Type Checking Liveness for Collaborative Processes with Bounded and Unbounded Recursion (Full version)

Søren Debois[1], Thomas Hildebrandt[1], Tijs Slaats[1,2], and Nobuko Yoshida[3]

[1] IT University of Copenhagen {debois,hilde,tslaats}@itu.dk
[2] Exformatics A/S
[3] Imperial College yoshida@doc.ic.ac.uk

Abstract. We present the first session typing system guaranteeing response liveness properties for possibly non-terminating communicating processes. The types augment the branch and select types of the standard binary session types with a set of required responses, indicating that whenever a particular label is selected, a set of other labels, its responses, must eventually also be selected. We prove that these extended types are strictly more expressive than standard session types. We provide a type system for a process calculus similar to a subset of collaborative BPMN processes with internal (data-based) and external (event-based) branching, message passing, bounded and unbounded looping. We prove that this type system is sound, i.e., it guarantees request-response liveness for dead-lock free processes. We exemplify the use of the calculus and type system on a concrete example of an infinite state system.

1 Introduction

Session types were originally introduced as typing systems for particular π-calculi, modelling the interleaved execution of some number of two-party protocols. A well-typed process is guaranteed freedom from race-conditions as well as communication compatibility, usually referred to as session fidelity [15,26,24]. Session types have subsequently been the subject of intense study, with much work on applications, typically to programming languages, e.g., [11,17,14,20]. A number of generalisations of the theory has been proposed, notably to multi-party session types [16]. Multi-party session types have a close resemblance to choreographies as found in standards for business process modelling languages such as BPMN [21] and WS-CDL, and has been argued in theory to be able to provide typed BPMN processes [8].

Behavioral types usually furnish *safety* guarantees, notably progress and lock-freedom [3,1,5,10,25]. In contrast, in this paper we extend binary session types to allow specification of *liveness*—the property of a process eventually "doing something good". Liveness properties are usually verified by model-checking techniques [6,2,4], requiring a state-space exploration. In the present paper we

show that a fundamental class of liveness properties, so-called *request-response* properties, can be dealt with by type rules, that is, without resorting to state-space exploration. As a consequence, we can deal statically with infinite state systems as exemplified below. Also, liveness properties specified in types can be understood and used as interface specifications and for compositional reasoning.

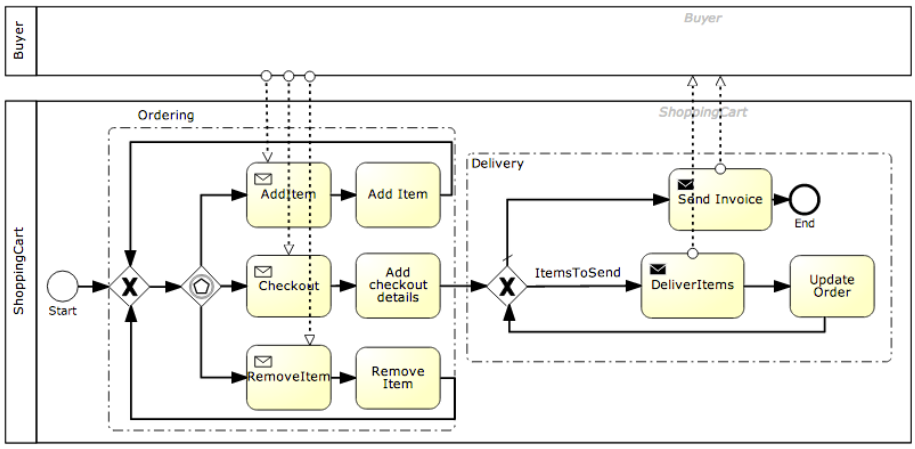

Fig. A. A Potentially Non-live Shopping Cart BPMN Process

As an example, the above diagram contains two pools: The Buyer and the ShoppingCart. Only the latter specifies a process, which has two parts: Ordering and Delivery. Ordering is a loop that starts with an event-based gateway, branching according to the message sent by the customer: messages AddItem and RemoveItem provides an item to be added respectively removed from the order, whereafter the loop repeats. A Checkout message provides delivery details of the order, exits the loop, and proceeds to the Delivery phase. This is again a loop, delivering the ordered items and then sending the invoice.

A buyer who wants to communicate *safely* with the Shopping Cart, must follow the protocol described above, and in particular must be able to receive an unbounded number of items before receiving the invoice. Writing $\mathsf{AI}, \mathsf{RI}, \mathsf{CO}, \mathsf{DI}$, and SI for the actions "Add Items", "Remove Items", "Checkout", "Deliver Items" and "Send Invoice"; we can describe this protocol with a session type:

$$\mu t.\&\{\mathsf{AI}.?.t, \mathsf{RI}.?.t, \mathsf{CO}.?.\mu t'.\oplus\{\mathsf{DI}.!.t', \mathsf{SI}.!.\mathtt{end}\}\} \ .$$

This session type can be regarded as a *behavioral* interface, specifying that the process first expects to receive either an AI (AddItem), RI (RemoveItem) or a CO (CheckOut) event. The two first events must be followed by a message (indicated by "?"), which in the implementation provides the item to be added

or removed, after which the protocol returns to the initial state. The checkout event is followed by a message (again indicated by a "?") after which the protocol enters a new loop, either sending a DI (DeliverItem) event followed by a message (indicated by a "!") and repeating, or sending an SI (SendInvoice) event followed by a message (the invoice) and ending.

However, standard session types can not specify the very relevant *liveness* property, that a CheckOut event is *eventually* followed by an invoice event. This is an example of a so-called *response* property: an action (the request) must be followed by a particular response. In this paper we conservatively extend binary session types to specify such response properties, and we show that this extension is strictly more expressive than standard session types. We do so by annotating the checkout selection in the type with the required response:

$$\mu t.\&\{\mathsf{AI}.?.t, \mathsf{RI}.?.t, \mathsf{CO}[\{\mathsf{SI}\}].?.\mu t'.\oplus\{\mathsf{DI}.!.t', \mathsf{SI}.!.\mathtt{end}\}\}\ .$$

Intuitively: "if CO is selected, then subsequently also SI must be selected."

Determining from the flow graph alone if this response property is guaranteed is in general not possible: Data values dictate whether the second loop terminates. However, we can remove this data-dependency by replacing the loop with a bounded iteration. In BPMN this can be realised by a Sequential Multiple Instance Subprocess, which sequentially executes a (run-time determined) number of instances of a sub-process. With this, we may re-define Delivery as in Fig. B, yielding a re-defined Shopping Cart process which has the response property.

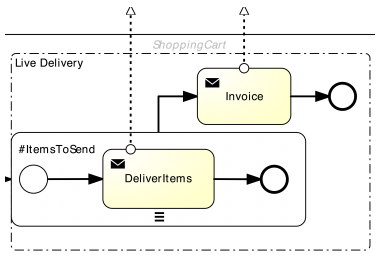

Fig. B. Live delivery with MI Sub-Process

In general, we need also be able to check processes where responses are requested within (potentially) infinite loops. The type system we present gives such guarantees, essentially by collecting all requested responses in a forward analysis, exploiting that potentially infinite loops can guarantee a particular response only if every path through the loop can; and that order (request-response vs response-request) is in this case irrelevant. We prove that, if the system is lock free, then the typing system indeed guarantees that all requested responses are eventually fulfilled. Lock-freedom is needed because, as is well known, collaborative processes with interleaved sessions may introduce dependency locks. Lock-freedom is well-studied for binary session types [3,1,5,10,25], or may alternatively be achieved by resorting to global types [16], the combination with which we leave for future work.

In summary, our contributions are as follows.

- We extend binary session types with a notion of *required response*.
- We prove that this extension induces a strictly more expressive language class than standard session types.

– We give a typing system conservatively extending standard binary session types which gives the further guarantee that a lock-free well-typed process will, in any execution, provide all its required responses.
– We exemplify the use of these extended types to guarantee both safety and liveness properties for a non-trivial, infinite state collaborative process, which exhibits both possibly infinite looping and bounded iteration.

