (g_i\cdot h)$$
$$(\&_{i\in I}\mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, g_i)\cdot h = \&_{i\in I}\mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, (g_i\cdot h)$$
$$\mathbf{end}\cdot h = h \qquad \mathbf{t}\cdot h = \mathbf{t} \qquad (\mu\mathbf{t}.\, g)\cdot h = \mu\mathbf{t}.\, (g\cdot h)$$

This does indeed form an ordered monoid: the multiplication operation is monotone with respect to $<:$, is associative, and has **end** as the unit.

To state our results, we also use a notion of *reduction* $g \overset{\alpha}{\rightsquigarrow} g'$ for grades, analogous to reduction of computations, defined inductively by the following rules.

$$\frac{v : \mathsf{b}_j}{\oplus_{i\in I}\mathsf{p}!\ell_i\langle \mathsf{b}_i\rangle.\, g_i \overset{\mathsf{p}!(\ell_j,v)}{\rightsquigarrow} g_j} \qquad \frac{v : \mathsf{b}_j}{\&_{i\in I}\mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, g_i \overset{\mathsf{p}?(\ell_j,v)}{\rightsquigarrow} g_j}$$

$$\frac{}{g \overset{\tau}{\rightsquigarrow} g} \qquad \frac{g[\mathbf{t}\mapsto\mu\mathbf{t}.\, g] \overset{\alpha}{\rightsquigarrow} g'}{\mu\mathbf{t}.\, g \overset{\alpha}{\rightsquigarrow} g'}$$

We write $g \overset{\mathsf{p}!}{\rightsquigarrow}$ when there is some (ℓ, v) and g' such that $g \overset{\mathsf{p}!(\ell,v)}{\rightsquigarrow} g'$; and similarly for $g \overset{\mathsf{p}?}{\rightsquigarrow}$.

Example III.1 (Two buyers) The session types that capture the informal description at the beginning of Example II.1 are as follows. In Section IV, these will be the grades we assign to the computations t_{s}, t_{b_1} and t_{b_2}.

$$g_{\mathsf{s}} = \mathsf{b}_1?\mathtt{query}\langle\mathbf{int}\rangle.\, \mathsf{b}_1!\mathtt{price}\langle\mathbf{int}\rangle.\begin{pmatrix}\mathsf{b}_1?\mathtt{buy}\langle\mathbf{unit}\rangle.\,\mathbf{end}\\ \&\,\mathsf{b}_1?\mathtt{cancel}\langle\mathbf{unit}\rangle.\,\mathbf{end}\end{pmatrix}$$

$$g_{\mathsf{b}_1} = \mathsf{s}!\mathtt{query}\langle\mathbf{int}\rangle.\, \mathsf{s}?\mathtt{price}\langle\mathbf{int}\rangle.\\ \mathsf{b}_2!\mathtt{price}\langle\mathbf{int}\rangle.\begin{pmatrix}\mathsf{b}_2?\mathtt{yes}\langle\mathbf{unit}\rangle.\, \mathsf{s}!\mathtt{buy}\langle\mathbf{unit}\rangle.\,\mathbf{end}\\ \&\,\mathsf{b}_2?\mathtt{no}\langle\mathbf{unit}\rangle.\, \mathsf{s}!\mathtt{cancel}\langle\mathbf{unit}\rangle.\,\mathbf{end}\end{pmatrix}$$

$$g_{\mathsf{b}_2} = \mathsf{b}_1?\mathtt{price}\langle\mathbf{int}\rangle.\begin{pmatrix}\mathsf{b}_1!\mathtt{yes}\langle\mathbf{unit}\rangle.\,\mathbf{end}\\ \&\,\mathsf{b}_1!\mathtt{no}\langle\mathbf{unit}\rangle.\,\mathbf{end}\end{pmatrix}$$

Example III.2 (State with two participants) Consider Example II.2. The session type for state is g_{state} below; the session type for $\mathsf{consumer}$ is its *dual* g_{consumer}.

$$g_{\mathsf{state}} = \mu\mathbf{t}.\, (\mathsf{consumer}?\mathtt{get}\langle\mathbf{unit}\rangle.\, \mathsf{consumer}!\mathtt{st}\langle\mathbf{int}\rangle.\, \mathbf{t})\\ \&(\mathsf{consumer}?\mathtt{put}\langle\mathbf{int}\rangle.\, \mathbf{t})$$
$$h^{\mathtt{cOnce}} = (\mathsf{state}!\mathtt{get}\langle\mathbf{unit}\rangle.\, \mathsf{state}?\mathtt{st}\langle\mathbf{int}\rangle.\, \mathbf{end})\\ \&(\mathsf{state}!\mathtt{put}\langle\mathbf{int}\rangle.\, \mathbf{end})$$
$$g_{\mathsf{consumer}} = \mu\mathbf{t}.\, h^{\mathtt{cOnce}}\cdot\mathbf{t}$$

For the computation $\mathtt{decr}\,0$, we have the following more specific session type $g_{\mathtt{decr}} <: g_{\mathsf{consumer}}$.

$$\mu\mathbf{t}.\, \mathsf{state}!\mathtt{get}\langle\mathbf{unit}\rangle.\, \mathsf{state}?\mathtt{st}\langle\mathbf{int}\rangle.\, \mathsf{state}!\mathtt{put}\langle\mathbf{int}\rangle.\, \mathbf{t}$$

Example III.3 (State with multiple participants) Consider Example II.3. We give session types g_{c_1} and g_{c_2} for the computations t_{c_1} and t_{c_2} as follows.

$$h_{\mathsf{p}}^{\mathtt{waitForYield}} = \mu\mathbf{t}.\, \mathsf{p}?\mathtt{noYield}\langle\mathbf{unit}\rangle.\, \mathbf{t}\\ \&\mathsf{p}?\mathtt{yield}\langle\mathbf{unit}\rangle.\, \mathbf{end}$$
$$h_{\mathsf{p}}^{\mathtt{cYield}} = \mu\mathbf{t}.\, h^{\mathtt{cOnce}}\cdot\begin{pmatrix}\mathsf{p}!\mathtt{noYield}\langle\mathbf{unit}\rangle.\, \mathbf{t}\\ \oplus\mathsf{p}!\mathtt{yield}\langle\mathbf{unit}\rangle.\, \mathbf{end}\end{pmatrix}$$
$$g_{\mathsf{c}_1} = \mathsf{c}_2?\mathtt{yield}\langle\mathbf{unit}\rangle.\, (\mu\mathbf{t}.\, h_{\mathsf{c}_2}^{\mathtt{cYield}}\cdot h_{\mathsf{c}_2}^{\mathtt{waitForYield}}\cdot\mathbf{t})$$
$$g_{\mathsf{c}_2} = \mathsf{c}_1!\mathtt{yield}\langle\mathbf{unit}\rangle.\, (\mu\mathbf{t}.\, h_{\mathsf{c}_1}^{\mathtt{waitForYield}}\cdot h_{\mathsf{c}_1}^{\mathtt{cYield}}\cdot\mathbf{t})$$

The participant $\mathsf{consumer}$ has the same session type g_{consumer} as in the previous example; we explain why in Section III-C.

B. Global types

Once we have assigned a grade to each computation in a session, we need to show that those grades agree in some sense on the protocol they are implementing; it is this that ensures communication safety and deadlock-freedom. In this paper we follow the *top-down* approach to MPST, in which one gives a closed *global type* describing the protocol to be implemented, and then checks that the session types conform to the protocol.

Global types are generated by the following.

$$G \;::=\; \mathbf{end} \mid \mathsf{p}\to\mathsf{q}\colon \{\ell_i\langle \mathsf{b}_i\rangle.\, G_i\}_{i\in I} \mid \mathbf{t} \mid \mu\mathbf{t}.\, G$$

The global type $\mathsf{p}\to\mathsf{q}\colon \{\ell_i\langle \mathsf{b}_i\rangle.\, G_i\}_{i\in I}$ means p sends a single message of type $\ell_i\langle \mathsf{b}_i\rangle$ to q, then continues as G_i. Our usual syntactic restrictions apply again, and we also require the participants p and q to be distinct. We also require G to be neither a type variable nor **end** in $\mu\mathbf{t}.\, G$.

We use the standard reduction $G \overset{\beta}{\rightsquigarrow} G'$ [6], [20] of global types in the statements of our results; this is defined inductively as follows.

$$[\textsc{Tau}]\ G \overset{\tau}{\rightsquigarrow} G \qquad [\textsc{Msg}]\ \frac{v : \mathsf{b}_i}{\mathsf{p} \to \mathsf{q}\colon \{\ell_i\langle \mathsf{b}_i\rangle.\, G_i\}_{i\in i} \overset{\mathsf{p}\to\mathsf{q}:(\ell_j,v)}{\rightsquigarrow} G_j}$$

$$[\textsc{Perm}]\ \frac{G_i \overset{\beta}{\rightsquigarrow} G_i' \text{ for each } i \quad \beta \text{ does not contain } \mathsf{p} \text{ or } \mathsf{q}}{(\mathsf{p} \to \mathsf{q}\colon \{\ell_i\langle \mathsf{b}_i\rangle.\, G_i\}_{i\in I}) \overset{\beta}{\rightsquigarrow} (\mathsf{p} \to \mathsf{q}\colon \{\ell_i\langle \mathsf{b}_i\rangle.\, G_i'\}_{i\in I})}$$

$$[\textsc{Rec}]\ \frac{G[\mathbf{t} \mapsto \mu\mathbf{t}.\, G] \overset{\beta}{\rightsquigarrow} G'}{\mu\mathbf{t}.\, G \overset{\beta}{\rightsquigarrow} G'}$$

Rule [Msg] defines an action from p to q; rule [Perm] allows an action under the body since β does not involve p nor q; and rule [Rec] is standard. We write $g \overset{\mathsf{p}\to\mathsf{q}}{\rightsquigarrow}$ when there is some (ℓ, v) and G' such that $G \overset{\mathsf{p}\to\mathsf{q}:(\ell,v)}{\rightsquigarrow} G'$.

There is also a notion of multiplication $G{\cdot}G'$ of global types, defined analogously to that for session types by replacing **end** with G' in G. We use this in the following examples, but it is not needed for the technical development.

Example III.4 (Two buyers) Consider Example II.1. The following describes the global protocol implemented by the store s and the two buyers b_1 and b_2.

$$\begin{aligned} G = \ & \mathsf{b}_1 \to \mathsf{s}\colon \{\mathtt{query}\langle\mathbf{int}\rangle.\, \mathsf{s} \to \mathsf{b}_1\colon \{\mathtt{price}\langle\mathbf{int}\rangle.\\ & \quad \mathsf{b}_1 \to \mathsf{b}_2\colon \{\mathtt{price}\langle\mathbf{int}\rangle.\, \mathsf{b}_2 \to \mathsf{b}_1\colon \{\\ & \qquad (\mathtt{yes}, \mathbf{unit}).\, \mathsf{b}_1 \to \mathsf{b}_2\colon \{\mathtt{buy}\langle\mathbf{unit}\rangle.\, \mathbf{end}\},\\ & \qquad (\mathtt{no}, \mathbf{unit}).\, \mathsf{b}_1 \to \mathsf{b}_2\colon \{\mathtt{cancel}\langle\mathbf{unit}\rangle.\, \mathbf{end}\}\}\}\}\} \end{aligned}$$

Example III.5 (State with two participants) Consider Example II.2. The following global type G describes the interaction between state and consumer.

$$\begin{aligned} H_{\mathsf{p}}^{\mathtt{cOnce}} &= \mathsf{p} \to \mathsf{state}\colon \{\mathtt{get}\langle\mathbf{unit}\rangle.\, \mathsf{state} \to \mathsf{p}\colon \{\mathtt{st}\langle\mathbf{int}\rangle.\, \mathbf{end}\},\\ &\qquad\qquad \mathtt{put}\langle\mathbf{int}\rangle.\, \mathbf{end}\}\\ G &= \mu\mathbf{t}.\, H_{\mathsf{consumer}}^{\mathtt{cOnce}} \cdot \mathbf{t} \end{aligned}$$

Example III.6 (State with multiple participants) For Example II.3, we have the following global type G.

$$\begin{aligned} & H_{\mathsf{p},\mathsf{q}}^{\mathtt{cYield}} = \mu\mathbf{t}.\, H_{\mathsf{p}}^{\mathtt{cOnce}} \cdot \left(\mathsf{p}\to\mathsf{q}\colon \begin{Bmatrix} \mathtt{noYield}\langle\mathbf{unit}\rangle.\, \mathbf{t}, \\ \mathtt{yield}\langle\mathbf{unit}\rangle.\, \mathbf{end} \end{Bmatrix}\right)\\ & G = \mathsf{c}_2 \to \mathsf{c}_1\colon \{\mathtt{yield}\langle\mathbf{unit}\rangle.\, \mu\mathbf{t}.\, H_{\mathsf{c}_1,\mathsf{c}_2}^{\mathtt{cYield}} \cdot H_{\mathsf{c}_2,\mathsf{c}_1}^{\mathtt{cYield}} \cdot \mathbf{t}\} \end{aligned}$$

C. Relating local and global types

In the standard top-down approach to MPST, given a global type G and a participant r, one defines the *projection* $G \upharpoonright \mathsf{r}$. The latter is a session type describing the behaviour of p in G, if it exists (the projection is a partial operation). To check that the participants in a session conform to the protocol described by G, one then checks that the session types assigned to the participants match the projection.

The definition of projection is as follows. We give the version with *full merging* [20]; merging is a partially defined binary operation $g \sqcap g'$ on session types.

$$\begin{aligned} &(\mathsf{p} \to \mathsf{q}\colon \{\ell_i\langle \mathsf{b}_i\rangle.\, G_i\}_{i\in I}) \upharpoonright \mathsf{r}\\ &\qquad = \begin{cases} \oplus_{i\in I} \mathsf{q}!\ell_i\langle \mathsf{b}_i\rangle.\, G_i \upharpoonright \mathsf{r} & \text{if } \mathsf{p} = \mathsf{r} \wedge \mathsf{q} \neq \mathsf{r}\\ \&_{i\in I} \mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, G_i \upharpoonright \mathsf{r} & \text{if } \mathsf{p} \neq \mathsf{r} \wedge \mathsf{q} = \mathsf{r}\\ \bigsqcap_{i\in I} G_i \upharpoonright \mathsf{r} & \text{if } \mathsf{p} \neq \mathsf{r} \wedge \mathsf{q} \neq \mathsf{r} \end{cases}\\ &\qquad \mathbf{end} \upharpoonright \mathsf{r} = \mathbf{end} \qquad \mathbf{t} \upharpoonright \mathsf{r} = \mathbf{t} \end{aligned}$$

$$(\mu\mathbf{t}.\, G) \upharpoonright \mathsf{r} = \begin{cases} \mathbf{end} & \text{if } G \upharpoonright \mathsf{r} \text{ is a type variable or } \mathbf{end}\\ \mu\mathbf{t}.\, (G \upharpoonright \mathsf{r}) & \text{otherwise} \end{cases}$$

The merge operator is defined, up to unfolding of recursive session types, by the two clauses $g \sqcap g = g$ and

$$\begin{aligned} &(\&_{i\in I} \mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, g_i)\\ &\quad \sqcap (\&_{i\in J} \mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, g_i') \end{aligned} = \begin{aligned} &(\&_{i\in I\cap J} \mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, g_i \sqcap g_i')\\ &\ \&\ (\&_{i\in I\setminus J} \mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, g_i)\\ &\ \&\ (\&_{i\in J\setminus I} \mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, g_i') \end{aligned}$$

We diverge slightly from the standard approach. First, since we permit nested sessions, we have to assign a session type h to the session as a whole; this session type describes how the participants in the session communicate with those outside of it. Second, since we cannot guarantee liveness due to general recursion, it makes sense to weaken the standard definition of conformance. The standard definition ensures that, if a participant still has some communication to do, then that communication happens after a finite number of communications in the global protocol; this is why the standard definition guarantees conformance. Here we require only that there is some finite sequence of messages that would lead to the communication happening, rather than requiring that every such sequence does.

For every set of participants R, we define a relation $G \downarrow_{\mathsf{R}} g$. The intuition is that, when this holds, g describes the messages that participants in R send to or receive from participants outside R. The definition needs a little extra notation. We write $g \leq^{\oplus} h$ if, informally, h and g have the same shape up to unfolding of recursion, but h can potentially have more labels on each send. More precisely, this is defined in the same way as $<:$, but with the rule [LeqRecv] requiring $J = I$. Similarly, we write $g \leq^{\&} h$ for the relation defined by the same rules as $<:$, but with $J = I$ in [LeqSend].

Definition III.7 Let R be a set of participants. We define $\downarrow_{\mathsf{R}}$ coinductively, as the largest relation between global types and local types satisfying the following.