Related work. There is a vast amount of work on verification of collaborative processes. Most of the work take a model-checking approach, where the system under verification is represented as a kind of automaton or Petri Nets. An example that explicitly addresses collaborative business processes is [23], which however does not cover liveness properties. The work on Live Sequence Charts (LSCs) in [6] is a conservative extension of Message Sequence Charts adding support for distinguishing possible (may) from required (must) behaviour and thus specification of liveness properties for collaborating processes. LSCs can be mapped to symbolic timed automata [2] but relies as all model-checking approaches on abstraction techniques for reducing a large or even infinite state space to a tractable size. Here the work in [4] is interesting for the fact that the model-checking can be split on components. The work in [19] allows for model-checking of ML programs by a translation to higher-order recursion schemes. Interestingly, the model-checking problem is reduced to a type-checking problem, but rely on a technique for generation of a specific type system for the property of interest. In contrast, our approach is based on a single type system directly applicable for the process language at hand, where the (less general) liveness and safety properties of interest are specified as the type to be checked and can also be used as interface descriptions of processes. The work on fair sub typing in [22] is the only work on session types addressing liveness we are aware of, which details a liveness-preserving subtyping-relation for session types. In comparison, our approach allows the specification of fine-grained request-response liveness properties—"*if* something happens, something else must happen"—something not allowed by [22].

Overview of this paper. In Sec. 2 we define our calculus and give an LTS-semantics for the calculus. In Sec. 3 we extend binary session types to allow specification of response liveness properties, give a transition semantics for the types, and sketch a proof that these extended types induce a strictly larger class of languages than does standard binary session types. In Sec. 4 we define exactly how our extended session types induce a notion of liveness on processes. In Sec. 5 we give our extended typing rules for sessions with responses and state its subject reduction result. In Sec. 6 we prove that the extended typing rules guarantees liveness for lock-free processes. Finally, in Sec. 7 we conclude.

2 Process Terms and Semantics

Processes communicate only via named communication (session) channels by synchronizing send and receive actions or synchronizing select and branch events

(as in standard session typed π-calculus). The session typing rules presented in the next session guarantees that there is always at most one active send and receive action for a given channel. To distinguish dual ends of communication channels, we employ *polarised names* [13,26]: If c is a channel name, c^+ and c^- are the dual ends of the channel c. We call these *polarised channel names*, with "+" and "-" polarities. If k is a polarised channel name, we write $\overline{k}$ for the dual polarised channel name, e.g., $\overline{c^+} = c^-$. The syntax of processes is given below.

Meta-variables:

c	channel names
p	polarities $+, -$
k, h	polarised channel names (c^p)
x	data variables
v	data values, including natural numbers and $\mathsf{true}, \mathsf{false}$
e	data expressions, including data variables and values
X, Y	process variables

Process syntax:

$$\begin{aligned} P \ ::= \ & k!\langle e\rangle.P \ \mid \ k?(x).P \ \mid \ k!l.P \ \mid \ k?\{l_i.P_i\}_{i\in I} \ \mid \ \mathbf{0} \ \mid \ P \mid Q \\ & \mid \ \mathrm{rec}\, X.P \ \mid \ (\mathrm{rec}^e\, X(i).P; Q) \ \mid \ X[\tilde{k}] \ \mid \ \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \end{aligned}$$

The first four process constructors are for taking part in a communication. These are standard for session typed π-calculi, except that for simplicity of presentation, we only allow data to be sent (see Section 7). The process $k!\langle e\rangle.P$ sends data v over channel k when $e \Downarrow v$, and proceeds as P. Dually, $k?(x).P$ receives a data value over channel k and substitutes it for the x binding in P. A *branch* process $k?\{l_i.P_i\}_{i\in I}$ offers a choice between labels l_i, proceeding to P_i if the i'th label is chosen. The process $\mathbf{0}$ is the standard inactive process (termination), and $P \mid Q$ is the parallel composition of processes P and Q.

Recursion comes in two forms: a general, potentially non-terminating recursion $\mathrm{rec}\, X.P$, where X binds in P; and a primitive recursion, guaranteed to terminate, with syntax $(\mathrm{rec}^e\, X(i).P; Q)$. The latter process, when $e \Downarrow n+1$, executes $P\{n/i\}$ and repeats, and when $e \Downarrow 0$, evolves to Q. We assume the following conventions:

In $(\mathrm{rec}^e\, X(i).P; Q)$, $\mathbf{0}$ does not occur in P.

In $(\mathrm{rec}^e\, X(i).P; Q)$, no process variable but X occurs free in P.

In $(\mathrm{rec}^e\, X(i).P; Q)$, there is no sub-term $\mathrm{rec}\, Y.Q$ or $(\mathrm{rec}^e\, Y(i).Q; R)$ in P.

In $(\mathrm{rec}^e\, X(i).P; Q)$, there is no sub-term $Q \mid R$ of P.

These conventions ensure that the process $(\mathrm{rec}^e\, X(i).P; Q)$ will eventually terminate the loop and execute Q. Process variables $X[\tilde{k}]$ mentions the channel names $\tilde{k}$ active at unfolding time for technical reasons.

We define the free polarised names $\mathsf{fn}(P)$ of a process P as usual, with $\mathsf{fn}(X[\tilde{k}]) = \tilde{k}$; substitution of process variables from $X[\tilde{k}]\{P/X\} = P$; and

finally value substitution $P\{v/x\}$ in the obvious way, e.g., $k!\langle e\rangle.P\{v/x\} = k!\langle e\{v/x\}\rangle.(P\{v/x\})$. Variable substitution can never affect channels.

Example 2.1. We now show how to model the example BPMN process given in the introduction. To illustrate the possibility of type checking infinite state systems, we use a persistent data object represented by a process $\mathrm{DATA}(o)$ communicating on a session channel o.

$$\mathrm{DATA}(o) = \mathrm{rec}\, X.\ o^+?(x).\ \mathrm{rec}\, Y.\ o^+? \begin{cases} \mathsf{read}.\ o^+!\langle x\rangle.\ Y[o^+] \\ \mathsf{write}.\ X[o^+] \\ \mathsf{quit}.\ \mathbf{0} \end{cases}$$

After having received its initial value, this process repeatedly accepts commands read and write on the session channel o for respectively reading and writing its value, or the command quit for discarding the data object.

To make examples more readable, we employ the following shorthands. We write $\mathsf{init}(o, v).P$ for $o^-!\langle v\rangle.P$, which initializes the data object; we write $\mathsf{free}\, o.P$ for $o^-!\mathsf{quit}..P$, the process which terminates the data object session; we write $\mathsf{read}\, o(x).P$ for $o^-!\mathsf{read}.\ o^-?(x).P.$, the process which loads the value of the data object o into the process-local variable x; and finally, we write $o := e.P$ for $o^-!\mathsf{write}.o^-!\langle e\rangle.P$, the process which sets the value of the data-object o.

The shopping cart process can then be modelled as

$$P(Q) = \mathrm{DATA}(o)\ \mid\ \mathsf{init}(o, \epsilon).\ \mathrm{rec}\, X.k \begin{cases} \mathsf{AI}.\ k?(x).\ \mathsf{read}\, o(y).\ o := add(y, x).\ X[ko^-] \\ \mathsf{RI}.\ k?(x).\ \mathsf{read}\, o(y).\ o := rem(y, x).\ X[ko^-] \\ \mathsf{CO}.\ k?(x).\ \mathsf{read}\, o(y).\ o := add(y, x).\ Q \end{cases}$$

Here k is the session channel shared with the customer and o is the session channel for communicating with the data object modelling order data. We assume our expression language has suitable operators "add" and "rem", adding and removing items from the order. Finally, the process Q is a stand-in for either the (non live) delivery part of the BPMN process in Fig. A or the live delivery part shown in Fig. B. The non-live delivery loop can be represented by the process

$$D_0 = \mathrm{rec}\, Y.\ \mathsf{read}\, o(y).\ \mathsf{if}\ n(y) > 0 \begin{array}{l} \mathsf{then}\ k!\mathsf{DI}.\ k!\langle next(y)\rangle.\ o := update(y).\ Y[ko^-] \\ \mathsf{else}\ k!\mathsf{SI}.\ k!\langle inv(y)\rangle.\ \mathsf{free}\, o.\mathbf{0} \end{array}$$

where $n(y)$ is the integer expression computing from the order y the number of items to send, $next(y)$, $update(y)$ and $inv(y)$ are, respectively, the next item(s) to be sent; an update of the order to mark that these items have indeed been sent; and the invoice for the order. Note that whether or not this process terminates is entirely dependent on the data operations.