1) If $G \downarrow_{\mathsf{R}} h$ and $G \overset{\mathsf{p}\to\mathsf{q}:(\ell,v)}{\rightsquigarrow} G'$, then there is some h' such that $G' \downarrow_{\mathsf{R}} h'$ and
$$\begin{aligned} h &\geq^{\oplus} h' && \text{if } \mathsf{p} \in \mathsf{R} \wedge \mathsf{q} \in \mathsf{R}\\ h &\overset{\mathsf{q}!(\ell,v)}{\rightsquigarrow} h' && \text{if } \mathsf{p} \in \mathsf{R} \wedge \mathsf{q} \notin \mathsf{R}\\ h &\overset{\mathsf{p}?(\ell,v)}{\rightsquigarrow} h' && \text{if } \mathsf{p} \notin \mathsf{R} \wedge \mathsf{q} \in \mathsf{R}\\ h &\leq^{\&} h' && \text{if } \mathsf{p} \notin \mathsf{R} \wedge \mathsf{q} \notin \mathsf{R} \end{aligned}$$
2) If $G \downarrow_{\mathsf{R}} h$ and $h \overset{\alpha}{\rightsquigarrow} h'$, then there is a finite reduction sequence

$$G \xrightarrow{\mathsf{p}_1 \to \mathsf{q}_1 : (\ell_1, v_1)} \cdots \xrightarrow{\mathsf{p}_m \to \mathsf{q}_m : (\ell_m, v_m)} G'$$

such that $(\mathsf{p}_m \to \mathsf{q}_m : (\ell_m, v_m)) \upharpoonright \mathsf{R} = \alpha$, and $(\mathsf{p}_i \to \mathsf{q}_i : (\ell_i, v_i)) \upharpoonright \mathsf{R} = \tau$ for every $i < m$. Moreover, unless there is such a reduction sequence with $m = 1$, for every such reduction sequence, there is some $i < m$ such that $\{\mathsf{p}_i, \mathsf{q}_i\} \cap \{\mathsf{p}_m, \mathsf{q}_m\} \neq \emptyset$, or such that α is a receive and $\{\mathsf{p}_i, \mathsf{q}_i\} \cap \mathsf{R} = \emptyset$.

3) If $G \downarrow_{\mathsf{R}} h$, then for every finite reduction sequence

$$G \xrightarrow{\mathsf{p}_1 \to \mathsf{q}_1 : (\ell_1, v_1)} \cdots \xrightarrow{\mathsf{p}_m \to \mathsf{q}_m : (\ell_m, v_m)} G'$$

with either (i) $\mathsf{p}_m \in \mathsf{R}$, $\mathsf{q}_m \notin \mathsf{R}$, and $\mathsf{p}_m \notin \{\mathsf{p}_i, \mathsf{q}_i\}$ for all $i < m$, or, (ii) $\mathsf{p}_m \notin \mathsf{R}$, $\mathsf{q}_m \in \mathsf{R}$, and $\mathsf{q}_m \notin \{\mathsf{p}_i, \mathsf{q}_i\}$ for all $i < m$; we have $(\mathsf{p}_i \to \mathsf{q}_i : (\ell_i, v_i)) \upharpoonright \mathsf{R} = \tau$ for all $i < m$.

We write $\downarrow_{\mathsf{r}}$ for $\downarrow_{\{\mathsf{r}\}}$. Clause (2) of this definition requires that every send or receive in h can actually happen, after a finite amount of communication happens, in the global type G. The side condition requires that the communication can happen immediately, unless it is blocked by some other participant. Clause (3) ensures that messages passing into or out of a (nested) session are routed correctly. In particular, it ensures that if a participant in a session is sending to a participant outside of a session, then that send cannot be blocked by some other communication happening across the session boundary. Now we show the projection satisfies our relation:

Proposition III.8 If $G \upharpoonright \mathsf{r}$ is defined, then $G \downarrow_{\mathsf{r}} (G \upharpoonright \mathsf{r})$.

Example III.9 (Two buyers) Consider the session types g_{p} for $\mathsf{p} \in \{\mathsf{s}, \mathsf{b}_1, \mathsf{b}_2\}$, and the global type G, from Examples III.1 and III.4. We have $G \upharpoonright \mathsf{p} = g_{\mathsf{p}}$, and therefore $G \downarrow_{\mathsf{p}} g_{\mathsf{p}}$.

Example III.10 (State with two participants) In the setting of Examples III.2 and III.5, we similarly have $G \upharpoonright \mathsf{p} = g_{\mathsf{p}}$, and hence $G \downarrow_{\mathsf{p}} g_{\mathsf{p}}$, for $\mathsf{p} \in \{\mathsf{state}, \mathsf{consumer}\}$.

Example III.11 (State with two participants) In the setting of Examples III.3 and III.6, we have $G \upharpoonright \mathsf{p} = g_{\mathsf{p}}$, and hence $G \downarrow_{\mathsf{p}} g_{\mathsf{p}}$, for $\mathsf{p} \in \{\mathsf{c}_1, \mathsf{c}_2\}$. We also have $G \downarrow_{\{\mathsf{c}_1, \mathsf{c}_2\}} g_{\mathsf{consumer}}$, so that c_1 and c_2 together act like a single $\mathsf{consumer}$.

In all three of the above cases, the local types *conform* to the protocol described by the global type.

Definition III.12 Let G be a closed global type, and let $\mathsf{R} = \{\mathsf{r}_1, \ldots, \mathsf{r}_n\}$ be a set of participants. The *conformance* predicate $\mathsf{Conf}_G(h; \mathsf{r}_1 \lhd g_1, \ldots, \mathsf{r}_n \lhd g_n)$ holds when (i) every communication $\mathsf{p} \to \mathsf{q}$ in G involves at least one participant from R; (ii) $G \downarrow_{\mathsf{R}} h$; and, (iii) $G \downarrow_{\mathsf{r}_i} g_i$ for every i.

IV. Enforcing Safety in $\lambda_{\mathtt{SafeMP}}$

We now come to our graded type system for $\lambda_{\mathtt{SafeMP}}$. The goal is to assign, to each computation t, a type B and a session type g; but only when the computation has communication safely, and is free of deadlocks. A computation that violates either of these is untypable.

Value typing $\boxed{\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} v : A}$

$$\frac{(x : A) \in \Gamma}{\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} x : A} \qquad \frac{}{\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} () : \mathbf{unit}} \qquad \frac{}{\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} n : \mathbf{int}}$$

$$\frac{}{\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} \mathbf{true} : \mathbf{bool}} \qquad \frac{}{\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} \mathbf{false} : \mathbf{bool}}$$

$$[\textsc{Rec}]\ \frac{\Phi \fatsemi \Gamma, f : A \xrightarrow{g} B, x : A \vdash t : B \fatsemi g}{\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} \mathbf{rec}\ f\, x.\, t : A \xrightarrow{g} B}$$

Computation typing $\boxed{\Phi \fatsemi \Gamma \vdash t : B \fatsemi h}$

$$[<:]\ \frac{\Phi \fatsemi \Gamma \vdash t : B \fatsemi g \quad g <: h}{\Phi \fatsemi \Gamma \vdash t : B \fatsemi h} \qquad [\textsc{Ret}]\ \frac{\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} v : A}{\Phi \fatsemi \Gamma \vdash \mathbf{return}\ v : A \fatsemi \mathbf{end}}$$

$$[\textsc{Let}]\ \frac{\Phi \fatsemi \Gamma \vdash t : A \fatsemi h \quad \Phi \fatsemi \Gamma, x : A \vdash u : B \fatsemi h'}{\Phi \fatsemi \Gamma \vdash \mathbf{let}\ x = t\ \mathbf{in}\ u : B \fatsemi h \cdot h'}$$

$$[-]\ \frac{\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} v : \mathbf{int} \quad \Phi \fatsemi \Gamma \vdash^{\mathrm{v}} w : \mathbf{int}}{\Phi \fatsemi \Gamma \vdash v - w : \mathbf{int} \fatsemi \mathbf{end}}$$

$$[<]\ \frac{\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} v : \mathbf{int} \quad \Phi \fatsemi \Gamma \vdash^{\mathrm{v}} w : \mathbf{int}}{\Phi \fatsemi \Gamma \vdash v < w : \mathbf{bool} \fatsemi \mathbf{end}}$$

$$[\textsc{If}]\ \frac{\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} v : \mathbf{bool} \quad \Phi \fatsemi \Gamma \vdash t_i : B \fatsemi h \quad i = 1, 2}{\Phi \fatsemi \Gamma \vdash \mathbf{if}\ v\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 : B \fatsemi h}$$

$$[\textsc{App}]\ \frac{\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} w : A \xrightarrow{h} B \quad \Phi \fatsemi \Gamma \vdash^{\mathrm{v}} v : A}{\Phi \fatsemi \Gamma \vdash w\, v : B \fatsemi h}$$

$$[\textsc{Snd}]\ \frac{\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} v : \mathsf{b} \quad \mathsf{p} \in \Phi \quad \Phi \fatsemi \Gamma \vdash t : A \fatsemi g}{\Phi \fatsemi \Gamma \vdash \mathbf{send}\,(\ell, v)\ \mathbf{to}\ \mathsf{p}\ \mathbf{then}\ t : A \fatsemi \mathsf{p}!\ell\langle \mathsf{b} \rangle.\, g}$$

$$[\textsc{Rcv}]\ \frac{\mathsf{p} \in \Phi \qquad \Phi \fatsemi \Gamma, x_i : \mathsf{b}_i \vdash t_i : A \fatsemi g_i \text{ for each } i \in I}{\Phi \fatsemi \Gamma \vdash \mathbf{recv}\ \mathsf{p}\, \{(\ell_i, x_i).\, t_i\}_{i \in I} : A \fatsemi \&_{i \in I} \mathsf{p}?\ell_i\langle \mathsf{b}_i \rangle.\, g_i}$$

$$[\textsc{Par}]\ \frac{\begin{array}{c}\Phi, \mathsf{r}_1, \ldots, \mathsf{r}_n \fatsemi \Gamma \vdash t_i : \mathsf{b}_i \fatsemi g_i \text{ for each } i \\ \Phi \fatsemi \Gamma, x_1 : \mathsf{b}_1, \ldots, x_n : \mathsf{b}_n \vdash u : B \fatsemi h' \\ \mathsf{Conf}_G(h; (\mathsf{r}_1 \lhd g_1, \ldots, \mathsf{r}_n \lhd g_n))\end{array}}{\Phi \fatsemi \Gamma \vdash \mathbf{let}\ x_1 = r_1 \lhd t_1 \parallel \cdots \parallel x_n = r_n \lhd t_n\ \mathbf{in}\ u : B \fatsemi h \cdot h'}$$

Fig. 2. Typing of values and computations in $\lambda_{\mathtt{SafeMP}}$

The types are as follows. Note that, as a subset of these, we have the ground types b that we use for message payloads.

$$A, B \ ::= \ \mathbf{unit} \mid \mathbf{bool} \mid \mathbf{int} \mid A \xrightarrow{g} B$$

The only new type here is the function type $A \xrightarrow{g} B$. In this type, the grade g is the session type that describes the communication that happens when the function is applied.

The typing judgement for computations is defined in Section IV; this is defined inductively, and mutually with the typing judgement for values (which do not have grades). It is necessary to track the participants that are in scope, and this is the role of the *participant context* Φ, which is simply a list of participant names. The typing context Γ is a list of (variable name, type) pairs. In both cases, we require that names are not repeated. We write $\cdot$ for the empty (participant or typing) context. The symbol $\fatsemi$ is merely there to separate the different components of a judgement.

Most of the rules are standard from the perspective of fine-grain call-by-value. We just point out a few. Rule [<:] is a

subtyping rule using the relation $<:$ on session types. This rule is standard for graded type systems (e.g. [11]) and MPST systems [7], [20], [21]. Rules [Snd] and [Rcv] simply ensure that the continuations are well-typed, and assign the appropriate session type for the computation. Rule [Par] is the crucial one. Once we have assigned grades g_i for the computations t_i that appear in the session, we need to check that these grades conform to some global type G.

We now make precise our claims about communication safety and deadlock-freedom. We can summarise these properties as a progress property: reduction will never get stuck. Crucially we want these properties to be preserved by reduction too, so we also give a subject reduction result.

Lemma IV.1 (Subject reduction) If $\Phi \fatsemi \cdot \vdash t : B \fatsemi h$ and $t \overset{\alpha}{\rightsquigarrow} t'$, then for all h' such that $h \overset{\alpha}{\rightsquigarrow} h'$, we have $\Phi \fatsemi \cdot \vdash t' : B \fatsemi h'$. Moreover:

- if $\alpha = \tau$ or $\alpha = \mathsf{p}!(\ell, v)$, then there exists such an h';
- if $\alpha = \mathsf{p}?(\ell, v)$, then $h \overset{\mathsf{p}?}{\rightsquigarrow}$.

In the statement of this lemma, the first sentence is the subject reduction we are interested in. We include the second sentence because it is a crucial part of the proof by induction on the reduction: it is needed for the inductive hypothesis to be sufficiently strong. Below we give the crucial lemma used to handle the [Par] reduction rule: the point is that, if the session reduces by some message being sent, then the global type reduces accordingly.

Lemma IV.2 Assume $\mathsf{Conf}_G(h; \mathsf{r}_1 \lhd g_1, \ldots, \mathsf{r}_n \lhd g_n)$.

1) If $g_j \overset{\mathsf{r}_k!(\ell,v)}{\rightsquigarrow} g'_j$ and $g_k \overset{\mathsf{r}_j?}{\rightsquigarrow}$, then there is some G' such that $G \overset{\mathsf{r}_j \to \mathsf{r}_k:(\ell,v)}{\rightsquigarrow} G'$.
2) If $g_j \overset{\mathsf{p}!(\ell,v)}{\rightsquigarrow} g'_j$, with $\mathsf{p} \notin \{\mathsf{r}_1, \ldots, \mathsf{r}_n\}$, then there is some G' such that $G \overset{\mathsf{r}_j \to \mathsf{p}:(\ell,v)}{\rightsquigarrow} G'$.
3) If $h \overset{\mathsf{p}?(\ell,v)}{\rightsquigarrow} h'$ and $g_k \overset{\mathsf{p}?}{\rightsquigarrow}$, with $\mathsf{p} \notin \{\mathsf{r}_1, \ldots, \mathsf{r}_n\}$, there is some G' such that $G \overset{\mathsf{p} \to \mathsf{r}_k:(\ell,v)}{\rightsquigarrow} G'$.

To establish progress, we use the following lemma, which we again state in the required form for the induction to go through. The induction in this case is on t.

Lemma IV.3 If $\Phi \fatsemi \cdot \vdash t : B \fatsemi h$, then either (i) there is a reduction $t \overset{\tau}{\rightsquigarrow} t'$, or (ii) one of the following holds.

1) $h = \mathbf{end}$, and $t = \mathbf{return}\ v$ for some $v : B$.
2) $h \overset{\mathsf{p}!}{\rightsquigarrow}$ for some p, and there exists a message (ℓ, v) such that there are reductions $h \overset{\mathsf{p}!(\ell,v)}{\rightsquigarrow} h'$ and $t \overset{\mathsf{p}!(\ell,v)}{\rightsquigarrow} t'$.
3) $h \overset{\mathsf{p}?}{\rightsquigarrow}$ for some p, and for every message (ℓ, v) such that there is a reduction $h \overset{\mathsf{p}?(\ell,v)}{\rightsquigarrow} h'$, there is a reduction $t \overset{\mathsf{p}?(\ell,v)}{\rightsquigarrow} t'$.

If we put this together with subject reduction, we arrive at the main theorem of this section, which captures both the **communication safety** and **deadlock-freedom** properties mentioned in the introduction.

Theorem IV.4 Assume that $\cdot \fatsemi \cdot \vdash t : B \fatsemi h$. Then if $t \overset{\tau}{\rightsquigarrow}{}^* t'$, either $t' = \mathbf{return}\ v$, or there is some t'' such that $t' \overset{\tau}{\rightsquigarrow} t''$.

V. Denotational semantics

One of the aims of this work is to give a denotational interpretation of multiparty session types. This section does so, by giving a denotational semantics for $\lambda_{\texttt{SafeMP}}$. Our semantics leans heavily on the typical techniques used to develop semantics for computational effects. In particular, since we have a graded type system, we base our semantics around a *graded monad*, as in [11]. Our semantics will also enable use to prove that there is no observable nondeterminism in $\lambda_{\texttt{SafeMP}}$.

Since we have general recursion in $\lambda_{\texttt{SafeMP}}$, we base our models on domain theory; for us ω*cpo*s will suffice. We start by giving the standard definitions and fixing some notation.

Definition V.1 A *ωcpo* is a set X equipped with a partial ordering $\leq$, such that each ω-chain $x_1 \leq x_2 \leq \ldots$ has an upper bound $\bigvee_i x_i$. A function $f \colon X \to Y$ between ωcpos is *continuous* when it is monotone, and preserves least upper bounds of ω-chains.