Using instead bounded iteration, the live delivery becomes:

$$\begin{aligned} D = \mathsf{read}\, o(y).\ (\mathrm{rec}^{n(y)}\, Y(i).\ & \\ & k!\mathsf{DI}.\mathsf{read}\, o(y).\ k!\langle pickitem(y, i)\rangle.Y[ko^-]; \\ & \qquad k!\mathsf{SI}.\ \mathsf{read}\, o(y).\ k!\langle inv(y)\rangle.\ \mathsf{free}\, o.\mathbf{0}) \end{aligned}$$

(The second line is the body of the loop; the third line is the continuation.) Here $pickitem(y, i)$ is the expression extracting the ith item from the order y. □

Transition Semantics. We give a labelled transition semantics in Fig C. We assume a total evaluation relation $e \Downarrow v$; note the absence of a structural congruence. Transition labels for processes are on one of the following forms.

$$\lambda \ ::= \ k!v \ \mid \ k?v \ \mid \ k \oplus l \ \mid \ k\&l \ \mid \ \tau \ \mid \ \tau : l$$

We assume τ is neither a channel nor a polarised channel. Define $\mathsf{subj}(k!v) = \mathsf{subj}(k?v) = \mathsf{subj}(k\&l) = \mathsf{subj}(k \oplus l) = k$ and $\mathsf{subj}(\tau) = \mathsf{subj}(\tau : l) = \tau$, and define as a technical convenience $\overline{\tau} = \tau$. We use these rules along with symmetric rules

$$\text{[C-Out]} \quad \frac{e \Downarrow v}{k!\langle e\rangle.P \xrightarrow{k!v} P} \ \overline{k} \notin \mathsf{fn}(P) \qquad \text{[C-In]} \quad \frac{}{k?(x).P \xrightarrow{k?v} P\{v/x\}} \ \overline{k} \notin \mathsf{fn}(P)$$

$$\text{[C-Sel]} \quad \frac{}{k!l.P \xrightarrow{k\oplus l} P} \ \overline{k} \notin \mathsf{fn}(P) \qquad \text{[C-Bra]} \quad \frac{}{k?\{l_i.P_i\}_{i\in I} \xrightarrow{k\&l_i} P_i} \ \overline{k} \notin \mathsf{fn}(P)$$

$$\text{[C-ParL]} \quad \frac{P \xrightarrow{\lambda} Q}{P \mid P' \xrightarrow{\lambda} Q \mid P'} \qquad \overline{\mathsf{subj}(\lambda)} \notin \mathsf{fn}(P')$$

$$\text{[C-Com1]} \quad \frac{P \xrightarrow{\overline{k}!v} P' \qquad Q \xrightarrow{k?v} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \text{[C-Com2]} \quad \frac{P \xrightarrow{\overline{k}\oplus l} P' \qquad Q \xrightarrow{k\&l} Q'}{P \mid Q \xrightarrow{\tau:l} P' \mid Q'}$$

$$\text{[C-Rec]} \quad \frac{P\{\mathrm{rec}\, X.P/X\} \xrightarrow{\lambda} Q}{\mathrm{rec}\, X.P \xrightarrow{\lambda} Q}$$

$$\text{[C-PRec0]} \quad \frac{e \Downarrow 0 \qquad Q \xrightarrow{\lambda} R}{(\mathrm{rec}^e\, X(i).P; Q) \xrightarrow{\lambda} R}$$

$$\text{[C-PRecN]} \quad \frac{e \Downarrow n+1 \qquad P\{n/i\}\{(\mathrm{rec}^n\, X(i).P; Q)/X\} \xrightarrow{\lambda} R}{(\mathrm{rec}^e\, X(i).P; Q) \xrightarrow{\lambda} R}$$

$$\text{[C-CondT]} \quad \frac{e \Downarrow \mathsf{true} \qquad P \xrightarrow{\lambda} P'}{\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\lambda} P'} \qquad \text{[C-CondF]} \quad \frac{e \Downarrow \mathsf{false} \qquad Q \xrightarrow{\lambda} Q'}{\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\lambda} Q'}$$

Fig. C. Transition semantics for terms

for [C-ParL] and [C-Com1/2]. Compared to standard CCS or π semantics,

there are two significant changes: (1) In the [C-ParL], a transition λ of P is *not* preserved by parallel composition if the co-channel of the subject of λ is in P'; and (2) in prefix rules, the co-name of the subject cannot appear in the continuation. We impose (1) because if the co-channel of the subject of λ is in P', then $P \mid P'$ does not offer synchronisation on λ to its environment; the synchronisation is offered only to P'. E.g., the process $P = c^+!\langle v\rangle.Q \mid c^-?(x).R$ does not have a transition $c^+!\langle v\rangle.Q \mid c^-?(x).R \xrightarrow{c^+!v} Q \mid c^-?(x).R$. If it had such a transition, no environment U able to receive on c^- could be put in parallel with P and form a well-typed process, since both U and $c^-?(d).R$ would then contain the name c^- free. The reason for (2) is similar: If a process $k!\langle e\rangle.P \xrightarrow{k!v} P$, and P contains $\overline{k}$, again no well-typed environment for that process can contain $\overline{k}$.

In recent papers [24,7,25], session types has been presented not with polarised names, but rather with seemingly disparate names, connected by a new-name operator, e.g., one writes $(\nu xy)(x!l. \mid y?\{l.0\})$ to form a session with endpoints x, y. This latter formulation is—while elegant for reduction semantics—is not viable for a the present transition semantics. Without the ability to recognise the two ends c^+, c^- of a polarised channels as either end of a session, we cannot express the rules [C-Par] nor [C-Com].

Lemma 2.2. *If* $P \xrightarrow{\lambda} Q$ *then* $\overline{\mathsf{subj}(\lambda)} \notin \mathsf{fn}(Q)$.

Proof. Straightforward induction on the derivation of the transition.

3 Session Types with Responses

In this section, we generalise binary session types to *session types with responses*. In addition to providing the standard communication safety properties, these also allow us to specify response liveness properties.

Compared to standard session types, we do not consider delegation (name passing). Firstly, as illustrated by our example calculus, the types are already expressive enough to cover a non-trivial subset of collaborative processes. Secondly, as we show in the end of the section, session types with responses are already strictly more expressive than standard session types with respect to the languages they can express. Thus, as we also address in Sec. 7, admitting delegation and answering the open question about how response obligations can be safely exchanged with the environment, is an interesting direction for future work which is beyond the scope of the present paper.

We first define request/response liveness in the abstract. In general, we shall take it to be the property that "a request is eventually followed by a response". For now, we will not concern ourselves exactly what "requests" and "responses" are or what it means for a responds to fulfil a request.

Definition 3.1. *A* request/response structure *is a tuple* $(A, R, \mathsf{req}, \mathsf{res})$ *where* A *is a set of actions,* R *is a set of responses, and* $\mathsf{req} : A \to R$ *and* $\mathsf{res} : A \to R$ *are maps defining the set of responses requested respectively performed by an action.*

Notation. Request/response liveness is naturally a property of sequences. We write ϵ for the empty string, we let ϕ, ψ range over finite strings, and we let α, β, γ range over finite or infinite sequences. We write sequence concatenation by juxtaposition, i.e., $\phi\alpha$.

Definition 3.2. *Suppose* $(A, R, \mathsf{req}, \mathsf{res})$ *is a request/response structure and* α *a sequence over* A*. Then the* responses $\mathsf{res}(\alpha)$ *of* α *is defined by* $\mathsf{res}(\alpha) = \cup\{\mathsf{res}(a) \mid \exists\varphi, \beta.\ \alpha = \varphi a\beta\}$*. Moreover,* α *is* live *iff* $\alpha = \phi a\beta \implies \mathsf{req}(a) \subseteq \mathsf{res}(\beta)$.