We write $\prod_{i \in I} X_i$ for a product of ωcpos indexed by a set I, and $X \Rightarrow Y$ for the ωcpo of continuous functions $X \to Y$. If X is an ωcpo with a least element $\bot$, then every continuous function $\phi \colon X \to X$ has a least pre-fixed point in X, which we write as $\mathit{fix}\ \phi$. Each ground type b is interpreted as a discrete ωcpo $[\![\mathsf{b}]\!]$: $[\![\mathbf{unit}]\!]$ has a single element $\star$; the elements of $[\![\mathbf{int}]\!]$ are the integers; and $[\![\mathbf{bool}]\!]$ has two elements true and false. We write $\mathcal{M}(\ell_i : \mathsf{b}_i)_{i \in I}$ for the set $\{(\ell_i, v) \mid i \in I \wedge v \in [\![\mathsf{b}_i]\!]\}$.

A. Send–receive algebras

We introduce a notion of *send–receive algebra*, which is a space in which we can interpret sending and receiving of messages.

Definition V.2 A *Φ-send–receive algebra* is a ωcpo Z (the *carrier*), equipped with a least element $\bot$ and continuous functions

$$\mathrm{send}_{\mathsf{p},(\ell,v)} \colon Z \to Z \quad \mathrm{recv}_{\mathsf{p},\{\ell_i:\mathsf{b}_i\}_{i \in I}} \colon Z^{\mathcal{M}(\ell_i:\mathsf{b}_i)_{i \in I}} \to Z$$

where in the subscript of recv, I ranges over finite nonempty sets, and the labels ℓ_i are distinct.

We interpret computations as elements of *free* send–receive algebras. For every ωcpo X, there is a *free* Φ-send–receive algebra on X, whose carrier we denote by $\mathrm{SR}_\Phi(X)$. This comes with a continuous function $\mathrm{return}_X \colon X \to \mathrm{SR}_\Phi(X)$, called the *unit*, such that for every Φ-send–receive algebra Z, there is a unique function

$$(\gg\!=^Z_X) \colon \mathrm{SR}_\Phi(X) \times (X \Rightarrow Z) \to Z$$

that is continuous in its first argument, and satisfies

$$(\mathrm{return}_X x) \gg\!=^Z_X f = f\,x$$
$$(\mathrm{send}_{\mathsf{p},(\ell,v)} t) \gg\!=^Z_X f = \mathrm{send}_{\mathsf{p},(\ell,v)}(t \gg\!=^Z_X f)$$
$$(\mathrm{recv}_{\mathsf{p},\{\ell_i:\mathsf{b}_i\}_i}(t_m)_m) \gg\!=^Z_X f = \mathrm{recv}_{\mathsf{p},\{\ell_i:\mathsf{b}_i\}_i}(t_m \gg\!=^Z_X f)_m$$

This function is moreover continuous in its second argument.

Concretely, we can construct the free send–receive algebra coinductively as follows.[3] Let $\mathrm{SR}_\Phi(X)$ be the largest set such that each element t is either: $\bot$, or $\mathrm{return}_X x$ for some $x \in X$, or $\mathrm{send}_{\mathsf{p},(\ell,v)} t$ where $t \in \mathrm{SR}_\Phi(X)$, or $\mathrm{recv}_{\mathsf{p},\{\ell_i:\mathsf{b}_i\}_{i\in I}}(t_m)_m$ where $t_m \in \mathrm{SR}_\Phi(X)$ for each $m \in \mathcal{M}(\ell_i : \mathsf{b}_i)_{i\in I}$. Thus we can depict an element of $\mathrm{SR}_\Phi(X)$ as a possibly-infinite tree, where each node is labeled by either $\mathrm{send}_{\mathsf{p},(\ell,v)}$ or $\mathrm{recv}_{\mathsf{p},\{\ell_i:\mathsf{b}_i\}_{i\in I}}$, and each leaf is either $\bot$ or $\mathrm{return}_X x$ for some $x \in X$. The ordering $\leq$ on elements of $\mathrm{SR}_\Phi(X)$ is also defined coinductively, by the following rules.

$$\bot \leq t \qquad \frac{x \leq x'}{\mathrm{return}_X x \leq \mathrm{return}_X x'}$$

$$\frac{t \leq t'}{\mathrm{send}_{\mathsf{p},m} t \leq \mathrm{send}_{\mathsf{p},m} t'} \qquad \frac{t_m \leq t'_m \text{ for all } m}{\mathrm{recv}_{\mathsf{p},\{\ell_i:\mathsf{b}_i\}_i} t \leq \mathrm{recv}_{\mathsf{p},\{\ell_i:\mathsf{b}_i\}_i} t'}$$

Since these are free algebras, they form a *strong monad* SR_Φ for each Φ, which following Moggi [26] is one of the primary tools for modelling computational effects. For fine-grain call-by-value, a strong monad provides a model of **return** and (non-parallel) **let**. We do not give the monads explicitly, since for us it is more useful to give a graded monad.

The above provides a model of most of $\lambda_{\texttt{SafeMP}}$, including recursion, message-passing, and the core of fine grain call-by-value. However, unless we introduce grades, it cannot model the parallel **let** construct. To model parallel **let**, we would like to have a continuous function

$$\mathrm{fork} \colon \textstyle\prod_i \mathrm{SR}_{\Phi,\mathsf{r}_1,\ldots,\mathsf{r}_n}(X_i)_{g_i} \to \mathrm{SR}_\Phi(\prod_i X_i)$$

which takes a tuple of elements of a free algebra, thought of as the interpretations of computations in a session, and carries out the interactions between participants in the session. The problem is that, without ensuring conformance, there are nondeterministic computations in $\lambda_{\texttt{SafeMP}}$, but free send–receive algebras only model deterministic computations. Thus we do not get such a function, but rather a relation $\succ$, which is given in the following definition. The intuition is that, if $(t_1,\ldots,t_n) \succ u$ holds, then u models the communication that happens between a participant in the session and a participant outside of the session. Interactions between participants inside the session get hidden by the rule [Int] below; this gives our semantics its extensional character. The rules defining $\succ$ are analogous to the operational semantics for parallel **let**.

[3]The justification for this construction requires some technology. Firstly, as is often the case in category theory [22], the free algebra is given as the colimit of a chain; in this case we take the the colimit of the chain of ωcpos of finite trees with height at most n. By the limit-colimit coincidence [23], [24], this colimit is equivalently a limit, and this limit is given by the coinductive construction here. We are not the first apply this kind of construction to model computational effects in the category of ωcpos; see for instance [25, Section 4.5] for a detailed example involving recursion, I/O and exceptions.

Definition V.3 Let $\mathsf{R} = \{\mathsf{r}_1,\ldots,\mathsf{r}_n\}$ be a set of participants, and let $X_1,\ldots,X_n$ be ωcpos. We define a relation $(t_1,\ldots,t_n) \succ u$, between trees $t_i \in \mathrm{SR}_{\Phi,\mathsf{r}_1,\ldots,\mathsf{r}_n}(X_i)$ and trees $u \in \mathrm{SR}_\Phi(X)$, inductively as the smallest relation that is closed under taking least upper bounds of ω-chains, and under the following rules.

$$[\textsc{Bot}]\ \frac{}{(t_1,\ldots,t_n) \succ \bot} \qquad [\textsc{Return}]\ \frac{x_i \geq x'_i \text{ for each } i}{(\mathrm{return}\,x_1,\ldots,\mathrm{return}\,x_n) \succ \mathrm{return}(x'_1,\ldots,x'_n)}$$

$$[\textsc{Send}]\ \frac{(\ldots,t_{j-1},t_j,t_{j+1},\ldots) \succ u \qquad \mathsf{p} \notin \mathsf{R}}{(\ldots,t_{j-1},\mathrm{send}_{\mathsf{p},(\ell,v)}(t_j),t_{j+1},\ldots) \succ \mathrm{send}_{\mathsf{p},(\ell,v)} u}$$

$$[\textsc{Recv}]\ \frac{(\ldots,t_{j-1},t_{j,m},t_{j+1},\ldots) \succ u_m \text{ for all } m \qquad \mathsf{p} \notin \mathsf{R}}{(\ldots,t_{j-1},\mathrm{recv}_{\mathsf{p},\{\ell_i,\mathsf{b}_i\}}(t_j),t_{j+1},\ldots) \succ \mathrm{recv}_{\mathsf{p},\{\ell_i:\mathsf{b}_i\}} u}$$

$$[\textsc{Int}]\ \frac{(t_1,\ldots,t_n) \succ u}{(t'_1,\ldots,t'_n) \succ u} \text{ where } t'_i = \begin{cases} \mathrm{send}_{\mathsf{r}_k,(\ell,v)}(t_j) & \text{if } i = j \\ \mathrm{recv}_{\mathsf{r}_j,\{\ell_i:\mathsf{b}_i\}_i}(t''_{k,m})_m & \text{if } i = k \\ t_i & \text{otherwise} \end{cases}$$
$$t_k = t''_{k,(\ell,v)}$$

B. Graded send–receive algebras

We have now set up most of the definitions we need to give the semantics, but as we mention above, we need to refine them using grades to actually model $\lambda_{\texttt{SafeMP}}$. Crucially, for each grade g, we can pick out the subset of each free send-receive algebra containing the interpretations of computations that match the grade.

Definition V.4 Let X be an ωcpo. We define a family of subsets $\mathrm{GSR}_\Phi(X)_g$ of $\mathrm{SR}_\Phi(X)$, indexed by closed session types g, coinductively, as the largest family of sets such that if $t \in \mathrm{GSR}_\Phi(X)_g$ then

- $g = \mathbf{end}$ implies $t = \bot$ or $t = \mathrm{return}_X x$ for some $x \in X$;
- $g = \oplus_{i\in I} \mathsf{p}!\ell_i\langle \mathsf{b}_i\rangle.\, g'_i$ implies $t = \bot$ or $t = \mathrm{send}_{\mathsf{p},(\ell_i,v)} t'$ for some $i \in I$, $v \in [\![\mathsf{b}_i]\!]$, and $t' \in \mathrm{GSR}_\Phi(X)_{g'}$;
- $g = \&_{i\in I} \mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, g'_i$ implies $t = \bot$ or $t = \mathrm{recv}_{\mathsf{p},\{\ell_i:\mathsf{b}_i\}_{i\in J}} t'$ for some $J \supseteq I$, $\{\ell_i, \mathsf{b}_i, g'_i\}_{i\in J\setminus I}$, and $t'_{(\ell_i,v)} \in \mathrm{GSR}_\Phi(X)_{g'_i}$ for each $i \in J$;
- $g = \mu \mathbf{t}.\, g'$ implies $t \in \mathrm{GSR}_\Phi(X)_{g'[\mathbf{t}\mapsto \mu\mathbf{t}.\, g']}$.

Crucially, grading the free send–receive algebras in this way enables us to define the continuous functions fork, in the presence of conformance.

Lemma V.5 Assume that $\mathsf{Conf}_G(h; \mathsf{r}_1 \lhd g_1,\ldots,\mathsf{r}_n \lhd g_n))$. If $t_i \in \mathrm{GSR}_{\Phi,\mathsf{r}_1,\ldots,\mathsf{r}_n}(X_i)_{g_i}$ for each i, then the set

$$\{u \mid (t_1,\ldots,t_n) \succ u\}$$

has a least upper bound $\mathrm{fork}(t_1,\ldots,t_n)$, and this least upper bound is in $\mathrm{GSR}_\Phi(\prod_i X_i)_h$. These least upper bounds form a continuous function

$$\mathrm{fork} \colon \textstyle\prod_i \mathrm{GSR}_{\Phi,\mathsf{r}_1,\ldots,\mathsf{r}_n}(X_i)_{g_i} \to \mathrm{GSR}_\Phi(\prod_i X_i)_h$$

such that $\mathrm{fork}(t_1,\ldots,t_n) \geq u$ is equivalent to $(t_1,\ldots,t_n) \succ u$.

We informally outline why this works. If there is some derivation of $(t_1,\ldots,t_n) \succ u$ using the rule [Return], then the least

upper bound will have the form $\mathrm{return}(x_1, \ldots, x_n)$. Similarly, a derivation using [SEND] or [RECV], because some participant r_i sends or receives, implies that the least upper bound involves $\mathrm{send}_{\mathsf{p}}$ or $\mathrm{recv}_{\mathsf{p}}$. Crucially, by conformance (specifically using Definition III.7(3)), there can only be one such r_i; so we do not have to worry about these cases overlapping. We do have to worry about [INT] overlapping with these cases, but this turns out to be harmless.

To model the core of fine-grain call-by-value in the presence of grading, we show that above subsets form a *graded monad* GSR_Φ; the bind operator $\gg\!=$ of a graded monad models sequencing of computations. The precise definition is as follows, where we write $\mathcal{G}_\Phi$ for the ordered monoid of closed session types, whose participants are from Φ.

Definition V.6 A $\mathcal{G}_\Phi$*-graded* ω*cpo* X comprises a ω*cpo* X_g for each $g \in \mathcal{G}_\Phi$, and a continuous function $X_{g<:h} \colon X_g \to X_h$, for each $g <: h$, satisfying the identity law $\mathrm{id} = X_{g<:g}$ and composition law $X_{h<:h'} \circ X_{g<:h} = X_{g<:h'}$.

Definition V.7 A $\mathcal{G}_\Phi$-graded $\omega\mathbf{Cpo}$-*monad*[4] T comprises:

1) a $\mathcal{G}_\Phi$-graded ωcpo $T(X)$ for each ωcpo X;
2) a continuous function $\mathrm{return}_X \colon X \to T(X)_{\mathbf{end}}$ for each X;
3) a continuous function $(\overset{g,g'}{\gg\!=}) \colon TXg \times (X \Rightarrow TYg') \to TY(g \cdot g')$ for each X, Y, g, g'.

These are required to satisfy the three *monad laws*

$$(\mathrm{return}_X x) \overset{\mathbf{end},g}{\gg\!=} f = f\,x \qquad t \overset{g,\mathbf{end}}{\gg\!=} \mathrm{return}_X = t$$
$$t \overset{g,g'\cdot g''}{\gg\!=} (f' \circ f) = (t \overset{g,g'}{\gg\!=} f) \overset{g\cdot g',g''}{\gg\!=} f'$$

and naturality of $\overset{g,g'}{\gg\!=}$ in g and g'.

In the case of $T = \mathrm{GSR}_\Phi$, the bind operator is given by taking the unique continuous functions $\gg\!=^{\mathrm{SR}_\Phi(Y)}$ defined above, and restricting the domain and codomain to subsets according to the grades.

C. Interpretation of $\lambda_{\texttt{SafeMP}}$

Above we already defined the interpretations $[\![\mathsf{b}]\!]$ of the ground types as ωcpos. Given a participant context Φ, we extend this to an interpretation $[\![A]\!]$ of each type A in participant context Φ by defining $\left[\!\left[A \xrightarrow{g} B\right]\!\right] = ([\![A]\!] \Rightarrow \mathrm{GSR}_\Phi([\![B]\!])_g)$. We extend this to an interpretation $[\![\Gamma]\!]$ of each Γ by defining $[\![\cdot]\!] = \{\star\}$ and $[\![\Gamma, x : A]\!] = [\![\Gamma]\!] \times [\![A]\!]$. We write ρ for a generic element of Γ, and, if $(x : A) \in \Gamma$, write $\rho_x \in [\![A]\!]$ for the projection of the value of the variable x in ρ.

The interpretation of a value $\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} v : A$ is then a continuous function $[\![v]\!] \colon [\![\Gamma]\!] \to [\![A]\!]$, and the interpretation of a computation $\Phi \fatsemi \Gamma \vdash t : B \fatsemi g$, is a continuous function $[\![t]\!] \colon [\![\Gamma]\!] \to \mathrm{GSR}_\Phi([\![B]\!])_g$. These are defined in Fig. 3.

[4]More precisely, here $\omega\mathbf{Cpo}$ denotes the category of ωcpos and continuous functions, and this is a definition of an $\omega\mathbf{Cpo}$-*enriched* monad on $\omega\mathbf{Cpo}$, presented in *extension form* [27]. The enrichment corresponds, by a result of Kock [28] to the existence of the *strength* required by Moggi [26]. The definition of graded monad below is also enriched, and hence strong.

$$\begin{aligned}
[\![x]\!](\rho) &= \rho_x \\
[\![()]\!](\rho) &= \star \\
[\![n]\!](\rho) &= n \\
[\![\mathbf{true}]\!](\rho) &= \mathrm{true} \\
[\![\mathbf{false}]\!](\rho) &= \mathrm{false} \\
[\![\mathbf{rec}\ f\,x.\,t]\!](\rho) &= \mathit{fix}(\lambda\phi.\,\lambda a.\,[\![t]\!](\rho, \phi\,a)) \\
[\![\mathbf{return}\ v]\!](\rho) &= \mathrm{return}_{[\![B]\!]}([\![v]\!](\rho)) \\
[\![\mathbf{let}\ x = t\ \mathbf{in}\ u]\!](\rho) &= [\![t]\!](\rho) \gg\!= (\lambda a.\,[\![u]\!](\rho, a)) \\
[\![v - w]\!](\rho) &= [\![v]\!](\rho) - [\![w]\!](\rho) \\
[\![v < w]\!](\rho) &= \begin{cases} \mathrm{true} & \text{if } [\![v]\!](\rho) < [\![w]\!](\rho) \\ \mathrm{false} & \text{otherwise} \end{cases} \\
[\![\mathbf{if}\ v\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2]\!](\rho) &= \begin{cases} [\![t_1]\!](\rho) & \text{if } [\![v]\!](\rho) = \mathrm{true} \\ [\![t_2]\!](\rho) & \text{if } [\![v]\!](\rho) = \mathrm{false} \end{cases} \\
[\![w\,v]\!](\rho) &= [\![w]\!](\rho)([\![v]\!](\rho)) \\
[\![\mathbf{send}\,(p, \ell)\ \mathbf{to}\ \mathsf{v}\ \mathbf{then}\ t]\!](\rho) &= \mathrm{send}_{\mathsf{p},(\ell,v)}([\![t]\!](\rho)) \\
[\![\mathbf{recv}\ \mathsf{p}\,\{(\ell_i, x_i).\,t_i\}_{i\in I}]\!](\rho) &= \mathrm{recv}_{\mathsf{p},\{\ell_i:\mathsf{b}_i\}}(\lambda(\ell_i, b_i).\,[\![t_i]\!](\rho, b_i)) \\
[\![\mathbf{let}\ x_1 = \mathsf{r}_1 \lhd t_1 \parallel \cdots \parallel x_n = \mathsf{r}_n \lhd t_n\ \mathbf{in}\ u]\!](\rho) &= \\
\mathrm{fork}([\![t_1]\!](\rho), \ldots, [\![t_n]\!](\rho)) \gg\!= (\lambda(x_1, \ldots, x_n).\,[\![u]\!](\rho, x_1, \ldots, x_n)) &
\end{aligned}$$

Fig. 3. Interpretation of values and computations of $\lambda_{\texttt{SafeMP}}$

Crucially, the denotational semantics defined in Fig. 3 is sound in the following sense.