Notation. We shall be specially interested in request/response liveness of sequences of transitions. A *finite transition sequence of length* n is a pair of sequences $(s_i)_{i<n}$ and $(t_i)_{i<n-1}$ s.t. $s_i \xrightarrow{t_i} s_{i+1}$ for $i < n$. An *infinite transition sequence* is a pair of sequences $(s_i)_{i\in\mathbb{N}}$ and $(t_i)_{i\in\mathbb{N}}$ s.t. $s_i \xrightarrow{t_i} s_{i+1}$. A finite or infinite transition sequence of a state s is finite or infinite transition sequence with $s_1 = s$. We write $(s_i, t_i)_{i\in\mathbb{N}}$ for infinite sequences and $((s_i, t_i)_{i<n}, s_n)$ for finite sequences, giving the final state explicitly. Slightly abusing notation, we sometimes write $(s_i, t_i)_{i\in I}$ or even just (s_i, t_i) for a finite or infinite transition sequence, saying that it is a transition sequence *of* s_1 over I.

Definition 3.3 (LTS with requests/responses). *Let* $(S, L, \to)$ *be an LTS. When the set of labels* L *is the set of actions of a request/response structure, we say that* $(S, L, \to)$ *is an* LTS with requests/responses*, and that a transition sequence of this LTS is* live *when its underlying sequence of labels is.*

Next, syntax of types.

$\mathcal{L}$	a countably infinite set of labels
l	ranges over $\mathcal{L}$
L	ranges over $\mathcal{P}(\mathcal{L})$

$$S, T \ ::= \ \&\{l_i[L_i].T_i\}_{i\in I} \ \mid \ \oplus\{l_i[L_i].T_i\}_{i\in I} \ \mid \ !.T \ \mid \ ?.T \ \mid \ \mu t.T \ \mid \ t \ \mid \ \mathtt{end}$$

By convention, the l_i in each $\&\{l_i[L_i].T_i\}_{i\in I}$ resp. $\oplus\{l_i[L_i].T_i\}_{i\in I}$ are distinct.

A session type is a (possibly infinite) tree of actions permitted for one partner of a two-party communication. The type $\&\{l_i[L_i].T_i\}_{i\in I}$, called *branch*, is the type of *offering* a choice between different continuations. If the partner chooses the label l_i, the session proceeds as T_i. Compared to standard session types, making the choice l_i also requests a subsequent response on every label mentioned in the set of labels L_i; we formalise this in the notion of *responsive trace* below. Dual to branch is *select* $\oplus\{l_i[L_i].T_i\}_{i\in I}$: the type of *making* a choice between different continuations. Like branch, making a choice l_i requests every label in L_i as future responses. The type $!.T$ and $?.T$ are the types of sending and receiving data values. As mentioned above, channels cannot be communicated. Also, we have deliberately omitted types of values (e.g. integers, strings, booleans) being sent, since this can be trivially added and we want to focus on the behavioural aspects of the types. Finally, session types with responses include recursive types. We take the equi-recursive view, identifying a type T and its unfolding into a

potentially infinite tree. We define the central notion of *duality* between types as the symmetric relation induced coinductively by the following rules.

$$\frac{}{\mathtt{end} \bowtie \mathtt{end}} \qquad \frac{T \bowtie T'}{!.T \bowtie\ ?.T'} \qquad \frac{T_i \bowtie T_i' \quad J \subseteq I}{\&\{l_i[L_i].T_i\}_{i\in I} \bowtie \oplus\{l_j[L_j'].T_j'\}_{j\in J}} \tag{1}$$

The first rule says that dual processes agree on when communication ends; the second that if a process sends a message, its dual must receive; and the third says that if one process offers a branch, its dual must choose among the offered choices. However, required responses do not need to match: the two participants in a session need not agree on the notion of liveness for the collaborative session.

Example 3.4. Recall from Ex. 2.1 the processes DATA(o) encoding data-object and $P(D)$ encoding the (live) shopping-cart process. The former treats the channel o as $T_D = \mu t.?.\mu s.\&\{\mathsf{read}.!.s,\ \mathsf{write}.t,\ \mathsf{quit}.\mathtt{end}\ \}$. The latter treats its channel k to the buyer as $T_P = \mu t.\&\{\mathsf{AI}.?.t,\ \mathsf{RI}.?.t,\ \mathsf{CO}[\{\mathsf{SI}\}].?.\mu t'.\oplus\{\mathsf{DI}.!.t',\ \mathsf{SI}.!.\mathtt{end}\}\}$. To illustrate both responses in unbounded recursion and duality of disparate responses, note that the $P(D)$ actually treats its data object channel o^- according to the type $T_E = \mu t.!.\mu s.\oplus\{\mathsf{read}.?.s,\ \mathsf{write}[\{\mathsf{read}\}].t,\ \mathsf{quit}.\mathtt{end}\ \}$, i.e., every write is eventually followed by a read. However, $T_D \bowtie T_E$: the types T_E and T_D are nonetheless dual. □

Having defined the syntax of session types with responses, we proceed to give their semantics. The meaning of a session type is the possible sequences of communication actions it allows, requiring that pending responses eventually be done. Formally, we equip session types with a labeled transition semantics in Fig. D. We emphasise that under the equi-recursive view of session types, the

$$\begin{array}{rl} \text{Type transition labels:} & \rho ::= !\ \mid\ ?\ \mid\ \&l[L]\ \mid\ \oplus l[L] \\ \text{Type transition label duality:} & !\bowtie ?\quad\text{and}\quad \&l[L]\bowtie\oplus l[L'] \end{array}$$

$$\text{[D-Out]}\ \frac{}{!.T \xrightarrow{!} T} \qquad \frac{}{?.T \xrightarrow{?} T}\ \text{[D-In]}$$

$$\text{[D-Bra]}\ \frac{i\in I}{\&\{l_i[L_i].T_i\}_{i\in I} \xrightarrow{\&l_i[L_i]} T_i} \qquad \frac{i\in I}{\oplus\{l_i[L_i].T_i\}_{i\in I} \xrightarrow{\oplus l_i[L_i]} T_i}\ \text{[D-Sel]}$$

Fig. D. Transitions of types (1)

transition system of a recursive type T may in general be infinite.

Taking actions A to be the set of labels ranged over by ρ, and recalling that $\mathcal{L}$ is our universe of labels for branch/select, we obtain a request/response structure $(A, \mathcal{P}(\mathcal{L}), \mathsf{req}, \mathsf{res})$ with the latter two operators defined as follows.

$$\begin{array}{ll} \mathsf{res}(!) = \mathsf{res}(?) = \emptyset & \mathsf{res}(\&l[L]) = \mathsf{res}(\oplus l[L]) = \{l\} \\ \mathsf{req}(!) = \mathsf{req}(?) = \emptyset & \mathsf{req}(\&l[L]) = \mathsf{req}(\oplus l[L]) = L \end{array}$$

In the right-hand column, selecting a label l performs the response l; pending responses L associated with that label are conversely requested. The LTS of Fig. D is thus one with responses, and we may speak of its transition sequences being live or not.

Definition 3.5. *Let T be a type. We define:*

1. *The* traces $\mathsf{tr}(T) = \{(\rho_i)_{i\in I} \mid (T_i, \rho_i)_{i\in I}$ *transition sequence of* $T\ \}$
2. *The* responsive traces $\mathsf{tr}_R(T) = \{\alpha \in \mathsf{tr}(T) \mid \alpha$ *live* $\}$.

That is, in responsive traces any request is followed by a response.

Definition 3.6. *A type T is a* standard *session type if it requests no responses, that is, every occurrence of L in it is has $L = \emptyset$. Define an operator $\mathsf{sel}(-)$ as follows, lifting it pointwise to sequences.*

$$\mathsf{sel}(!.T) = \mathsf{sel}(?.T) = \epsilon \qquad \mathsf{sel}(\&l[L]) = \mathsf{sel}(\oplus l[L]) = l$$

We then define:

1. *The* selection traces $\mathsf{str}(T) = \{\mathsf{sel}(\alpha) \mid \alpha \in \mathsf{tr}(T)\}$
2. *The* responsive selection traces $\mathsf{str}_R(T) = \{\mathsf{sel}(\alpha) \mid \alpha \in \mathsf{tr}_R(T)\}$.
3. *The* language of standard session types
   $\mathcal{T} = \{\alpha \mid \alpha \in \mathsf{str}(T), T$ *is a standard session type*$\}$.
4. *The* language of responsive session types
   $\mathcal{R} = \{\alpha \mid \alpha \in \mathsf{str}_R(T), T$ *is a session type with responses*$\}$.

That is, we compare standard session types and session types of responses by considering the sequences of branch/select labels they admit. This follows recent work on multi-party session types and automata [8,9].

A fine point: because the $\mathsf{sel}(-)$ map is lifted pointwise and maps "no selection" to the empty string ϵ, this definition of languages is oblivious to send and receive. E.g, if ϕ_S, ψ_T are the unique traces of the two types $S = \ !.\oplus l.\oplus l'.\mathtt{end}$ and $T = \oplus l.?.\oplus l'.\mathtt{end}$, then $\mathsf{sel}(\phi_S) = \mathsf{sel}(\psi_T) = ll'$. We formalise this insight in the following lemma.

Lemma 3.7. *Let T be a standard session type. There exists a session type T' with no occurrences of send $!.T$ or receive $?.T$ s.t. $\mathsf{str}(T) = \mathsf{str}(T')$.*

Example 3.8. The type T_P of Example 3.4 has (amongst others) the two selection traces: $t = \mathsf{AI}\,\mathsf{CO}\,\mathsf{DI}\,\mathsf{DI}\,\mathsf{SI}$ and $u = \mathsf{AI}\,\mathsf{CO}\,\mathsf{DI}\,\mathsf{DI}\,\mathsf{DI}\cdots$. Of these, only t is responsive; u is not, since it never selects SI as required by its CO action. That is, $t, u \in \mathsf{str}(T_P)$ and $t \in \mathsf{str}_R(T_P)$, but $u \notin \mathsf{str}_R(T_P)$. □

Lemma 3.9 (Session types with responses are deterministic). *(1). If $T \xrightarrow{\rho}$ and $T \xrightarrow{\rho'}$ and $\mathsf{sel}(\rho) = \mathsf{sel}(\rho') \neq \epsilon$, then $\rho = \rho'$. (2). Consider equally long finite transition sequences $((T_i, \rho_i)_{i<n}, T_n)$ and $((S_i, \rho'_i)_{i<n}, S_n)$. If $T_1 = S_1$ and for each $i < n$ $\rho_i = \rho'_i$, then also $T_i = S_i$ for each $i \leq n$.*

Proof. (1). Immediate from the convention that each label in a branch or selected is distinct. (2). By induction on n. The base case is trivial. For $n = k+1$ we have by the induction hypothesis $T_k = S_k$. By convention, each label in a branch or select is distinct, so there is at most one S with $T_k \xrightarrow{\rho_k} S$. But then $S = T_{k+1} = S_{k+1}$. □

Theorem 3.10. *The language of session types with responses $\mathcal{R}$ is strictly more expressive than that of standard session types $\mathcal{T}$; that is, $\mathcal{T} \subset \mathcal{R}$.*

Proof. The non-strict inclusion is immediate by definition; it remains to prove it strict. Consider the following session type with responses, T.

$$T \;=\; \mu t.\, \oplus \begin{cases} a[b] : t \\ b[a] : t \end{cases}$$

We shall prove that $\mathsf{str}_R(T) \notin \mathcal{T}$. Suppose not; then there exists a session type S with $\mathsf{str}(S) = \mathsf{str}_R(T)$. Clearly the responsive selection traces $\mathsf{str}_R(T)$ is the set of infinite strings over the alphabet $\{a, b\}$ where both a, b occur infinitely often. It follows that for all $k > 0$, the string a^k is a prefix of an infinite string in $\mathsf{str}_R(T)$. We have assumed $\mathsf{str}(S) = \mathsf{str}_R(T)$, so each a^k must also be a prefix of an infinite string in $\mathsf{tr}(S)$. By Lemma 3.7, we may assume S has no occurrences of send or receive, and so for each k there is a transition sequence $((S_i^k, \rho_i^k)_{i<k}, S_k^k)$ with $S_1^k = S$ and $\mathsf{sel}(\rho_i^k) = a$. By induction on k using Lemma 3.9, we find that $\rho_i^k = \rho_i^{k+1}$ and $S_i^k = S_i^{k+1}$ when $i \leq k$. It follows that $S_i^i = S_i^{i+1} \xrightarrow{\rho_i} S_{i+1}^{i+1}$ when $i + 1 \leq k$, and so $(S_i^i, \rho_i)_{i\in\mathbb{N}}$ is an infinite transition sequence with $S_1^1 = S$. But then $(\mathsf{sel}(\rho_i))_{i\in\mathbb{N}} = a^\omega \in \mathsf{str}(S)$ while clearly not in $\mathsf{str}_R(T)$, contradicting $\mathsf{str}_R(T) = \mathsf{str}(S)$. □

4 Session Typing

Recall the standard type system for session types, presented in Fig. E with the obvious extension for primitive recursion. In this judgement, Θ takes process variables to session type environments; in turn, a *session typing environment* Δ is a finite partial map from channel names and polarised channel names to types. We write Δ, Δ' for the union of Δ and Δ', defined when their domains are disjoint. We say Δ is *completed* if $\Delta(T) = \mathtt{end}$ when defined; it is *balanced* if $k : T, \overline{k} : U \in \Delta$ implies $T \bowtie U$.

We generalise transitions of types (Fig. D) to session typing environments in Fig. F, with transitions ranged over by δ as follows; recall that ρ is a type transition label.

$$\delta ::= \tau \mid \tau : l, L \mid k : \rho$$

We define $\mathsf{subj}(k : \rho) = k$ and $\mathsf{subj}(\tau : l, L) = \mathsf{subj}(\tau) = \tau$. We lift $\mathsf{sel}(-), \mathsf{req}(-)$, and $\mathsf{res}(-)$ to actions δ as follows.

$$\begin{array}{lll} \mathsf{sel}(\tau) = \epsilon & \mathsf{sel}(k : \rho) = \mathsf{sel}(\rho) & \mathsf{sel}(\tau : l, L) = l \\ \mathsf{req}(\tau) = \emptyset & \mathsf{req}(\tau : \rho) = \mathsf{req}(\rho) & \mathsf{req}(\tau : l, L) = L \\ \mathsf{res}(\tau) = \emptyset & \mathsf{res}(k : \rho) = \mathsf{res}(\rho) & \mathsf{res}(k : l, L) = \{l\} \end{array}$$

$$\text{[E-Out]}\quad \frac{\Theta \vdash_{\mathsf{std}} P \triangleright \Delta, k : T}{\Theta \vdash_{\mathsf{std}} k!\langle e\rangle.P \triangleright \Delta, k : !.T} \qquad \frac{\Theta \vdash_{\mathsf{std}} P \triangleright \Delta, k : T}{\Theta \vdash_{\mathsf{std}} k?(x).P \triangleright \Delta, k : ?.T}\quad \text{[E-In]}$$

$$\text{[E-Bra]}\quad \frac{\forall i \in I : \quad \Theta \vdash_{\mathsf{std}} P_i \triangleright \Delta, k : T_i}{\Theta \vdash_{\mathsf{std}}\ k?\{l_i.P_i\}_{i\in I} \triangleright \Delta, k : \&\{l_i[L_i].T_i\}_{i\in I}}$$

$$\text{[E-sel]}\quad \frac{\Theta \vdash_{\mathsf{std}} P \triangleright \Delta, k : T_j}{\Theta \vdash_{\mathsf{std}} k!l_j.P \triangleright \Delta, k : \oplus\{l_i[L_i].T_i\}_{i\in I}}\quad (j \in I)$$

$$\text{[E-Par]}\quad \frac{\Theta \vdash_{\mathsf{std}} P_1 \triangleright \Delta_1 \qquad \Theta \vdash_{\mathsf{std}} P_2 \triangleright \Delta_2}{\Theta \vdash_{\mathsf{std}} P_1 \mid P_2 \triangleright \Delta_1, \Delta_2} \qquad \frac{\Delta \text{ completed}}{\Theta \vdash_{\mathsf{std}} \mathbf{0} \triangleright \Delta}\quad \text{[E-Inact]}$$

$$\frac{\Theta, X : \Delta \vdash_{\mathsf{std}} P \triangleright \Delta \qquad \Theta \vdash_{\mathsf{std}} Q \triangleright \Delta}{\Theta \vdash_{\mathsf{std}} (\mathrm{rec}^e\, X(i).P; Q) \triangleright \Delta}\quad \text{[E-RecP]}$$

$$\text{[E-Rec]}\quad \frac{\Theta, X : \Delta \vdash_{\mathsf{std}} P \triangleright \Delta}{\Theta \vdash_{\mathsf{std}} \mathrm{rec}\, X.P \triangleright \Delta} \qquad \frac{\mathsf{dom}(\Delta) = \tilde{k}}{\Theta, X : \Delta \vdash_{\mathsf{std}} X[\tilde{k}] \triangleright \Delta}\quad \text{[E-Var]}$$

$$\text{[E-Cond]}\quad \frac{\Theta \vdash_{\mathsf{std}} P \triangleright \Delta \qquad \Theta \vdash_{\mathsf{std}} Q \triangleright \Delta}{\Theta \vdash_{\mathsf{std}} \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta}$$

Fig. E. Standard Session Typing System

The type environment transition is thus an LTS with responses, and we may speak of its transition sequences being live.