Lemma V.8 (Soundness) Let $t \overset{\alpha}{\rightsquigarrow} t'$ be a reduction, starting from a computation t such that $\Phi \fatsemi \cdot \vdash t : B \fatsemi h$. If $[\![t]\!] = \bot$, then $[\![t']\!] = \bot$; otherwise, we have the following.

1) If $\alpha = \tau$, then $[\![t]\!] = [\![t']\!]$.
2) If $\alpha = \mathsf{p}!(\ell, v)$, then $[\![t]\!] = \mathrm{send}_{\mathsf{p},(\ell,[\![v]\!])}[\![t']\!]$.
3) If $\alpha = \mathsf{p}?(\ell, v)$, then $[\![t]\!] = \mathrm{recv}_{\mathsf{p},\{\ell_i:\mathsf{b}_i\}_i}([\![t'_m]\!])_m$, where $\mathcal{M}(\ell_i : \mathsf{b}_i)_{i\in I}$ consists of those m such that there is some t'_m satisfying $t \overset{\mathsf{p}?(\ell,v)}{\rightsquigarrow} t'_m$.

As a corollary, we obtain the fact that $\lambda_{\texttt{SafeMP}}$ has **no observable nondeterminism**:

Corollary V.9 Let t be a computation such that $\cdot \fatsemi \cdot \vdash t : \mathsf{b} \fatsemi g$. If $t \overset{\tau}{\rightsquigarrow}{}^*$ **return** v and $t \overset{\tau}{\rightsquigarrow}{}^*$ **return** w then $v = w$.

Proof. By soundness, we have $\mathrm{return}[\![v]\!] = [\![\mathbf{return}\ v]\!] = [\![t]\!] = [\![\mathbf{return}\ w]\!] = \mathrm{return}[\![w]\!]$, which implies $[\![v]\!] = [\![w]\!]$ and hence $v = w$. □

This property is also enjoyed by other systems in the MPST literature, where it follows from the fact that communication is limited to message-passing, and that there are typically no nested sessions. The above corollary shows that we recover it because of our graded type system.

VI. Enforcing liveness

Our graded type system for $\lambda_{\texttt{SafeMP}}$ (Section IV) does not satisfy liveness, primarily because we permit general recursion: even if the grade of a computation suggests the computation will send or receive a message, the computation itself may diverge without sending or receiving anything.

The solution to this is to control recursion. Specifically, we adjust the typing rule for recursive functions to guarantee if a function wants to make a recursive call, it must first send or receive a message. This solution is inspired by *guarded recursion* [29], in which recursive calls have to be in some sense guarded, though here we record the relevant information about guardedness in the grade, rather than in the type.

The interpretation of each session type g thus slightly changes, so that they no longer permit divergence. We adjust our grammar of session types, by adding a new session type $\mathbf{tick}\ g$. This informally means *the computation will behave as in g, but potentially only after some communication happens (e.g. an internal communication within some session)*:

$$g, h \ ::= \ \cdots \mid \mathbf{tick}\ g$$

In recursive session types $\mu\mathbf{t}.\, g$, we require g to have the form $\mathbf{tick}^{\kappa}\ g'$ for a natural number κ, where g' is either a send $\oplus$, receive $\&$, or a recursive type.

We extend multiplication of grades in the obvious way, and add two new rules to the coinductive definition of $<::$

$$[\textsc{Con}] \quad (\mathbf{tick}\ g) \cdot h = \mathbf{tick}\ (g \cdot h)$$

$$[\textsc{Tick}]\ \frac{g <: h}{\mathbf{tick}\ g <: \mathbf{tick}\ h} \qquad [\textsc{Extra}]\ \frac{g <: h}{g <: \mathbf{tick}\ h}$$

We adjust the reduction rules for sending and receiving to account for the fact that the continuation occurs after some communication happens, and add a congruence rule for **tick**.

$$\frac{v : \mathsf{b}_j}{\oplus_i \mathsf{p}!\ell_i\langle \mathsf{b}_i\rangle.\, g_i \overset{\mathsf{p}!(\ell_j, v)}{\rightsquigarrow} \mathbf{tick}\ g_j} \qquad \frac{v : \mathsf{b}_j}{\&_i \mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, g_i \overset{\mathsf{p}?(\ell_j, v)}{\rightsquigarrow} \mathbf{tick}\ g_j}$$

$$\frac{g \overset{\alpha}{\rightsquigarrow} g'}{\mathbf{tick}\ g \overset{\alpha}{\rightsquigarrow} \mathbf{tick}\ g'}$$

We make similar adjustments to global types, extending the grammar, adjusting the reduction rule for communication, and adding a congruence rule for **tick**:

$$G \ ::= \ \cdots \mid \mathbf{tick}\ G \qquad [\textsc{Tick}]\ \frac{G \overset{\beta}{\rightsquigarrow} G'}{\mathbf{tick}\ G \overset{\beta}{\rightsquigarrow} \mathbf{tick}\ G'}$$

$$[\textsc{Tprem}]\ \frac{v : \mathsf{b}_j}{\mathsf{p} \to \mathsf{q}\colon \{\ell_i\langle \mathsf{b}_i\rangle.\, G_i\}_{i\in I} \overset{\mathsf{p}\to\mathsf{q}:(\ell_j, v)}{\rightsquigarrow} \mathbf{tick}\ G_j}$$

Finally, we adjust $G \downarrow_{\mathsf{R}} h$ (Definition III.7), specifically by adding two new clauses to account for **tick**:

4) If $G \downarrow_{\mathsf{R}} h$ and $G = \mathbf{tick}\ G'$, then there is some h' such that $G' \downarrow_{\mathsf{R}} h'$ and $h = \mathbf{tick}\ h'$.
5) If $G \downarrow_{\mathsf{R}} h$ and $h = \mathbf{tick}\ h'$, then there is some G' such that $G' \downarrow_{\mathsf{R}} h'$, and either $G = \mathbf{tick}\ G'$, or $G \overset{\beta}{\rightsquigarrow} \mathbf{tick}\ G'$ with $\beta \upharpoonright \mathsf{R} = \tau$.

Note that in clause (1) of the existing definition, the relations $\leq^{\oplus}$ and $\leq^{\&}$ are defined in the same way as before, but with a congruence rule for **tick**. We do *not* have [Extra] in the definitions of these. We use the same definition of Conf as before, but accounting for our changes to the definition of $\downarrow_{\mathsf{R}}$.

Example VI.1 We give a small example to emphasize the impact of our changes. Consider the global type $G = \mu\mathbf{t}.\, \mathsf{p} \to \mathsf{q}\colon \{(\mathtt{cont}, \mathbf{unit}).\, \mathbf{t}, (\mathtt{stop}, \mathbf{unit}).\, \mathsf{q} \to \mathsf{r}\colon \{\ell\langle \mathbf{unit}\rangle.\ \mathbf{end}\}\}$. This global type permits an infinite sequence of `cont` messages, and hence starvation of r. In the setup of the previous sections, we have $G \downarrow_{\mathsf{r}} (\mathsf{q}?\ell\langle \mathbf{unit}\rangle.\ \mathbf{end})$ (though the projection $G \upharpoonright \mathsf{r}$ is undefined). With the addition of **tick** however, there is no h such that $G \downarrow_{\mathsf{r}} h$. If there were, since G reduces in κ steps to $\mathbf{tick}^{\kappa}\ G$, we would need to have, for every κ, that $h \leq^{\oplus} \mathbf{tick}^{\kappa}\ h'$ for some h'. No h satisfies this.

We adjust our graded type system by replacing three rules. We change the typing rules [Snd] and [Rcv] to match reduction of local types, as follows.

$$[\textsc{TSnd}]\ \frac{\Phi \fatsemi \Gamma \vdash^{\mathsf{v}} v : \mathsf{b} \qquad \mathsf{p} \in \Phi \qquad \Phi \fatsemi \Gamma \vdash t : A \fatsemi \mathbf{tick}\ g}{\Phi \fatsemi \Gamma \vdash \mathbf{send}\ (\ell, v)\ \mathbf{to}\ \mathsf{p}\ \mathbf{then}\ t : A \fatsemi \mathsf{p}!\ell\langle \mathsf{b}\rangle.\, g}$$

$$[\textsc{TRcv}]\ \frac{\mathsf{p} \in \Phi \qquad \{\Phi \fatsemi \Gamma, x_i : \mathsf{b} \vdash t_i : A \fatsemi \mathbf{tick}\ g_i\}_{i\in I}}{\Phi \fatsemi \Gamma \vdash \mathbf{recv}\ \mathsf{p}\, \{(\ell_i, x_i).\, t_i\}_{i\in I} : A \fatsemi \&_{i\in I} \mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, g_i}$$

The remaining change, and the one that guarantees liveness, is replacing the typing rule [Rec] with the following.

$$\frac{\Phi \fatsemi \Gamma, f : A \xrightarrow{\mathbf{tick}\ g} B, x : A \vdash t : B \fatsemi g}{\Phi \fatsemi \Gamma \vdash^{\mathsf{v}} \mathbf{rec}\ f\, x.\, t : A \xrightarrow{\mathbf{tick}^{\kappa}\ g} B}$$

With this rule, a recursive call is possible for instance after a **send** or **recv**, but the function $\mathbf{rec}\ f\, x.\, f\, x$ is not typable.

We claim that the modifications above are enough to guarantee liveness, which is Theorem VI.3 below. The key lemma needed to establish liveness is the following, which is a strengthened version of our progress lemma (Lemma IV.3).

Lemma VI.2 Assume that t satisfies $\Phi \fatsemi \cdot \vdash t : B \fatsemi h$.

1) If $h = \mathbf{tick}^{\kappa}\ \mathbf{end}$, then there exists t' such that $t \overset{\tau}{\rightsquigarrow}{}^{*}\ t'$.
2) If $h \overset{p!}{\rightsquigarrow}$, then for some reduction $h \overset{\mathsf{p}!(\ell, v)}{\rightsquigarrow} h'$, there exists t' such that $t \overset{\mathsf{q}!(\ell_i, v)}{\rightsquigarrow}{}^{*}\ t'$.
3) If $h \overset{\mathsf{p}?}{\rightsquigarrow}$, then for every reduction $h \overset{\mathsf{p}?(\ell, v)}{\rightsquigarrow} h'$, there exists t' such that $t \overset{\mathsf{q}?(\ell_i, v)}{\rightsquigarrow}{}^{*}\ t'$.

The proof of this lemma is by a logical relations argument; we give details in Section D. It then follows that, in every closed session that conforms to a global protocol in the sense of this section, we have **liveness**.

Theorem VI.3 (Liveness) Let $t_1, \ldots, t_n$ be computations satisfying $\Phi, \mathsf{r}_1, \ldots, \mathsf{r}_n \fatsemi \cdot \vdash t_i : \mathsf{b}_i \fatsemi g_i$, and assume that $\mathsf{Conf}_G(h; (\mathsf{r}_1 \lhd g_1 \parallel \cdots \parallel \mathsf{r}_n \lhd g_n))$ holds.

1) If $g_k \overset{\mathsf{q}!}{\rightsquigarrow}$, then there is a finite sequence of reductions
$$(\mathsf{r}_1 \lhd t_1 \parallel \cdots \parallel \mathsf{r}_n \lhd t_n) \overset{\beta_1}{\rightsquigarrow} \cdots \overset{\beta_n}{\rightsquigarrow} (\mathsf{r}_1 \lhd t_1' \parallel \cdots \parallel \mathsf{r}_n \lhd t_n')$$

such that $\beta_n = \mathsf{r_k} \to \mathsf{q} : (\ell, v)$ and $g_k \overset{\mathsf{q}!(\ell,v)}{\rightsquigarrow} g'$ for some ℓ, v, g'.

2) If $g_k \overset{\mathsf{q}?}{\rightsquigarrow}$, then there is a finite sequence of reductions

$$(\mathsf{r}_1 \lhd t_1 \parallel \cdots \parallel \mathsf{r}_n \lhd t_n) \overset{\beta_1}{\rightsquigarrow} \cdots \overset{\beta_n}{\rightsquigarrow} (\mathsf{r}_1 \lhd t_1' \parallel \cdots \parallel \mathsf{r}_n \lhd t_n')$$

such that $\beta_n = \mathsf{q} \to \mathsf{r_k} : (\ell, v)$ and such that $g_k \overset{\mathsf{q}?(\ell,v)}{\rightsquigarrow} g'$ for some ℓ, v, g'.

VII. Asynchronous message passing

Up until now, we have only considered synchronous message passing. In this section, we show briefly how to adapt $\lambda_{\texttt{SafeMP}}$ to asynchronous message passing.

Our first task is to replace the synchronous reduction relation $t \overset{\alpha}{\rightsquigarrow}_{\mathsf{a}} t'$ with an asynchronous one. Previous work on asynchronous MPST (e.g. [8]) defines asynchronous reduction by keeping a queue of messages. Here we introduce a simpler approach to represent the same semantics: sending to one participant q does not block either sending to or receiving from another participant p. Our asynchronous reduction relation $t \overset{\alpha}{\rightsquigarrow}_{\mathsf{a}} t'$ has the same form as our synchronous one, and includes all of the same rules in the inductive definition, except the addition of the following two extra rules:

$$\frac{t \overset{\mathsf{p}!(\ell,v)}{\rightsquigarrow}_{\mathsf{a}} t' \qquad \mathsf{p} \neq \mathsf{q}}{\textbf{send}\,(\ell, w)\ \textbf{to}\ \mathsf{q}\ \textbf{then}\ t \overset{\mathsf{p}!(\ell,v)}{\rightsquigarrow}_{\mathsf{a}} \textbf{send}\,(\ell, w)\ \textbf{to}\ \mathsf{q}\ \textbf{then}\ t'}$$

$$\frac{t \overset{\mathsf{p}?(\ell,v)}{\rightsquigarrow}_{\mathsf{a}} t' \qquad \mathsf{p} \neq \mathsf{q}}{\textbf{send}\,(\ell, w)\ \textbf{to}\ \mathsf{q}\ \textbf{then}\ t \overset{\mathsf{p}?(\ell,v)}{\rightsquigarrow}_{\mathsf{a}} \textbf{send}\,(\ell, w)\ \textbf{to}\ \mathsf{q}\ \textbf{then}\ t'}$$

To prove the same properties for $\lambda_{\texttt{SafeMP}}$, we first simplify internal choices $\oplus_{i\in I}\mathsf{p}!\ell_i\langle \mathsf{b}_i\rangle.\, g$ with equal continuations. For asynchronous type reduction $g \overset{\alpha}{\rightsquigarrow}_{\mathsf{a}} g'$, we add two extra rules to the synchronous ones, mirroring the extra rules for asynchronous computation reduction.

$$\frac{g \overset{\mathsf{p}!(\ell,v)}{\rightsquigarrow}_{\mathsf{a}} g' \qquad \mathsf{p} \neq \mathsf{q}}{\oplus_{i\in I}\mathsf{q}!\ell_i\langle \mathsf{b}_i\rangle.\, g \overset{\mathsf{p}!(\ell,v)}{\rightsquigarrow}_{\mathsf{a}} \oplus_{i\in I}\mathsf{q}!\ell_i\langle \mathsf{b}_i\rangle.\, g'}$$

$$\frac{g \overset{\mathsf{p}?(\ell,v)}{\rightsquigarrow}_{\mathsf{a}} g' \qquad \mathsf{p} \neq \mathsf{q}}{\oplus_{i\in I}\mathsf{q}!\ell_i\langle \mathsf{b}_i\rangle.\, g \overset{\mathsf{p}?(\ell,v)}{\rightsquigarrow}_{\mathsf{a}} \oplus_{i\in I}\mathsf{q}!\ell_i\langle \mathsf{b}_i\rangle.\, g'}$$

We then define the asynchronous reduction $G \overset{\beta}{\rightsquigarrow}_{\mathsf{a}} G'$ of global types; we obtain this by replacing [Perm] with the following slightly more general rule.

$$\frac{G_i \overset{\mathsf{p}\to\mathsf{q}:(\ell,v)}{\rightsquigarrow}_{\mathsf{a}} G_i' \text{ for each } i \qquad \mathsf{q}' \notin \{\mathsf{p}, \mathsf{q}\}}{(\mathsf{p}' \to \mathsf{q}' : \{\ell_i\langle \mathsf{b}_i\rangle.\, G_i\}_{i\in I}) \overset{\mathsf{p}\to\mathsf{q}:(\ell,v)}{\rightsquigarrow}_{\mathsf{a}} (\mathsf{p}' \to \mathsf{q}' : \{\ell_i\langle \mathsf{b}_i\rangle.\, G_i'\}_{i\in I})}$$

Communication safety and deadlock-freedom (Theorem IV.4), and liveness (Theorem VI.3) can be extended smoothly to the asynchronous semantics $\overset{\alpha}{\rightsquigarrow}_{\mathsf{a}}$ of computations.