Definition 4.1. *We define a binary relation on type transition labels δ and transition labels λ, written $\delta \simeq \lambda$, as follows.*

$$\begin{array}{ccccc} \tau \simeq \tau & & k : \&l[L] \simeq k\&l & & k : ! \simeq k!v \\ \tau : l, L \simeq \tau : l & & k : \oplus l[L] \simeq k \oplus l & & k : ? \simeq k?x \end{array}$$

Theorem 4.2. *If $\Gamma \vdash_{\mathsf{std}} P \triangleright \Delta$ and $P \xrightarrow{\lambda} Q$, then there exists $\delta \simeq \lambda$ s.t. $\Delta \xrightarrow{\delta} \Delta'$ and $\Gamma \vdash_{\mathsf{std}} Q \triangleright \Delta'$.*

The proof is in Appendix A.

Definition 4.3. *The* typed transition system *is the transition system which has states $\Gamma \vdash_{\mathsf{std}} P \triangleright \Delta$ and transitions $\Gamma \vdash_{\mathsf{std}} P \triangleright \Delta \xrightarrow{\lambda,\delta} \Gamma \vdash_{\mathsf{std}} P' \triangleright \Delta'$ whenever there exist transitions $P \xrightarrow{\lambda} P'$ and $\Delta \xrightarrow{\delta} \Delta'$ with $\delta \simeq \lambda$.*

We can now say what it means for a process to be live (relying on the definition of maximal transition sequences given in Def. 6.5 below).

$$[\text{F-Lift}] \quad \frac{T \xrightarrow{\rho} T'}{k : T \xrightarrow{k:\rho} k : T'}$$

$$[\text{F-Par}] \quad \frac{\Delta \xrightarrow{\delta} \Delta'}{\Delta, \Delta'' \xrightarrow{\delta} \Delta', \Delta''}$$

$$[\text{F-Com1}] \quad \frac{\Delta_1 \xrightarrow{k:!} \Delta_1' \qquad \Delta_2 \xrightarrow{\overline{k}:?} \Delta_2'}{\Delta_1, \Delta_2 \xrightarrow{\tau} \Delta_1', \Delta_2'}$$

$$[\text{F-Com2}] \quad \frac{\Delta_1 \xrightarrow{k:\oplus l[L]} \Delta_1' \qquad \Delta_2 \xrightarrow{\overline{k}:\& l[L']} \Delta_2'}{\Delta_1, \Delta_2 \xrightarrow{\tau:l, L \cup L'} \Delta_1', \Delta_2'}$$

Fig. F. Transitions of types (2)

Definition 4.4 (Live process). *A well-typed process $\Theta \vdash_{\mathsf{std}} P \triangleright \Delta$ is* live *wrt. Θ, Δ iff for any maximal transition sequence $(P_i, \lambda_i)_i$ of P there exists a live type transition sequence $(\Delta_i, \delta_i)_i$ of Δ s.t. $((P_i, \Delta_i), (\lambda_i, \delta_i))_i$ is a typed transition sequence of $\Theta \vdash_{\mathsf{std}} P \triangleright \Delta$.*

Example 4.5. Wrt. the standard session typing system, *both* of the processes $P(D_0)$ and $P(D)$ of Example 2.1 are typable wrt. the types we postulated for them in Example 3.4. Specifically, we have $\cdot \vdash_{\mathsf{std}} P(D_0) \triangleright k : T_P, o^+ : T_D, o^- : \overline{T_D}$ and similarly for $P(D)$. The judgement means that the process $P(D)$ treats k according to T_P and the (two ends of) the data object according to T_D and its syntactic dual $\overline{T_D}$. The standard session typing system of course does not act on our liveness annotations, and so does not care that $P(D_0)$ is not live.

For the subsequent development, we will need the following lemmas.

Lemma 4.6. *If $\Delta \xrightarrow{\delta} \Delta'$ then $\mathsf{dom}(\Delta) = \mathsf{dom}(\Delta')$.*

Proof. Straightforward induction on the derivation of the transition. □

Lemma 4.7. *If $\Delta \xrightarrow{\delta} \Delta'$ then either:*

1. *$\delta = k : \rho$ and $\Delta = \Delta'', k : T$ and $\Delta' = \Delta'', k : T'$ and $T \xrightarrow{\rho} T'$; or*
2. *$\delta = \tau$ or $\delta = \tau : l, L$ and $\Delta = \Delta'', k : T, \overline{k} : S$ and $\Delta' = \Delta'', k : T', \overline{k} : S'$ where $T \xrightarrow{\rho} T'$ and $S \xrightarrow{\rho'} S'$ and $\rho \bowtie \rho'$.*

Proof. Straightforward induction on the derivation of the transition. □

Lemma 4.8. *If $\Delta \xrightarrow{\delta} \Delta'$ with Δ balanced and $\overline{\mathsf{subj}(\delta)} \notin \mathsf{dom}(\Delta)$, then also Δ' balanced.*

Proof. By induction on the derivation of the transition.

Case [F-Lift]**.** Trivial.

Case [F-Par]**.** Suppose Δ, Δ'' balanced with $\overline{\mathsf{subj}(\delta)} \notin \mathsf{dom}(\Delta, \Delta'')$, and consider $k, \overline{k} \in \mathsf{dom}(\Delta', \Delta'')$. If both are in $\mathsf{dom}(\Delta'')$, we are done. If both are in $\mathsf{dom}(\Delta')$ then by the Lemma 4.6 they are also in $\mathsf{dom}(\Delta)$. Then, because Δ, Δ'' balanced implies Δ balanced, we find by the induction hypothesis that also Δ' balanced, whence $(\Delta', \Delta'')(k) = \Delta'(k) \bowtie \Delta'(\overline{k}) = (\Delta', \Delta'')(\overline{k})$. Finally, suppose wlog $k \in \mathsf{dom}(\Delta')$ and $\overline{k} \in \mathsf{dom}(\Delta'')$; we shall see that this is not possible. By Lemma 4.7 either

$$\Delta = \Delta_1, k : T \xrightarrow{k:\rho} \Delta_1, k : T' = \Delta' \qquad \text{with } T \xrightarrow{\rho} T', \tag{2}$$

or

$$\Delta = \Delta_2, k : T, \overline{k} : S \xrightarrow{\delta} \Delta_2, k : T', \overline{k} : S' = \Delta' \qquad \text{with } T \xrightarrow{\rho} T' \text{ and } S \xrightarrow{\overline{\rho}} S'. \tag{3}$$

We consider these two possibilities in turn. It cannot be true that (2) holds, because by the assumption $\overline{\mathsf{subj}(\delta)} \notin \mathsf{dom}(\Delta, \Delta'')$ we must have $\overline{k} = \overline{\mathsf{subj}(\delta)} \notin \mathsf{dom}(\Delta'')$, contradicting $\overline{k} \in \mathsf{dom}(\Delta'')$. If instead (3) holds, then because $\overline{k} \in \mathsf{dom}(\Delta'')$ then Δ, Δ'' is not defined, contradicting the existence of the transition $\Delta, \Delta'' \xrightarrow{\delta} \Delta', \Delta''$.