Theorem VII.1 Assume that $\cdot \fatsemi \cdot \vdash t : B \fatsemi h$. Then if $t \overset{\tau}{\rightsquigarrow}_{\mathsf{a}}^{*} t'$, either $t' = \textbf{return}\ v$, or there is some t'' such that $t' \overset{\tau}{\rightsquigarrow}_{\mathsf{a}} t''$.

For liveness, we use the same **tick** adaptations as before.

Theorem VII.2 Let $t_1, \ldots, t_n$ be computations satisfying $\Phi, \mathsf{r}_1, \ldots, \mathsf{r}_n \fatsemi \cdot \vdash t_i : \mathsf{b}_i \fatsemi g_i$, and assume that $\mathsf{Conf}_G(h; (\mathsf{r}_1 \lhd g_1 \parallel \cdots \parallel \mathsf{r}_n \lhd g_n))$ holds. If $g_k \overset{\mathsf{q}!}{\rightsquigarrow}$, then there is a finite sequence of reductions

$$(\mathsf{r}_1 \lhd t_1 \parallel \cdots \parallel \mathsf{r}_n \lhd t_n) \overset{\beta_1}{\rightsquigarrow}_{\mathsf{a}} \cdots \overset{\beta_n}{\rightsquigarrow}_{\mathsf{a}} (\mathsf{r}_1 \lhd t_1' \parallel \cdots \parallel \mathsf{r}_n \lhd t_n')$$

such that $\beta_n = \mathsf{r_k} \to \mathsf{q} : (\ell, v)$ and $g_k \overset{\mathsf{q}!(\ell,v)}{\rightsquigarrow}_{\mathsf{a}} g'$ for some ℓ, v, g'. A similar result holds if $g_k \overset{\mathsf{q}?}{\rightsquigarrow}$.

VIII. Related and future work

a) Effects and session types: Orchard and Yoshida [30] show that one can encode *binary* session-typed π-calculus in graded variant of PCF, the session types being encoded as grades, and vice-versa. Their motivation is proposing encodability results, and their work does not study safety, deadlock-freedom and liveness properties. There are few other works that approach message passing from the perspective of computational effects. Sanada [31] uses message passing to exemplify *category-graded effect handlers*. The work does not give a concurrent language, and does not use grades to enforce liveness or deadlock-freedom.

There are various works on monadic models of concurrency via monads, concentrating on shared state [32]–[34], but not message passing. The models presented in these works bear little resemblance to ours: they are designed to model shared memory concurrency, while ours relies on the insight that, for our calculus, we do not need to model concurrency.

Marshall and Orchard [35] take the linear logic perspective on binary synchronous session types, and use grades to track linearity. They observe that communication primitives are computational effects, but do not use grades to track them directly, and therefore has left deadlock-freedom open (cf., future work in Section 11 in [35]). Our work go deeply into the connection between grading and session types, and use grades to guarantee three desired properties, including deadlock-freedom and liveness.

b) Semantics of session types: Starting from the work in [36], [37], strong foundations of session types have been developed based on linear logic, exploring Curry-Howard correspondence between linear logic and typed session π-calculi [1], [38], [39]. The connection to linear logic ensures various desirable properties, such as type-safety and deadlock freedom for free. The most effective and advanced semantics of these calculi is grounded on *logical relations*, which are incorporated into various programming language features including parametricity [40] and higher-order functions [41]. Among them, built on [9], the recent work [42] enables to track cyclic dependencies of channels based on classical logic session types and applies it to information flow analysis; and the work in [43] develops logical relations based on instuitionistic linear logic enriched with temporal predicates. Notice that semantics foundations on session types developed based on logical relations so far are limited into *binary*.

The work [44] introduces MPGV, a functional language with multiparty session types, based on the linear logic view of session types. They use separation logic to define configuration invariants to maintain the acyclic nature of the communication topology and to prove the subject reduction theorem They support *session delegation*, which we leave to future work. Their work does not aim to develop (denotational) semantics interpretations of programs.

The works in [2], [3] give interpretations of asynchronous and synchronous multiparty sessions as Flow Event Structures and prove that if a multiparty session N is typable by global type G, then their interpretations are equivalent. Their multiparty session corresponds to a set of local types in our framework, and does not contain standard programming constructs such as conditionals, expressions or higher-order functions unlike ours. Moreover, their semantics is entirely *intensional*; with the interpretation of a session being an event structure that records all of the communication within a session. We give an extensional semantics, where the communication within an event structure is hidden. None of above works develops MPST processes from the perspective of computational effects, and ours is the only extensional semantics we know of.

In a different line of the work, hinted by a connection between linear logics and game semantics, the work in [10] proposes fully abstract game semantics for binary session typed processes. They have left the extension to asynchrony and multiparty open. Our focus is to give *extensional* denotational semantics characterisation of *multiparty interactions*, which subsumes core synchronous and asynchronous MPST features studied in [6], [8].

REFERENCES

[1] L. Caires and F. Pfenning, "Session types as intuitionistic linear propositions," in *Proceedings of CONCUR 2010*, ser. LNCS, vol. 6269. Springer, 2010, pp. 222–236.

[2] I. Castellani, M. Dezani-Ciancaglini, and P. Giannini, "Global types and event structure semantics for asynchronous multiparty sessions," *Fundam. Informaticae*, vol. 192, no. 1, pp. 1–75, 2024. [Online]. Available: https://doi.org/10.3233/FI-242188

[3] ——, "Event structure semantics for multiparty sessions," *J. Log. Algebraic Methods Program.*, vol. 131, p. 100844, 2023. [Online]. Available: https://doi.org/10.1016/j.jlamp.2022.100844

[4] K. Honda, N. Yoshida, and M. Carbone, "Multiparty asynchronous session types," *Journal of the ACM*, vol. 63, no. 1, pp. 9:1–9:67, 2016.

[5] N. Yoshida and L. Gheri, "A very gentle introduction to multiparty session types," in *Distributed Computing and Internet Technology*, D. V. Hung and M. D´Souza, Eds. Cham: Springer International Publishing, 2020, pp. 73–93.

[6] S. Ghilezan, S. Jakšić, J. Pantović, A. Scalas, and N. Yoshida, "Precise subtyping for synchronous multiparty sessions," *Journal of Logical and Algebraic Methods in Programming*, vol. 104, pp. 127–173, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352220817302237

[7] A. Scalas and N. Yoshida, "Less is more: Multiparty session types revisited," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–29, Jan. 2019.

[8] S. Ghilezan, J. Pantović, I. Prokić, A. Scalas, and N. Yoshida, "Precise subtyping for asynchronous multiparty sessions," *ACM Trans. Comput. Logic*, vol. 24, no. 2, nov 2023. [Online]. Available: https://doi.org/10.1145/3568422

[9] S. Balzer, F. Derakhshan, R. Harper, and Y. Yao, "Logical relations for session-typed concurrency," 2023. [Online]. Available: https://arxiv.org/abs/2309.00192

[10] S. Castellan and N. Yoshida, "Two sides of the same coin: session types and game semantics: a synchronous side and an asynchronous side," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3290340

[11] S. Katsumata, "Parametric effect monads and semantics of effect systems," in *Proc. of 41st Ann. ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*. New York: ACM Press, 2014, pp. 633–645.

[12] D. Orchard, V.-B. Liepelt, and H. Eades III, "Quantitative program reasoning with graded modal types," *Proc. ACM Program. Lang.*, vol. 3, no. ICFP, Jul. 2019. [Online]. Available: https://doi.org/10.1145/3341714

[13] F. Borceux, G. Janelidze, and G. M. Kelly, "Internal object actions," *Comment. Math. Univ. Carolin.*, vol. 46, no. 2, pp. 235–255, 2005.

[14] A. Smirnov, "Graded monads and rings of polynomials," *J. Math. Sci.*, vol. 151, no. 3, pp. 3032–3051, 2008.

[15] P.-A. Melliès, "Parametric monads and enriched adjunctions," Manuscript, 2012. [Online]. Available: https://www.irif.fr/~mellies/tensorial-logic/8-parametric-monads-and-enriched-adjunctions.pdf

[16] P. B. Levy, J. Power, and H. Thielecke, "Modelling environments in call-by-value programming languages," *Information and Computation*, vol. 185, no. 2, pp. 182–210, 2003.

[17] G. Plotkin, "Call-by-name, call-by-value and the λ-calculus," *Theoretical Computer Science*, vol. 1, no. 2, pp. 125–159, 1975. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0304397575900171

[18] K. Honda, N. Yoshida, and M. Carbone, "Multiparty asynchronous session types," in *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '08. New York, NY, USA: Association for Computing Machinery, Jan. 2008, pp. 273–284.

[19] G. Plotkin and M. Pretnar, "Handlers of algebraic effects," in *European Symposium on Programming*. Springer, 2009, pp. 80–94.

[20] N. Yoshida and P. Hou, "Less is more revisited," 2024, Accepted by Cliff B. Jones Festschrift Proceeding.

[21] S. Ghilezan, S. Jakšić, J. Pantović, A. Scalas, and N. Yoshida, "Precise subtyping for synchronous multiparty sessions," *Journal of Logical and Algebraic Methods in Programming*, vol. 104, pp. 127–173, Apr. 2019.

[22] G. M. Kelly, "A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on," *Bull. Austral. Math. Soc.*, vol. 22, no. 1, pp. 1–83, 1980.

[23] D. S. Scott, "Continuous lattices," *Toposes, algebraic geometry and logic*, vol. 274, pp. 97–136, 1972.

[24] M. B. Smyth and G. D. Plotkin, "The category-theoretic solution of recursive domain equations," *SIAM Journal on Computing*, vol. 11, no. 4, pp. 761–783, 1982.

[25] O. Kammar, "Algebraic theory of type-and-effect systems," Ph.D. dissertation, University of Edinburgh, UK, 2014.

[26] E. Moggi, "Computational lambda-calculus and monads," in *Proc. of 4th Annual IEEE Symposium on Logic in Computer Science, LICS '89*. IEEE, 1989, pp. 14–23.

[27] E. G. Manes, "Monads of sets," in *Handbook of algebra*. Elsevier, 2003, vol. 3, pp. 67–153.

[28] A. Kock, "Strong functors and monoidal monads," *Archiv der Mathematik*, vol. 23, no. 1, pp. 113–120, 1972.

[29] H. Nakano, "A modality for recursion," in *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No. 99CB36332)*. IEEE, 2000, pp. 255–266.

[30] D. Orchard and N. Yoshida, "Effects as sessions, sessions as effects," in *POPL 2016*. ACM, 2016.

[31] T. Sanada, "Category-graded algebraic theories and effect handlers," *Electronic Notes in Theoretical Informatics and Computer Science*, vol. 1, 2023.

[32] N. Benton, M. Hofmann, and V. Nigam, "Effect-dependent transformations for concurrent programs," in *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming*, 2016, pp. 188–201.

[33] Y. Dvir, O. Kammar, and O. Lahav, "A denotational approach to release/acquire concurrency," in *European Symposium on Programming*. Springer, 2024, pp. 121–149.

[34] E. Rivas and T. Uustalu, "Concurrent monads for shared state," in *Proceedings of the 26th International Symposium on Principles and Practice of Declarative Programming*, 2024, pp. 1–13.

[35] D. Marshall and D. Orchard, "Non-linear communication via graded modal session types," *Information and Computation*, vol. 301, p.

105234, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0890540124000993

[36] K. Honda, "Types for dyadic interaction," in *CONCUR'93*, E. Best, Ed. Berlin, Heidelberg: Springer, 1993, pp. 509–523.

[37] K. Honda, V. T. Vasconcelos, and M. Kubo, "Language Primitives and Type Discipline for Structured Communication-Based Programming," in *Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, ser. Lecture Notes in Computer Science, C. Hankin, Ed., vol. 1381. Springer, 1998, pp. 122–138.

[38] B. Toninho, L. Caires, and F. Pfenning, "Higher-order processes, functions, and sessions: A monadic integration," in *Programming Languages and Systems*, M. Felleisen and P. Gardner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 350–369.

[39] P. Wadler, "Propositions as sessions," in *Proceedings of ICFP 2012*. ACM, 2012, pp. 273–286. [Online]. Available: http://doi.acm.org/10.1145/2364527.2364568

[40] L. Caires, J. A. Pérez, F. Pfenning, and B. Toninho, "Behavioral polymorphism and parametricity in session-based communication," in *ESOP*, ser. LNCS, vol. 7792. Springer, 2013, pp. 330–349.

[41] B. Toninho, L. Caires, and F. Pfenning, "Higher-order processes, functions, and sessions: A monadic integration," in *Programming Languages and Systems*, M. Felleisen and P. Gardner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 350–369.

[42] B. van den Heuvel, F. Derakhshan, and S. Balzer, "Information Flow Control in Cyclic Process Networks," in *38th European Conference on Object-Oriented Programming (ECOOP 2024)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), J. Aldrich and G. Salvaneschi, Eds., vol. 313. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, pp. 40:1–40:30. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECOOP.2024.40

[43] Y. Yao, G. Iraci, C.-E. Chuang, S. Balzer, and L. Ziarek, "Semantic logical relations for timed message-passing protocols (extended version)," 2024, to appear in POPL'25. [Online]. Available: https://arxiv.org/abs/2411.07215

[44] J. Jacobs, S. Balzer, and R. Krebbers, "Multiparty GV: functional multiparty session types with certified deadlock freedom," *Proc. ACM Program. Lang.*, vol. 6, no. ICFP, pp. 466–495, 2022. [Online]. Available: https://doi.org/10.1145/3547638

[45] A. W. Appel and D. McAllester, "An indexed model of recursive types for foundational proof-carrying code," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 23, no. 5, pp. 657–683, 2001.

APPENDIX

A. Session types

Proposition III.8 If $G \upharpoonright \mathsf{r}$ is defined, then $G \downarrow_{\mathsf{r}} (G \upharpoonright \mathsf{r})$.

Proof. Let $G \downarrow'_{\mathsf{r}} h$ hold when the projection $G \upharpoonright \mathsf{r}$ is defined, and $h = G \upharpoonright \mathsf{r}$. Then it is enough to show that $\downarrow'_{\mathsf{r}}$ satisfies the three conditions of Definition III.7, with $\mathsf{R} = \{\mathsf{r}\}$.

For (1), consider the reduction $G \overset{\mathsf{p}\to\mathsf{q}:(\ell,v)}{\rightsquigarrow} G'$. We proceed by a case split on whether one of the participants p or q is r. If $\mathsf{p} = \mathsf{r}$, then we need to show that, if $G \upharpoonright \mathsf{r}$ exists, then so does $G' \upharpoonright \mathsf{r}$, and we have $G \upharpoonright \mathsf{r} \overset{\mathsf{q}!(\ell,v)}{\rightsquigarrow} G' \upharpoonright \mathsf{r}$. This is by induction on the derivation of the global type reduction; for a [MSG] reduction the result is trivial, for [PERM] it follows from the fact that reduction of local types is closed under merging, and for [REC] it is trivial again. The case where $\mathsf{q} = \mathsf{r}$ is similar. Finally, if $\mathsf{r} \notin \{\mathsf{p}, \mathsf{q}\}$, then we need to show that if $G \upharpoonright \mathsf{r}$ exists, then so does $G' \upharpoonright \mathsf{r}$, and we have $G \upharpoonright \mathsf{r} \leq^{\&} G' \upharpoonright \mathsf{r}$. This is again an induction on the derivation of the global type reduction; for [MSG] we use the fact that $\bigsqcap_{i\in I} g_i \leq^{\&} g_i$, for [PERM] we use the fact that $\leq^{\&}$ is closed under merging, and for [REC] this is trivial.