Case [F-Com1]**.** Suppose $\Delta_1, \Delta_2 \xrightarrow{\tau} \Delta_1', \Delta_2'$ with Δ_1, Δ_2 balanced. Using Lemma 4.7, and [F-Com1] we have transitions $\Delta_1 = \Delta_3, k : S \xrightarrow{k:!} \Delta_3, k : S' = \Delta_1'$ and $\Delta_2 = \Delta_4, \overline{k} : T \xrightarrow{\overline{k}:?} \Delta_4, \overline{k} : T' = \Delta_2'$, with $S \xrightarrow{!} S'$ and $T \xrightarrow{?} T'$. It follows that our original transition is on the form

$$\Delta_1, \Delta_2 = \Delta_3, k : S, \Delta_4, \overline{k} : T \xrightarrow{\tau} \Delta_3, k : S', \Delta_4, \overline{k} : T' = \Delta_1', \Delta_2'.$$

Because Δ_1, Δ_2 balanced then also Δ_3, Δ_4 is, and so $\Delta_1', \Delta_2' = \Delta_3, k : S', \Delta_4, \overline{k} : T'$ is balanced if $S' \bowtie T'$. But $S \xrightarrow{!} S'$ implies $S = !.S'$ and $T \xrightarrow{?} T'$ implies $T = ?.T'$. But we have $!.S' = S \bowtie T = ?.T'$ by Δ_1, Δ_2 balanced, and so by definition $S' \bowtie T'$.

Case [F-Com2]**.** We have $\Delta_1, \Delta_2 \xrightarrow{\tau, l:L \cup L'} \Delta_1', \Delta_2'$ with Δ_1, Δ_2 balanced. By Lemma 4.7, and [F-Com2] we have transitions $\Delta_1 = \Delta_3, k : S \xrightarrow{k:\oplus l[L]} \Delta_3, k : S' = \Delta_1'$ and $\Delta_2 = \Delta_4, \overline{k} : T \xrightarrow{\overline{k}:\& l[L']} \Delta_4, \overline{k} : T' = \Delta_2'$ and $S \xrightarrow{\oplus l[L]} S'$ and $T \xrightarrow{\& l[L']} T'$. Then our original transition is on the form

$$\Delta_1, \Delta_2 = \Delta_3, k : S, \Delta_4, \overline{k} : T \xrightarrow{\tau, l:L \cup L'} \Delta_3, k : S', \Delta_4, \overline{k} : T' = \Delta_1', \Delta_2'.$$

Because Δ_1, Δ_2 balanced then also Δ_3, Δ_4 is, and so $\Delta_1', \Delta_2' = \Delta_3, k : S', \Delta_4, \overline{k} : T'$ is balanced if $S' \bowtie T'$. But $S \xrightarrow{k:\oplus l[L]} S'$ implies $S = \oplus\{l_j[L_j'].S_j'\}_{j \in J}$ with $l = l_i$ and $L = L_i$ for some $i \in J$, and $S' = S_i'$. Similarly $T \xrightarrow{\& l[L]} T'$ implies $T = \{l_i[L_i'].T_i'\}_{i \in I}$ with $j \in I$, $l = l_j$, $L' = L_j$, and $T' = T_j$. Because $S \bowtie T$ we may assume $J \subseteq I$ and $i = j$, whence by definition $T_i' \bowtie S_j'$. ◻

5 Typing System for Liveness

In this section, we introduce a variant of the standard session-typing system, give intuition for it, and establish its basic properties, notably subject reduction. In the next section, we shall prove that this typing system does indeed guarantee liveness of well-typed processes.

The central judgement will be $\Gamma; L \vdash P \triangleright \Delta$, with the intended meaning that "with process variables Γ and pending responses L, the process P conforms to Δ." We shall see in the next section that a well-typed lock-free P is live and will eventually perform every response in in L.

In detail, here are the environments used in the typing system, along with auxiliary operations on them.

1. *Session typing environments* Δ defined at the start of Section 4.
2. *Response environments* L are simply sets of branch/select labels.
3. *Process variable environments* Γ are finite partial maps from process variables X to tuples (L, L, Δ) or (L, Δ). We write these (A, I, Δ) for (A)ccumulated selections and request (I)nvariant.
   We write $\Gamma + L$ for the environment satisfying that

$$(\Gamma + L)(X) = \begin{cases} (A \cup L, I, \Delta) & \text{whenever } \Gamma(X) = (A, I, \Delta) \\ \Gamma(X) & \text{otherwise} \end{cases}$$

   We sometimes write $\Gamma + l$ instead of $\Gamma + \{l\}$.

Our typing system is in Fig. G. The rules [G-Bra]/[G-Sel] types branch/select. To type $k!l.P$ wrt. $k : \oplus l[L'].T$, P must do every response in L'. For this we maintain an environment L of pending responses. In the hypothesis, when typing P, we add to this the new pending responses L'. But selecting l performs the response l, so altogether, to support pending responses L in the conclusion, we must have pending responses $L \setminus \{l\} \cup L'$ in the hypothesis. Branching is similar.

For finite processes, if only the inactive process can be typed with the empty request environment, liveness is ensured. Hence in the rule **0**, that environment is required to be empty. For infinite processes there is no point at which we can insist on having no pending responses. Indeed, much like the contemporary post doc, a process can be live, meeting its requirements, even though it is always have some pending responses. Take for instance this process, typeable with the type used in the proof of Theorem 3.10.

$$\operatorname{rec} X.k \oplus a.\ k \oplus b.\ X[k] \qquad \triangleright \qquad k : \mu t. \oplus \begin{cases} a[b] : t \\ b[a] : t \end{cases} .$$

This process has the single transition sequence

$$P \xrightarrow{k \oplus a} k \oplus b.\ P \xrightarrow{k \oplus b} P \xrightarrow{k \oplus a} \cdots$$

At each state but the initial one there is a pending response: either b is pending or a is. Yet the process is live: *any response requested in the body of the recursion*

$$[\text{G-Out}] \quad \frac{\Gamma; L \vdash P \triangleright \Delta, k : T}{\Gamma; L \vdash k!\langle e\rangle.P \triangleright \Delta, k : !.T}$$

$$[\text{G-In}] \quad \frac{\Gamma; L \vdash P \triangleright \Delta, k : T}{\Gamma; L \vdash k?(x).P \triangleright \Delta, k : ?.T}$$

$$[\text{G-Bra}] \quad \frac{\forall i \in I : \quad \Gamma + l_i; (L \setminus l_i) \cup L_i \vdash P_i \triangleright \Delta, k : T_i}{\Gamma; L \vdash \ k?\{l_i.P_i\}_{i\in I} \triangleright \Delta, k : \&\{l_i[L_i].T_i\}_{i\in I}}$$

$$[\text{G-Sel}] \quad \frac{\Gamma + l_j; (L \setminus l_j) \cup L_j \vdash P \triangleright \Delta, k : T_j}{\Gamma; L \vdash k!l_j.P \triangleright \Delta, k : \oplus\{l_i[L_i].T_i\}_{i\in I}} \quad (j \in I)$$

$$[\text{G-Par}] \quad \frac{\Gamma; L_1 \vdash P_1 \triangleright \Delta_1 \qquad \Gamma; L_2 \vdash P_2 \triangleright \Delta_2}{\Gamma; L_1 \cup L_2 \vdash P_1 \mid P_2 \triangleright \Delta_1, \Delta_2} \qquad [\text{G-Inact}] \quad \frac{\Delta \text{ completed}}{\Gamma; \emptyset \vdash \mathbf{0} \triangleright \Delta}$$

$$[\text{G-Var}] \quad \frac{L \subseteq I \subseteq A \qquad \mathsf{dom}(\Delta) = \tilde{k}}{\Gamma, X : (A, I, \Delta); L \vdash X[\tilde{k}] \triangleright \Delta}$$

$$[\text{G-VarP}] \quad \frac{L \subseteq L' \qquad \mathsf{dom}(\Delta) = \tilde{k}}{\Gamma, X : (L', \Delta); L \vdash X[\tilde{k}] \triangleright \Delta}$$

$$[\text{G-RecP}] \quad \frac{\Gamma, X : (L', \Delta); L' \vdash P \triangleright \Delta \qquad \Gamma; L' \vdash Q \triangleright \Delta \qquad L \subseteq L'}{\Gamma; L \vdash (\text{rec}^e\, X(i).P; Q) \triangleright \Delta}$$

$$[\text{G-Rec}] \quad \frac{\Gamma, X : (\emptyset, I, \Delta); I \vdash P \triangleright \Delta \qquad L \subseteq I}{\Gamma; L \vdash \text{rec}\, X.P \triangleright \Delta}$$

$$[\text{G-Cond}] \quad \frac{\Gamma; L \vdash P \triangleright \Delta \qquad \Gamma; L \vdash Q \triangleright \Delta}{\Gamma; L \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta}$$

Fig. G. Typing System

is also discharged in the body, although not necessarily in the proper order. In general, infinite behaviour arises because of unfolding of recursion, so if the body of every recursion discharges the requests of that body, even if not in the proper order, responses are ensured.