For (2), note that the $\alpha = \tau$ case is trivial. If $\alpha = \mathsf{p}?(\ell, v)$ or $\alpha = \mathsf{p}!(\ell, v)$, then we proceed by induction on the session type reduction. If it is by unfolding recursion, then we consider the form of G. If G is a recursive type, then we may unfold G, and then proceed using the inductive hypothesis. Otherwise, G has the form $\mathsf{p}' \to \mathsf{q}' \colon \{\ell_i\langle \mathsf{b}_i\rangle . G_i\}_{i\in I}$, and we necessarily have $\mathsf{r} \notin \{\mathsf{p}', \mathsf{q}'\}$. If $\alpha = \mathsf{p}?(\ell, v)$, then there is some i such that $G_i \upharpoonright \mathsf{r}$ reduces with action α, so result is trivial by taking the reduction $G \overset{\mathsf{p}'\to\mathsf{q}':(\ell_i,v)}{\rightsquigarrow} G_i$ for arbitrary v and using the inductive hypothesis. If $\alpha = \mathsf{p}!(\ell, v)$, then if every G_i makes the required reduction in one step, then we can merge these reductions by rule [PERM]. If some G_i requires more than one reduction, then some communication $\mathsf{p}'' \to \mathsf{q}''$, with $\mathsf{p} \in \{\mathsf{p}'', \mathsf{q}''\}$, is blocking the communication $\mathsf{r} \to \mathsf{p}$. Since the projection $G \upharpoonright \mathsf{p}$ is defined, and is given by merging the projections $G_i \upharpoonright \mathsf{p}$, the same communication can be found in every G_i, and is blocking $\mathsf{r} \to \mathsf{p}$ in every case. Thus every sequence of reductions must involve one that blocks $\mathsf{r} \to \mathsf{p}$. The other possibility for the session type reduction $h \overset{\alpha}{\rightsquigarrow} h'$ is a base case. Again we inspect G, which necessarily has the form $\mathsf{p}' \to \mathsf{q}' \colon \{\ell_i\langle \mathsf{b}_i\rangle . G_i\}_{i\in I}$. If $\mathsf{r} \notin \{\mathsf{p}', \mathsf{q}'\}$, then we use similar reasoning to the above; otherwise we can conclude with a single step reduction.

(3) follows directly from the fact that $\{\mathsf{r}\}$ is a singleton. □

Lemma A.1 If $\mathsf{Conf}_G(h; \mathsf{r}_1 \lhd g_1, \ldots, \mathsf{r}_n \lhd g_n)$ and $G \overset{\beta}{\rightsquigarrow} G'$, then there exist $g'_1, \ldots, g'_n$ and $h' \leq^{\oplus} h$ such that $\mathsf{Conf}_{G'}(h'; \mathsf{r}_1 \lhd g'_1, \ldots, \mathsf{r}_n \lhd g'_n)$, and, for all i, if $\beta \upharpoonright \mathsf{r}_i = \tau$ then $g_i \leq^{\&} g'_i$, otherwise, $g_i \overset{\beta\upharpoonright \mathsf{r}_i}{\rightsquigarrow} g'_i$.

Proof. This is direct from Definition III.7(1). □

B. Enforcing safety in $\lambda_{\mathtt{SafeMP}}$

Lemma A.2 Assume $\mathsf{Conf}_G(h; \mathsf{r}_1 \lhd g_1, \ldots, \mathsf{r}_n \lhd g_n)$. If $\mathbf{end} <: g_i$ for all i, then $h = \mathbf{end}$.

Proof. If $h \neq \mathbf{end}$, then we would have either $h \overset{\mathsf{p}!}{\rightsquigarrow}$ or $h \overset{\mathsf{p}?}{\rightsquigarrow}$ for some p. Since we assume conformance, Definition III.7(2) implies there is a sequence of reductions, starting from G, in which at least one of the actions is not τ. Thus $G \overset{\mathsf{q}\to\mathsf{q}'}{\rightsquigarrow}$ for some participants q and q', at least one of which has to be r_i for some i. But then, using Definition III.7(1), we have either $g_i \overset{\mathsf{q}'!}{\rightsquigarrow}$ or $g_i \overset{\mathsf{q}?}{\rightsquigarrow}$. Both of these contradict $\mathbf{end} <: g_i$. □

Lemma IV.2 Assume $\mathsf{Conf}_G(h; \mathsf{r}_1 \lhd g_1, \ldots, \mathsf{r}_n \lhd g_n)$.

1) If $g_j \overset{\mathsf{r}_k!(\ell,v)}{\rightsquigarrow} g'_j$ and $g_k \overset{\mathsf{r}_j?}{\rightsquigarrow}$, then there is some G' such that $G \overset{\mathsf{r}_j\to\mathsf{r}_k:(\ell,v)}{\rightsquigarrow} G'$.
2) If $g_j \overset{\mathsf{p}!(\ell,v)}{\rightsquigarrow} g'_j$, with $\mathsf{p} \notin \{\mathsf{r}_1, \ldots, \mathsf{r}_n\}$, then there is some G' such that $G \overset{\mathsf{r}_j\to\mathsf{p}:(\ell,v)}{\rightsquigarrow} G'$.
3) If $h \overset{\mathsf{p}?(\ell,v)}{\rightsquigarrow} h'$ and $g_k \overset{\mathsf{p}?}{\rightsquigarrow}$, with $\mathsf{p} \notin \{\mathsf{r}_1, \ldots, \mathsf{r}_n\}$, there is some G' such that $G \overset{\mathsf{p}\to\mathsf{r}_k:(\ell,v)}{\rightsquigarrow} G'$.

Proof. For (1), since g_j reduces, there are global type reductions

$$G \overset{\mathsf{p}_1\to\mathsf{q}_1:(\ell_1,v_1)}{\rightsquigarrow} \cdots \overset{\mathsf{r}_j\to\mathsf{r}_k:(\ell,v)}{\rightsquigarrow} G'$$

We may have $m = 1$, in which case we are done. If there were no reduction with $m = 1$, then either r_j or r_k must appear at some step $i < m$. But since we also have $g_k \overset{\mathsf{r}_j?}{\rightsquigarrow}$, this cannot be the case, so we do indeed have the required reduction.

The proof of (2) is similar, except that we employ Definition III.7(3), and the fact that all communication in G must involve a participant from R, to establish that $m = 1$.

The proof of (3) is again similar. □

Lemma A.3 (Inversion)

1) If $\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} \mathbf{rec}\ f\, x.\, t : B$ then there exist A, A', g, κ such that $B = A \xrightarrow{g} A'$ and $\Phi \fatsemi \Gamma, f : A \xrightarrow{g} A', x : A \vdash t : A' \fatsemi g$.
2) If $\Phi \fatsemi \Gamma \vdash \mathbf{return}\ v : B \fatsemi h$ then $\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} v : B$ and $\mathbf{end} <: h$.
3) If $\Phi \fatsemi \Gamma \vdash \mathbf{let}\ x = t\ \mathbf{in}\ u : B \fatsemi h$ then there exists A, h', h'' such that $\Phi \fatsemi \Gamma \vdash t : A \fatsemi h'$, $\Phi \fatsemi \Gamma \vdash t : B \fatsemi h''$, and $h' \cdot h'' <: h$.
4) If $\Phi \fatsemi \Gamma \vdash v - w : B \fatsemi h$ then $\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} v : \mathbf{int}$, $\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} w : \mathbf{int}$, $B = \mathbf{int}$, and $\mathbf{end} <: h$.
5) If $\Phi \fatsemi \Gamma \vdash v < w : B \fatsemi h$ then $\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} v : \mathbf{int}$, $\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} w : \mathbf{int}$, $B = \mathbf{bool}$ and $\mathbf{end} <: h$.
6) If $\Phi \fatsemi \Gamma \vdash \mathbf{if}\ v\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 : B \fatsemi h$, then $\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} v : \mathbf{bool}$, $\Phi \fatsemi \Gamma \vdash t_1 : B \fatsemi h$, $\Phi \fatsemi \Gamma \vdash t_2 : B \fatsemi h$.
7) If $\Phi \fatsemi \Gamma \vdash w\, v : B \fatsemi h$ then there exists A, g such that $\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} w : A \xrightarrow{g} B$, $\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} v : A$, and $g <: h$.
8) If $\Phi \fatsemi \Gamma \vdash \mathbf{send}\,(\ell, v)\ \mathbf{to}\ \mathsf{p}\ \mathbf{then}\ t : B \fatsemi h$ then there exists b, g such that $\Phi \fatsemi \Gamma \vdash t : A \fatsemi g$ and $(\mathsf{p}!\ell\langle \mathsf{b}\rangle.\, g) <: h$.
9) If $\Phi \fatsemi \Gamma \vdash \mathbf{recv}\ \mathsf{p}\,\{(\ell_i, x_i).\, t_i\}_{i\in I} : B \fatsemi h$ then there exist $\{\mathsf{b}_i\}_{i\in I}$ and $\{g_i\}_{i\in I}$ such that $\Phi \fatsemi \Gamma, x_i : \mathsf{b}_i \vdash t_i : B \fatsemi g_i$ for all $i \in I$, and $(\&_{i\in I}\mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, g_i) <: h$.
10) If $\Phi \fatsemi \Gamma \vdash \mathbf{let}\ x_1 = \mathsf{p}_1 \lhd t_1 \parallel \cdots \parallel x_n = \mathsf{p}_n \lhd t_n\ \mathbf{in}\ u : B \fatsemi h$ then there exist $\mathsf{b}_1, \ldots, \mathsf{b}_n$, $g_1, \ldots, g_n$, h', h'' and G, such that

$$\Phi, p_1, \ldots, p_n \fatsemi \Gamma \vdash t_i : \mathsf{b}_i \fatsemi g_i \text{ for each } i$$
$$\Phi \fatsemi \Gamma, x_1 : \mathsf{b}_1, \ldots, x_n : \mathsf{b}_n \vdash u : B \fatsemi h''$$
$$\mathsf{Conf}_G(h'; (\mathsf{p}_1 \lhd g_1 \parallel \cdots \parallel \mathsf{p}_n \lhd g_n)) \qquad h' \cdot h'' <: h$$

Proof. Each case is an easy induction on the typing derivation. □

Lemma A.4 (Substitution) Assume that $\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} v_i : A_i$ for each i.

1) If $\Phi \fatsemi x_1 \colon A_1, \ldots, x_n : A_n \vdash^{\mathrm{v}} w : B$, then $\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} w[x_i \mapsto v_i]_i : B$.
2) If $\Phi \fatsemi x_1 \colon A_1, \ldots, x_n : A_n \vdash t : B \fatsemi g$, then $\Phi \fatsemi \Gamma \vdash t[x_i \mapsto v_i]_i : B \fatsemi g$.

Proof. By mutual induction on the typing derivations. □

Lemma IV.1 (Subject reduction) If $\Phi \fatsemi \cdot \vdash t : B \fatsemi h$ and $t \overset{\alpha}{\rightsquigarrow} t'$, then for all h' such that $h \overset{\alpha}{\rightsquigarrow} h'$, we have $\Phi \fatsemi \cdot \vdash t' : B \fatsemi h'$. Moreover:

- if $\alpha = \tau$ or $\alpha = \mathsf{p}!(\ell, v)$, then there exists such an h';
- if $\alpha = \mathsf{p}?(\ell, v)$, then $h \overset{\mathsf{p}?}{\rightsquigarrow}$.

Proof. By induction on the derivation of the reduction, using the inversion lemma (Lemma A.3) to obtain the typing of subterms. We omit the proofs of the trivial cases.

- For [LetR], we have the following by inversion,

$$\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} v : A \qquad \Phi \fatsemi \Gamma, x : A \vdash u : B \fatsemi g \qquad g <: h$$

  By Lemma A.4, we therefore have

$$\Phi \fatsemi \Gamma \vdash u[x \mapsto v] : B \fatsemi g$$

  which implies the required result by subtyping.
- For [LetR], we have the following by inversion.

$$\Phi \fatsemi \Gamma \vdash t : A \fatsemi h' \qquad \Phi \fatsemi \Gamma, x : A \vdash u : B \fatsemi h'' \qquad h' \cdot h'' <: h$$

  The inductive hypothesis tells us that $\Phi \fatsemi \Gamma \vdash t' : A \fatsemi g'$, for all reductions $h' \overset{\alpha}{\rightsquigarrow} g'$. It follows that $h' \cdot h'' \overset{\alpha}{\rightsquigarrow} g' \cdot h''$, and hence that $h \overset{\alpha}{\rightsquigarrow} g$ for some $(g' \cdot g'') :> g$. Moreover, for every reduction $h \overset{\alpha}{\rightsquigarrow} g$ we obtain $(g' \cdot g'') :> g$. We then conclude that $\Phi \fatsemi \Gamma \vdash \mathbf{let}\ x = t\ \mathbf{in}\ u : B \fatsemi g$ as required.
- For [Rec], we have the following by inversion.

$$\Phi \fatsemi f : A \xrightarrow{g} B, x : A \vdash u : B \fatsemi g \quad \Phi \fatsemi \cdot \vdash^{\mathrm{v}} v : A \quad g <: h$$

  We obtain $\Phi \fatsemi \cdot \vdash u[f \mapsto (\mathbf{rec}\ f\, x.\, u), x \mapsto v] : B \fatsemi g$ from Lemma A.4, and hence also $\Phi \fatsemi \cdot \vdash u[f \mapsto (\mathbf{rec}\ f\, x.\, u), x \mapsto v] : B \fatsemi h$ because $g <: h$.
- For [Recv], we have the following by inversion.

$$\mathsf{p} \in \Phi \qquad \Phi \fatsemi x_i : \mathsf{b} \vdash t_i : A \fatsemi g_i \text{ for all } i \qquad (\&_{i\in I}\mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, g_i) <: h$$

We have $h \overset{\mathsf{p}?}{\rightsquigarrow}$ trivially. If $h \overset{\mathsf{p}?(\ell_j,v)}{\rightsquigarrow} h'$, then we have $h' :> g_j$. Lemma A.4 tells us that $\Phi \fatsemi \cdot \vdash t_j[x_j \mapsto v] : A \fatsemi g_j$, so $\Phi \fatsemi \cdot \vdash t_j[x_j \mapsto v] : A \fatsemi h'$ by subgrading.

- For [Send], we have the following by inversion.

$$\Phi \fatsemi \cdot \vdash^{\mathrm{v}} v : \mathsf{b} \qquad \mathsf{p} \in \Phi \qquad \Phi \fatsemi \cdot \vdash t : B \fatsemi g \qquad \mathsf{p}!\ell\langle \mathsf{b} \rangle . g <: h$$

Since $(\mathsf{p}!\ell\langle g \rangle .) \overset{\mathsf{p}!}{\rightsquigarrow}$, we have $h \overset{\mathsf{p}!}{\rightsquigarrow}$, and there is some $h' :> g$ such that $h \overset{\mathsf{p}!(\ell,v)}{\rightsquigarrow} h'$. We conclude that $\Phi \fatsemi \cdot \vdash t : B \fatsemi h'$ by subgrading.

- For [TRet], we have the following by inversion.

$$\Phi \fatsemi \cdot \vdash^{\mathrm{v}} v_i : \mathsf{b}_i \text{ for each } i$$
$$\Phi \fatsemi x_1 : \mathsf{b}_1, \ldots, x_n : \mathsf{b}_n \vdash u : B \fatsemi h''$$
$$\mathsf{Conf}_\Phi(h'; (\mathsf{p}_1 \lhd g_1 \parallel \cdots \parallel \mathsf{p}_n \lhd g_n))$$

for some $g_i :>$ **end** and $h' \cdot h'' <: h$. We have **end** $<: h'$ by Lemma A.2, so that $h'' <: h$ and $\Phi \fatsemi \cdot \vdash u[x_i \mapsto v_i]_[i] : B \fatsemi h$.

- For [Par], we have the following by inversion.

$$\Phi, \mathsf{p}_1, \ldots, \mathsf{r}_n \fatsemi \cdot \vdash t_i : \mathsf{b}_i \fatsemi g_i \text{ for each } i$$
$$\Phi \fatsemi x_1 : \mathsf{b}_1, \ldots, x_n : \mathsf{b}_n \vdash u : B \fatsemi h''$$
$$\mathsf{Conf}_G(h'; (\mathsf{r}_1 \lhd g_1 \parallel \cdots \parallel \mathsf{r}_n \lhd g_n)) \qquad (h' \cdot h'') <: h$$

There are two cases to consider. If the reduction of the session is by rule [Com], then by the inductive hypothesis, there is a grade g_j' such that $g_j \overset{\mathsf{p}_k!(\ell,v)}{\rightsquigarrow} g_j'$, and $\Phi \fatsemi \cdot \vdash t_j' : B_j \fatsemi g_j'$, and we also have $g_k \overset{\mathsf{p}_j?}{\rightsquigarrow}$. By Lemma IV.2, we therefore have a global type reduction $G \overset{\mathsf{r}_j \to \mathsf{r}_k:(\ell,v)}{\rightsquigarrow} G'$. It follows that there are grades g_i' and a grade $h''' \leq^{\oplus} h'$, such that

$$g_k \overset{\mathsf{p}_j?(\ell,v)}{\rightsquigarrow} g_k' \qquad g_i' \geq^{\&} g_i \ (i \notin \{j,k\}) \qquad \mathsf{Conf}_\Phi(h'''; \mathsf{p}_1 \lhd g_1' \parallel \cdots \parallel \mathsf{p}_n \lhd g_n')$$

By the inductive hypothesis again, we have $\Phi \fatsemi \cdot \vdash t_j' : B_j \fatsemi g_j'$, and by subgrading, also $\Phi \fatsemi \cdot \vdash t_i' : B_i \fatsemi g_i'$ for $i \notin \{j,k\}$. Hence, since $h''' \cdot h'' <: h' \cdot h'' <: h$, we have the desired result.

If instead the reduction is by [SPar], we split into three cases.

- If $\alpha = \tau$, then the result is immediate from the inductive hypothesis.
- If $\alpha = \mathsf{p}!(\ell, v)$, with $\mathsf{p} \notin \{\mathsf{r}_1, \ldots, \mathsf{r}_n\}$ then we have $g_j \overset{\alpha}{\rightsquigarrow} g_j'$ and $\Phi \fatsemi \cdot \vdash t_j' : \mathsf{b}_j \fatsemi g_j'$ by the inductive hypothesis. Hence by Lemma IV.2, we have a global type reduction $G \overset{\mathsf{r}_j \to \mathsf{p}:(\ell,v)}{\rightsquigarrow} G'$. The rest of the proof is as in the [Com] case.
- If $\alpha = \mathsf{p}?(\ell, v)$, then the proof is similar to the previous case, except that now we have $h \overset{\alpha}{\rightsquigarrow} h'$, and it follows that $g_j \overset{\alpha}{\rightsquigarrow} g_j'$. □

Lemma IV.3 If $\Phi \fatsemi \cdot \vdash t : B \fatsemi h$, then either (i) there is a reduction $t \overset{\tau}{\rightsquigarrow} t'$, or (ii) one of the following holds.

1) $h =$ **end**, and $t =$ **return** v for some $v : B$.
2) $h \overset{\mathsf{p}!}{\rightsquigarrow}$ for some p, and there exists a message (ℓ, v) such that there are reductions $h \overset{\mathsf{p}!(\ell,v)}{\rightsquigarrow} h'$ and $t \overset{\mathsf{p}!(\ell,v)}{\rightsquigarrow} t'$.
3) $h \overset{\mathsf{p}?}{\rightsquigarrow}$ for some p, and for every message (ℓ, v) such that there is a reduction $h \overset{\mathsf{p}?(\ell,v)}{\rightsquigarrow} h'$, there is a reduction $t \overset{\mathsf{p}?(\ell,v)}{\rightsquigarrow} t'$.

Proof. By induction on the typing derivation for t. Again we omit the easy cases.

- For [Let], in the computation **let** $x = u$ **in** u', the computation u is either a **return**, or it reduces. In the former case we can conclude by using the reduction rule [LetR]; in the latter case by using [Let].
- For [Snd], the computation has the form **send** (ℓ, v) **to** p **then** u; the message (ℓ, v) is precisely what we need to conclude.
- For [Rcv], the computation has the form **recv** $\mathsf{p}\,\{(\ell_i, x_k).\, u_i\}_{i \in I}$, and the messages (ℓ, v) such that $h \overset{\mathsf{p}?(\ell,v)}{\rightsquigarrow} h'$ are precisely those such that $\ell = \ell_i$ and $v : \mathsf{b}_i$ for some i. The computation t can make a reduction labelled by every such message.
- For [Par], if any of the computations t_i reduce by a τ action, then we can conclude using rules [SPar] and [Par]. This is not the case, then we look at the global type G that witnesses conformance:

$$\mathsf{Conf}_G(h; \mathsf{r}_1 \lhd g_1, \ldots, \mathsf{r}_n \lhd g_n)$$

If $G = \mathbf{end}$, then the grades of the computations t_i are all **end**, and so each of the computations t_i is a **return**. We can therefore conclude using rule [TRET]. If instead G has the form $\mathsf{p} \to \mathsf{q}\colon \{\ell_i\langle \mathsf{b}_i\rangle.\, G_i\}_{i\in I}$, then we split into three cases.

1) If p and q are both in $\{\mathsf{r}_1, \ldots, \mathsf{r}_n\}$, say $\mathsf{p} = \mathsf{r}_j$ and $\mathsf{q} = \mathsf{r}_k$, then we know by Definition III.7(1) that $g_j \overset{\mathsf{r}_k!}{\rightsquigarrow}$. By the inductive hypothesis, there is a message (ℓ_i, v) such that $t_j \overset{\mathsf{r}_k!(\ell,v)}{\rightsquigarrow} t'_j$. We also have $G \overset{\mathsf{r}_j\to\mathsf{r}_k:(\ell_i,v)}{\rightsquigarrow} G_i$, so that $g_k \overset{\mathsf{r}_j?(\ell,v)}{\rightsquigarrow} g'_k$, and by the inductive hypothesis, $t_k \overset{\mathsf{r}_k?(\ell,v)}{\rightsquigarrow} t'_k$. We conclude using rules [PAR] and [COM].
2) If $\mathsf{p} \in \{\mathsf{r}_1, \ldots, \mathsf{r}_n\}$, say $\mathsf{p} = \mathsf{r}_j$, and $\mathsf{q} \notin \{\mathsf{r}_1, \ldots, \mathsf{r}_n\}$, then we have that $g_j \overset{\mathsf{q}!}{\rightsquigarrow}$, so that there is some (ℓ, v) such that $t_j \overset{\mathsf{q}!(\ell,v)}{\rightsquigarrow} t'_j$. We conclude using rules [PAR] and [SPAR].
3) The case where $\mathsf{p} \notin \{\mathsf{r}_1, \ldots, \mathsf{r}_n\}$ and $\mathsf{q} \in \{\mathsf{r}_1, \ldots, \mathsf{r}_n\}$ is similar.

Finally, if G instead has the form $\mu\mathbf{t}.\, G'$, then we inspect $G'[\mathbf{t} \mapsto \mu\mathbf{t}.\, G']$. □

Theorem IV.4 Assume that $\cdot \fatsemi \cdot \vdash t : B \fatsemi h$. Then if $t \overset{\tau}{\rightsquigarrow}{}^* t'$, either $t' = \mathbf{return}\ v$, or there is some t'' such that $t' \overset{\tau}{\rightsquigarrow} t''$.

Proof. We have $\cdot \fatsemi \cdot \vdash t' : B \fatsemi h$ by Lemma IV.1, and the result immediately follows from Lemma IV.3. □

C. Denotational semantics

Lemma V.5 Assume that $\mathsf{Conf}_G(h; \mathsf{r}_1 \lhd g_1, \ldots, \mathsf{r}_n \lhd g_n))$. If $t_i \in \mathrm{GSR}_{\Phi,\mathsf{r}_1,\ldots,\mathsf{r}_n}(X_i)_{g_i}$ for each i, then the set

$$\{u \mid (t_1, \ldots, t_n) \succ u\}$$

has a least upper bound $\mathrm{fork}(t_1, \ldots, t_n)$, and this least upper bound is in $\mathrm{GSR}_\Phi(\prod_i X_i)_h$. These least upper bounds form a continuous function

$$\mathrm{fork}\colon \textstyle\prod_i \mathrm{GSR}_{\Phi,\mathsf{r}_1,\ldots,\mathsf{r}_n}(X_i)_{g_i} \to \mathrm{GSR}_\Phi(\prod_i X_i)_h$$

such that $\mathrm{fork}(t_1, \ldots, t_n) \geq u$ is equivalent to $(t_1, \ldots, t_n) \succ u$.

Proof. We first note that, if a derivation of $(t_1, \ldots, t_n) \succ u$ involves any of the rules [SEND], [RECV], [INT], then one can obtain session type reductions for the grades of the computations t_i, and then use Lemma IV.2 to obtain a global type reduction. These reductions are crucial in the proof.

If the only u in the set is $\bot$, then the least upper bound is $\bot$. Otherwise, the least upper bound is the same as the least upper bound of the non-empty set $S(t_1, \ldots, t_n) = \{u \neq \bot \mid (t_1, \ldots, t_n) \succ u\}$. Let u_0 be an arbitrary element of $S(t_1, \ldots, t_n)$; we split into cases based on the form of u_0.

- If $u_0 = \mathrm{return}(x'_1, \ldots, x'_n)$, then we have $t_i = \mathrm{return}(x_i)$, with $x_i \leq x'_i$, for each i. It is easy to see that every element of $S(t_1, \ldots, t_n)$ has the form $\mathrm{return}(x''_1, \ldots, x''_n)$ with $x''_i \geq x'_i$. The least upper bound is therefore $\mathrm{return}(x_1, \ldots, x_n)$.
- If $u_0 = \mathrm{send}_{p,(\ell,v)} u'_0$ for some u'_0, then we proceed by induction on the derivation of $(t_1, \ldots, t_n) \succ u_0$. There are only two rules that apply. If the derivation is by the rule [SEND], then we have $t_j = \mathrm{send}_{\mathsf{p},(\ell,v)} t'_j$ for some t'_j. We claim that the least upper bound is $\mathrm{send}_{\mathsf{p},(\ell,v)} u'$, where u' is the least upper bound of $\{u \neq \bot \mid (\ldots, t_{j-1}, t'_j, t_{j+1}, \ldots) \succ u\}$. To see that it is an upper bound, one take an arbitrary $u_1 \in S(t_1, \ldots, t_n)$, and looks at the derivation. The derivation clearly cannot involve [RETURN], but nor can it involve [SEND] or [RECV], except for the t_j case of [SEND] we are considering. The reason for this is that, if there were such a derivation involving a conflicting [SEND] or [RECV], then one would have conflicting global type reductions that violate Definition III.7(3).
  If instead the derivation is by the rule [INT] then we have $t_j = \mathrm{send}_{\mathsf{r}_k,(\ell,v)}(t'_j)$ and $t_k = \mathrm{recv}_{\mathsf{r}_j,\{\ell_i:\mathsf{b}_i\}}(t''_{j,m})_m$, and the least upper bound is the same as that of $S(t'_1, \ldots, t'_n)$, where $t'_i = t_i$ for $i \notin \{j, k\}$ and $t'_j = t''_{j,(\ell,v)}$.
- If $u_0 = \mathrm{recv}_{p,(\ell,v)} u'_0$ for some u'_0, then we reason analogously to the previous case.

One can then read off the continuous function fork from the rules for $\succ$, as least prefixed point. □

Lemma V.8 (Soundness) Let $t \overset{\alpha}{\rightsquigarrow} t'$ be a reduction, starting from a computation t such that $\Phi \fatsemi \cdot \vdash t : B \fatsemi h$. If $[\![t]\!] = \bot$, then $[\![t']\!] = \bot$; otherwise, we have the following.

1) If $\alpha = \tau$, then $[\![t]\!] = [\![t']\!]$.
2) If $\alpha = \mathsf{p}!(\ell, v)$, then $[\![t]\!] = \mathrm{send}_{\mathsf{p},(\ell,[\![v]\!])}[\![t']\!]$.
3) If $\alpha = \mathsf{p}?(\ell, v)$, then $[\![t]\!] = \mathrm{recv}_{\mathsf{p},\{\ell_i:\mathsf{b}_i\}_i}([\![t'_m]\!])_m$, where $\mathcal{M}(\ell_i : \mathsf{b}_i)_{i\in I}$ consists of those m such that there is some t'_m satisfying $t \overset{\mathsf{p}?(\ell,v)}{\rightsquigarrow} t'_m$.

Proof. The proof is by induction on the derivation of the reduction. We again omit the easy cases.

- For [LETR], we use the left unit law of the graded monad, which is immediate from the definition of $\gg\!=$.

- For [LET], the case where $\alpha = \tau$ is trivial. For the other two actions however, we use the equations $(\mathrm{send}_{\mathsf{p},(\ell,v)} t) \gg\!=^{Z}_{X} f = \mathrm{send}_{\mathsf{p},(\ell,v)}(t \gg\!=^{Z}_{X} f)$ and $(\mathrm{recv}_{\mathsf{p},\{\ell_i:\mathsf{b}_i\}_i}(t_m)_m) \gg\!=^{Z}_{X} f = \mathrm{recv}_{\mathsf{p},\{\ell_i:\mathsf{b}_i\}_i}(t_m \gg\!=^{Z}_{X} f)_m$.
- For [SEND], and [RECV], we simply read off the interpretations of the computations.
- For [TRET], we use the definition of $\succ$, specifically, the rule [RETURN], which implies that $\mathrm{fork}_{(\mathrm{return} x_1,\ldots,\mathrm{return} x_n)} = \mathrm{return}(x_1,\ldots,x_n)$. We then use the left unit law of the graded monad.
- For [PAR], the session can reduce either by rule [COM] or by rule [SPAR]. If the reduction is by [PAR], then we obtain $[\![t_j]\!] = \mathrm{send}_{\mathsf{r}_k,(\ell,v)} [\![t'_j]\!]$ and $[\![t_k]\!] = \mathrm{recv}_{\mathsf{r}_j,\{\ell_i:\mathsf{b}_i\}_i} \left[\!\!\left[t'_{k,m}\right]\!\!\right]$, with $(\ell, v) \in \mathcal{M}(\{\ell_i : \mathsf{b}_i\}_i)$, from the inductive hypothesis. It follows that $\mathrm{fork}_{[\![t_1]\!],\ldots,[\![t_n]\!]}$ is given by the [INT] rule of $\succ$. If the reduction is by [SPAR], then, we split into three cases. If the action is $\alpha = \tau$, the result is immediate from the induction hypothesis. If the action is $\alpha = \mathsf{p}!(\ell, v)$, then we have $[\![t_j]\!] = \mathrm{send}_{\mathsf{p},(\ell,[\![v]\!])}$, and $\mathrm{fork}_{[\![t_1]\!],\ldots,[\![t_n]\!]}$ is given by the [SEND] rule of $\succ$. If the action is $\alpha = \mathsf{p}?(\ell, v)$, then $\mathrm{fork}_{[\![t_1]\!],\ldots,[\![t_n]\!]}$ is similarly given by the [RECV] rule of $\succ$. To complete the latter two cases, we again use the defining equations for $\gg\!=$. □

D. Enforcing liveness

The proof of liveness is by a *logical relations* argument, where the logical relation captures the reduction behaviour of computations of a given type and grade. The logical relation we use here consists of a subset $\mathcal{R}_\kappa [\![B]\!]_g$ of computations, for each natural number κ, type B, and closed session type g; along with a subset $\mathcal{R}_\kappa [\![A]\!]_{\mathsf{v}}$ of values, for each κ and A. The natural number κ, plays a similar role to the natural number used in *step-indexed* logical relations proofs [45], which are typically used for reasoning about recursive types. The idea is that the logical relation, at step κ, only provides a guarantee about what happens until κ many messages have been sent.

The definition of the logical relation is as follows. For values, the definition is by recursion on the type, while for values, it is by recursion on the grade. The case for recursive types $\mu \mathbf{t}.\, g$ is well-founded because we require g to be a **tick**, $\oplus$, or $\&$.

$$\mathcal{R}_\kappa [\![\mathsf{b}]\!]_{\mathsf{v}} = \{v \mid v : \mathsf{b}\}$$
$$\mathcal{R}_\kappa \left[\!\!\left[A \xrightarrow{g} B \right]\!\!\right]_{\mathsf{v}} = \{w \mid \forall \kappa' \leq \kappa, v \in \mathcal{R}_{\kappa'} [\![A]\!]_{\mathsf{v}}.\, (w\, v) \in \mathcal{R}_{\kappa'} [\![B]\!]_g\}$$

$$\mathcal{R}_\kappa [\![A]\!]_{\mathbf{end}} = \{t \mid \exists v \in \mathcal{R}_\kappa [\![A]\!]_{\mathsf{v}}.\, t \overset{\tau}{\rightsquigarrow}{}^{*}\ \mathbf{return}\ v\}$$
$$\mathcal{R}_\kappa [\![A]\!]_{\oplus_{i\in I} \mathsf{q}!\ell_i\langle \mathsf{b}_i\rangle.\, h_i} = \{t \mid \exists i \in I, v : \mathsf{b}, t' \in \mathcal{R}_\kappa [\![A]\!]_{\mathbf{tick}\ h_i}.\, t \overset{\mathsf{q}!(\ell_i,v)}{\rightsquigarrow}{}^{*}\ t'\}$$
$$\mathcal{R}_\kappa [\![A]\!]_{\&_{i\in I} \mathsf{q}?\ell_i\langle \mathsf{b}_i\rangle.\, h_i} = \{t \mid \forall i \in I, v : \mathsf{b}.\, \exists t' \in \mathcal{R}_\kappa [\![A]\!]_{\mathbf{tick}\ h_i}.\, t \overset{\mathsf{q}?(\ell_i,v)}{\rightsquigarrow}{}^{*}\ t'\}$$
$$\mathcal{R}_\kappa [\![A]\!]_{\mu \mathbf{t}.\, g} = \mathcal{R}_\kappa [\![A]\!]_{g[\mathbf{t} \mapsto \mu \mathbf{t}.\, g]}$$
$$\mathcal{R}_\kappa [\![A]\!]_{\mathbf{tick}\ g} = \textstyle\bigcap_{\kappa' < \kappa} \mathcal{R}_{\kappa'} [\![A]\!]_g$$

It is useful to have a little more machinery. Define the *depth-κ unfolding* $\mathrm{unf}_\kappa g$ of a session type g by $\mathrm{unf}_0 g = g$ and

$$\mathrm{unf}_{\kappa+1}\, \mathbf{end} = \mathbf{end}$$
$$\mathrm{unf}_{\kappa+1}(\oplus_{i\in I} \mathsf{p}!\ell_i\langle \mathsf{b}_i\rangle.\, g_i) = \oplus_{i\in I} \mathsf{p}!\ell_i\langle \mathsf{b}_i\rangle.\, (\mathrm{unf}_\kappa g_i)$$
$$\mathrm{unf}_{\kappa+1}(\&_{i\in I} \mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, g_i) = \&_{i\in I} \mathsf{p}?\ell_i\langle \mathsf{b}_i\rangle.\, (\mathrm{unf}_\kappa g_i)$$
$$\mathrm{unf}_{\kappa+1}(\mu \mathbf{t}.\, g) = \mathrm{unf}_{\kappa+1}(g[\mathbf{t} \mapsto \mu \mathbf{t}.\, g])$$

This is well-defined because we permit only guarded recursion in local types. We clearly have $g <: \mathrm{unf}_\kappa g$ and $\mathrm{unf}_\kappa g <: g$, and also that $\mathcal{R}_{\kappa'} [\![A]\!]_g = \mathcal{R}_{\kappa'} [\![A]\!]_{\mathrm{unf}_\kappa g}$. There is a simlar definition of $\mathrm{unf}_\kappa G$ for global types.

We also note that our subject reduction lemma carries over to our modified type system:

Lemma A.5 If $\Phi \fatsemi \cdot \vdash t : B \fatsemi h$ and $t \overset{\alpha}{\rightsquigarrow} t'$, then for all h' such that $h \overset{\alpha}{\rightsquigarrow} h'$, we have $\Phi \fatsemi \cdot \vdash t' : B \fatsemi h'$. Moreover:

- if $\alpha = \tau$ or $\alpha = \mathsf{p}!(\ell, v)$, then there exists such an h';
- if $\alpha = \mathsf{p}?(\ell, v)$, then $h \overset{\mathsf{p}?}{\rightsquigarrow}$.

Proof. The proof is essentially the same as that of Lemma IV.1, except that one has to insert **tick** in a few of the cases. We will give the proof for the rule [REC], since this requires special attention.

$$(\mathbf{rec}\ f\, x.\, u)\, v \overset{\tau}{\rightsquigarrow} u[f \mapsto (\mathbf{rec}\ f\, x.\, u), x \mapsto v]$$

We have the following by inversion.

$$\Phi \fatsemi f : A \xrightarrow{\mathsf{tick}\ g} B, x : A \vdash u : B \fatsemi g \qquad \Phi \fatsemi \cdot \vdash^{\mathrm{v}} v : A \qquad \mathbf{tick}^{\kappa}\ g <: h$$

We obtain $\Phi \fatsemi \cdot \vdash u[f \mapsto (\mathbf{rec}\ f\, x.\, u), x \mapsto v] : B \fatsemi g$ from by the substitution lemma, and hence also $\Phi \fatsemi \cdot \vdash u[f \mapsto (\mathbf{rec}\ f\, x.\, u), x \mapsto v] : B \fatsemi h$ because $g <: \mathbf{tick}^{\kappa}\ g <: h$. □

Lemma A.6

1) If $\kappa' \leq \kappa$, then $\mathcal{R}_{\kappa'}[\![A]\!]_{\mathrm{v}} \supseteq \mathcal{R}_{\kappa}[\![A]\!]_{\mathrm{v}}$ and $\mathcal{R}_{\kappa'}[\![A]\!]_{g} \supseteq \mathcal{R}_{\kappa}[\![A]\!]_{g}$.
2) If $g <: h$, then $\mathcal{R}_{\kappa}[\![A]\!]_{g} \subseteq \mathcal{R}_{\kappa}[\![A]\!]_{h}$.
3) If $t \overset{\tau}{\rightsquigarrow} t'$ and $t' \in \mathcal{R}_{\kappa}[\![A]\!]_{g}$, then $t \in \mathcal{R}_{\kappa}[\![A]\!]_{g}$.

Proof. We prove each of these inductively, by showing that, if the result holds for all $\kappa' < \kappa$, then the result holds for κ.

For (1) the proof is by mutual induction on A (for values) and g (for computations); every case is trivial. For (2), we note that instead of $g <: h$, we can consider $\mathrm{unf}_{\kappa} g <: \mathrm{unf}_{\kappa} h$. The proof is then by inspecting the root of the derivation of $\mathrm{unf}_{\kappa} g <: \mathrm{unf}_{\kappa} h$. The proof of (3) is an easy induction on g. □

We use the logical relation to define a *semantic* typing judgment $\vDash$, which captures the the behaviour of a computation as it reduces, to go along with the syntactic judgment $\vdash$.

Definition A.7 Let Φ be a participant context, $\Gamma = z_1 : A_1, \ldots, z_n : A_n$ be a typing context, and B be a type.

1) We write $\Phi \fatsemi \Gamma \vDash^{\mathrm{v}}_{\kappa} w : B$ when, for every tuple of values $v_i \in \mathcal{R}_{\kappa}[\![A_i]\!]_{\mathrm{v}}$, we have $w[z_i \mapsto v_i]_i \in \mathcal{R}_{\kappa}[\![B]\!]_{\mathrm{v}}$. When this holds for every κ, we write $\Phi \fatsemi \Gamma \vDash^{\mathrm{v}} w : B$.
2) We write $\Phi \fatsemi \Gamma \vDash_{\kappa} t : B \fatsemi h$ when, for every tuple of values $v_i \in \mathcal{R}_{\kappa}[\![A_i]\!]_{\mathrm{v}}$, we have $t[z_i \mapsto v_i]_i \in \mathcal{R}_{\kappa}[\![B]\!]_{g}$. When this holds for every κ, we write $\Phi \fatsemi \Gamma \vDash t : B \fatsemi h$.

The important fact is that, if a computation is well-typed according to our syntactic typing relation $\vdash$, then it is semantically well-typed; this is the *fundamental lemma* of our logical relation.

Lemma A.8 (Fundamental lemma)

1) If $\Phi \fatsemi \Gamma \vdash^{\mathrm{v}} w : B$, then $\Phi \fatsemi \Gamma \vDash^{\mathrm{v}} w : B$.
2) If $\Phi \fatsemi \Gamma \vdash t : B \fatsemi h$, then $\Phi \fatsemi \Gamma \vDash t : B \fatsemi h$.

Proof. We show, for each κ, that all of the semantic typing judgments hold at κ, by induction on κ. So assuming they hold for all $\kappa' < \kappa$, we need to show they hold at κ.

The proof of this is by mutual induction on the typing derivations. Again most of the cases are easy (and standard from the logical relations literature). We give only the interesting cases, and write σ for the substitution $(z_i \mapsto v_i)_i$ that appears in the definition of the semantic typing relation.

- For the rule [<:] we use Lemma A.6(2).
- For the rule [Let], the inductive hypothesis tells us that the computation $t[\sigma]$ reduces in finitely many steps to a computation of the form $\mathrm{return}\, w$, and thus the **let** reduces to $u[\sigma][x \mapsto w]$. By the inductive hypothesis again, the latter computation is in the logical relation, and so the original let binding is by Lemma A.6(3).
- The [Snd] and [Rcv] cases are both easy.
- For [Par], we let $\mathsf{R} = \{\mathsf{r}_1, \ldots, \mathsf{r}_n\}$ be the set of participants that appear in the session, and inspect the global type G that witnesses the conformance predicate $\mathsf{Conf}_G(h; (\mathsf{r}_1 \lhd g_1, \ldots, \mathsf{r}_n \lhd g_n))$.
  - If $G = \mathbf{end}$, then $g_i = \mathbf{end}$ for all i and $h = \mathbf{end}$. Each of the computations $t_i[\sigma]$ reduce to a term of the form $\mathbf{return}\ w_i$, and thus the parallel **let** further reduces to $u[\sigma][x_i \mapsto w_i]_i$. The result then follows from the inductive hypothesis.
  - If G has the form $\mathbf{tick}\ G'$, the result follows from the inductive hypothesis.
  - If G has the form $\mathsf{p} \to \mathsf{q} \colon \{\ell_i\langle \mathsf{b}_i \rangle.\, G_i\}_{i \in I}$, then we do a case split.
    - If $\mathsf{p}, \mathsf{q} \in \mathsf{R}$, say $\mathsf{p} = \mathsf{r}_j$ and $\mathsf{q} = \mathsf{r}_k$ then, by applying the inductive hypothesis to t_j and t_k we obtain reductions for those computations. Hence the session reduces, by rule [Com]. There is also a global type reduction $G \overset{\mathsf{p} \to \mathsf{q} : (\ell, v)}{\rightsquigarrow} \mathbf{tick}\ G'$, and the result of reducing the session conforms to the global type $\mathbf{tick}\ G'$. The result then follows from the inductive hypothesis, because G' is guarded by **tick**
    - The cases where only one of p and q are in R are similar; one again can reduce the session, but this time using [SPar], and one obtains a global type reduction $G \overset{\mathsf{p} \to \mathsf{q} : (\ell, v)}{\rightsquigarrow} \mathbf{tick}\ G'$.
  - If G is a recursive type $\mu \mathbf{t}.\, G'$, then consider instead its unfolding $G'[\mathbf{t} \mapsto \mu \mathbf{t}.\, G']$. □

Lemma VI.2 Assume that t satisfies $\Phi \fatsemi \cdot \vdash t : B \fatsemi h$.

1) If $h = \mathbf{tick}^\kappa\ \mathbf{end}$, then there exists t' such that $t \overset{\tau}{\rightsquigarrow}{}^* t'$.
2) If $h \overset{p!}{\rightsquigarrow}$, then for some reduction $h \overset{\mathsf{p}!(\ell,v)}{\rightsquigarrow} h'$, there exists t' such that $t \overset{\mathsf{q}!(\ell_i,v)}{\rightsquigarrow}{}^* t'$.
3) If $h \overset{\mathsf{p}?}{\rightsquigarrow}$, then for every reduction $h \overset{\mathsf{p}?(\ell,v)}{\rightsquigarrow} h'$, there exists t' such that $t \overset{\mathsf{q}?(\ell_i,v)}{\rightsquigarrow}{}^* t'$.

Proof. By Lemma A.8 we have $\Phi \fatsemi \cdot \vDash t : B \fatsemi g$. All three cases then follow from this. □

Theorem VI.3 (Liveness) Let $t_1, \ldots, t_n$ be computations satisfying $\Phi, \mathsf{r}_1, \ldots, \mathsf{r}_n \fatsemi \cdot \vdash t_i : \mathsf{b}_i \fatsemi g_i$, and assume that $\mathsf{Conf}_G(h; (\mathsf{r}_1 \lhd g_1 \parallel \cdots \parallel \mathsf{r}_n \lhd g_n))$ holds.

1) If $g_k \overset{\mathsf{q}!}{\rightsquigarrow}$, then there is a finite sequence of reductions

   $(\mathsf{r}_1 \lhd t_1 \parallel \cdots \parallel \mathsf{r}_n \lhd t_n) \overset{\beta_1}{\rightsquigarrow} \cdots \overset{\beta_n}{\rightsquigarrow} (\mathsf{r}_1 \lhd t_1' \parallel \cdots \parallel \mathsf{r}_n \lhd t_n')$

   such that $\beta_n = \mathsf{r}_k \to \mathsf{q} : (\ell, v)$ and $g_k \overset{\mathsf{q}!(\ell,v)}{\rightsquigarrow} g'$ for some ℓ, v, g'.
2) If $g_k \overset{\mathsf{q}?}{\rightsquigarrow}$, then there is a finite sequence of reductions

   $(\mathsf{r}_1 \lhd t_1 \parallel \cdots \parallel \mathsf{r}_n \lhd t_n) \overset{\beta_1}{\rightsquigarrow} \cdots \overset{\beta_n}{\rightsquigarrow} (\mathsf{r}_1 \lhd t_1' \parallel \cdots \parallel \mathsf{r}_n \lhd t_n')$

   such that $\beta_n = \mathsf{q} \to \mathsf{r}_k : (\ell, v)$ and such that $g_k \overset{\mathsf{q}?(\ell,v)}{\rightsquigarrow} g'$ for some ℓ, v, g'.

Proof. We give only the proof for (1); the proof of (2) is similar. Since $g_k \overset{\mathsf{q}!}{\rightsquigarrow}$, we have $g_k <: \mathbf{tick}^\kappa\ (\oplus_{i \in I} \mathsf{q}!\ell_i\langle \mathsf{b}_i \rangle.\, g_i')$. We show, for all $\kappa' \leq \kappa$, that if H witnesses conformance and $\mathrm{unf}_{\kappa-\kappa'} H = \mathbf{tick}^{\kappa-\kappa'}\ G$, then we have the required reductions. The statement in the lemma follows by taking $\kappa' = \kappa$. We proceed by induction on κ'.

When $\kappa' = 0$, we know that $\mathrm{unf}_{\kappa+1} G$ has the form $\mathbf{tick}^\kappa\ (\mathsf{r}_k \to \mathsf{q} : \{\ell_i\langle \mathsf{b}_i \rangle.\, G_i'\}_{i \in I})$, since otherwise, we would not have $g_k <: \mathbf{tick}^\kappa\ (\oplus_{i \in I} \mathsf{q}!\ell_i\langle \mathsf{b}_i \rangle.\, g_i')$. The result therefore follows from conformance and Lemma VI.2.

If $\kappa > 0$, then we split into cases, based on the form of $\mathrm{unf}_1 G$.

- $\mathrm{unf}_1 G$ cannot be **end**, because this would contradict the assumption about g_k.
- If $\mathrm{unf}_1 G = \mathbf{tick}\ G'$, then $\mathrm{unf}_{\kappa-(\kappa'-1)} H = \mathbf{tick}^{\kappa-(\kappa'-1)}\ (G')$, and the result follows from the inductive hypothesis for $\kappa' - 1$.
- If $\mathrm{unf}_1 G = \mathsf{p} \to \mathsf{p}' : \{\ell_j\langle \mathsf{b}_j \rangle.\, G_j'\}_{j \in J}$, then $r_k \notin \{\mathsf{p}, \mathsf{p}'\}$, since otherwise, this would contradict the assumption about g_k. Hence by conformance and Lemma VI.2, we have a reduction

  $$(\mathsf{r}_1 \lhd t_1 \parallel \cdots \parallel \mathsf{r}_n \lhd t_n) \overset{\beta}{\rightsquigarrow}{}^* (\mathsf{r}_1 \lhd u_1 \parallel \cdots \parallel \mathsf{r}_n \lhd u_n)$$

  and there is a global type H' witnessing conformance of $(\mathsf{r}_1 \lhd u_1 \parallel \cdots \parallel \mathsf{r}_n \lhd u_n)$. The latter moreover satisfies $\mathrm{unf}_1 H' = \mathbf{tick}\ G_j'$ for some j, so that $\mathrm{unf}_{\kappa-(\kappa'-1)} H' = \mathbf{tick}^{\kappa-(\kappa-1)}\ G_j'$, and we have $g_k <: \mathbf{tick}^\kappa\ (G_j' \upharpoonright \mathsf{r}_k)$. Hence the result follows from the inductive hypothesis for $\kappa' - 1$. □

E. Asynchronous message passing

The main change we need to make to the previous proofs, to account for asynchronous message passing, is to add the two extra cases to the proof of subject reduction.

Lemma A.9 (Subject reduction) If $\Phi \fatsemi \cdot \vdash t : B \fatsemi h$ and $t \overset{\alpha}{\rightsquigarrow}_{\mathsf{a}} t'$, then for all h' such that $h \overset{\alpha}{\rightsquigarrow}_{\mathsf{a}} h'$, we have $\Phi \fatsemi \cdot \vdash t' : B \fatsemi h'$. Moreover:

- if $\alpha = \tau$ or $\alpha = \mathsf{p}!(\ell, v)$, then there exists such an h';
- if $\alpha = \mathsf{p}?(\ell, v)$, then $h \overset{\mathsf{p}?}{\rightsquigarrow}$.

Proof. The same proof as before (Lemma IV.1), by induction on the reduction, works here. We only need to add the two extra cases. Both of them are in fact very simple, and they are similar to each other. One simply applies the inductive hypothesis to the assumption of the rule to determine that the grade g of the continuation reduces; the grade $\mathsf{p}!\ell\langle \mathsf{b} \rangle.\, g$ then reduces asynchronously, and thus so does the grade of the computation. □

There is no need to do anything to the proof of our progress result; it does not rely on asynchronous message passing. Safety (Theorem VII.1) then follows as before.