For general recursion, [G-Rec] and [G-Var], we need to check that there exists an invariant, a set of responses, such that the body of a recursion requests at most that set, and reponds with at least that set. In the process variable environment Γ we record this response invariant for each variable, along with a tally of the responses performed since the start of the recursion. That tally is then updated by the rules [G-Sel]/[G-Bra] for select and branch. The rule for

process variable [G-Var] typing then check that the tally includes the invariant, and that the invariant includes every currently pending response.

This concludes our walk-through of the rules.

Definition 5.1. *We define the* standard *process variable environment* $\mathsf{std}(\Gamma)$ *associated with a process variable environment* Γ *as follows.*

$$\mathsf{std}(\Gamma)(X) = \begin{cases} \Delta & \text{whenever } \Gamma(X) = (A, I, \Delta) \\ \Delta & \text{whenever } \Gamma(X) = (I, \Delta) \end{cases}$$

Theorem 5.2. *If* $\Gamma; L \vdash P \triangleright \Delta$ *then also* $\mathsf{std}(\Gamma) \vdash_{\mathsf{std}} P \triangleright \Delta$.

Proof. Straightforward induction on the typing derivation, using for [G-Bra] and [G-Sel] that $\mathsf{std}(\Gamma + L) = \mathsf{std}(\Gamma)$; for [G-Var]/[G-Rec] that $\mathsf{std}(\Gamma, X : (A, I, \Delta)) = \mathsf{std}(\Gamma), X : \Delta$; and for [G-VarP]/[G-RecP] that $\mathsf{std}(\Gamma, X : (I, \Delta)) = \mathsf{std}(\Gamma), X : \Delta$. □

We proceed to establish basic properties of our typing system, eventually arriving at subject reduction.

Lemma 5.3. *If* $\Gamma; L \vdash P \triangleright \Delta$ *and* $L' \subseteq L$*, then also* $\Gamma; L' \vdash P \triangleright \Delta$.

Proof. By induction on the derivation of the typing of P.

Case [G-Inact]**.** We have $\Gamma; L \vdash \mathbf{0} \triangleright \Delta$. By typing $L = \emptyset$ and our desired property is vacously true.

Case [G-Out]**.** Immediate from the induction hypothesis.

Case [G-In]**.** Immediate from the induction hypothesis.

Case [G-Bra]**.** We have $\Gamma; L \vdash \ x?\{l_i.P_i\} \triangleright \Delta, k : \&\{l_i[L_i].T_i\}_{i \in I}$. By typing we must have for all $i \in I$ that $\Gamma + l_i; (L \setminus l_i) \cup L_i \vdash P_i \triangleright \Delta, k : T_i$. By the induction hypothesis $\Gamma + l_i; (L' \setminus l_i) \cup L_i \vdash P_i \triangleright \Delta, k : T_i$, and we conclude $\Gamma; L' \vdash \ k?\{l_i.P_i\} \triangleright \Delta, k : \&\{l_i[L_i].T_i\}_{i \in I}$.

Case [G-Sel]**.** Similar to [G-Bra].

Case [G-Par]**.** We have $\Gamma; L \vdash P_1 \mid P_2 \triangleright \Delta$. By typing we have $\Gamma; L_i \vdash P_i \triangleright \Delta_i$ with $L = L_1 \cup L_2$ and $\Delta = \Delta_1, \Delta_2$. Consider a subset $L' \subseteq L_1 \cup L_2$. By the induction hypothesis $\Gamma; L_i \cap L' \vdash P_i \triangleright \Delta_i$ and, noting that $(L_1 \cap L') \cup (L_2 \cap L') = (L_1 \cup L_2) \cap L' = L'$, we find $\Gamma; L' \vdash P_1 \mid P_2 \triangleright \Delta$

Case [G-VarP]**.** Immediate from the premise $L \subseteq L'$.

Case [G-RecP]**.** Immediate from the premise $L \subseteq L'$.

Case [G-Var]**.** Immediate from the premise $L \subseteq I$.

Case [G-Rec]**.** Immediate from the premise $L \subseteq I$.

Case [G-Cond]**.** Immediate from the induction hypothesis.

□

Lemma 5.4 (Process variable substitution). *Suppose that* $\Gamma, X : t; L \vdash P \triangleright \Delta$ *where either* $t = (A, I, \Delta')$ *or* $t = (I, \Delta')$*. Suppose moreover that* $\Gamma; I \vdash Q \triangleright \Delta'$ *with* X *is not free in* Q*. Then also* $\Gamma; L \vdash P\{Q/X\} \triangleright \Delta$

Proof. By induction on the typing derivation.

Case [G-Inact]**.** We have $\Gamma, X : t; L \vdash \mathbf{0} \triangleright \Delta$. By typing $L = \emptyset$. Observe that $\mathbf{0}\{Q/X\} = \mathbf{0}$. Thus, by [G-Inact], we have $\Gamma; L \vdash \mathbf{0}\{Q/X\} \triangleright \Delta$.

Case [G-Out]**.** Immediate from the induction hypothesis.

Case [G-In]**.** Immediate from the induction hypothesis.

Case [G-Bra]**.** By typing, we have

$$\frac{\forall i \in I : \quad (\Gamma, X : t) + l_i; (L \setminus l_i) \cup L_i \vdash P_i \triangleright \Delta, k : T_i}{\Gamma, X : t; L \vdash \ k?\{l_i.P_i\}_{i \in I} \triangleright \Delta, k : \&\{l_i[L_i].T_i\}_{i \in I}}$$

Suppose first $t = (A, I, \Delta')$. Then $(\Gamma, X : (A, I, \Delta')) + l_i = (\Gamma + l_i), X : (A \cup l_i, I, \Delta')$. But then we may apply the induction hypothesis and [G-Bra] to obtain

$$\frac{\Gamma + l_i; (L \setminus l_i) \cup L_i \vdash P_i\{Q/X\} \triangleright \Delta, k : T_i}{\Gamma; L \vdash \ k?\{l_i.P_i\{Q/X\}\}_{i \in I} \triangleright \Delta, k : \&\{l_i[L_i].T_i\}_{i \in I}} \quad . \tag{4}$$

Suppose instead $t = (I, \Delta')$. Then $(\Gamma, X : (I, \Delta')) + l_i = (\Gamma + l_i), X : (I, \Delta')$, and again we may apply the induction hypothesis and [G-Bra] to obtain (4).

Case [G-Sel]**.** Similar to [G-Bra].

Case [G-Par]**.** We have $\Gamma, X : t; L \vdash P_1 \mid P_2 \triangleright \Delta$. By typing we find some $L_1 \cup L_2 = L$ and $\Delta_1, \Delta_2 = \Delta$ such that $\Gamma, X : t; L_i \vdash P_i \triangleright \Delta_i$. By the induction hypothesis we find $\Gamma; L_i \vdash P_i\{Q/X\} \triangleright \Delta_i$, which in turn yields $\Gamma; L_1 \cup L_2 \vdash (P_1 \mid P_2)\{Q/X\} \triangleright \Delta_1, \Delta_2$.

Case [G-VarP]**.** Suppose first $X \neq Y$; then by typing we have

$$\frac{L \subseteq L' \qquad \mathsf{dom}(\Delta) = \tilde{k}}{\Gamma, Y : (L', \Delta), X : t; L \vdash Y[\tilde{k}] \triangleright \Delta} \ ,$$

so by [G-VarP] also

$$\Gamma, Y : (L', \Delta); L \vdash Y[\tilde{k}]\{Q/X\} \triangleright \Delta \ .$$

If on the other hand $X = Y$ we have by typing

$$\frac{L \subseteq L' \qquad \mathsf{dom}(\Delta) = \tilde{k}}{\Gamma, X : (L', \Delta); L \vdash X[\tilde{k}] \triangleright \Delta} \ ;$$

and it must be the case that $I = L'$ and $\Delta = \Delta'$. We have by assumption $\Gamma; I \vdash Q \triangleright \Delta'$, that is $\Gamma; L' \vdash Q \triangleright \Delta$. By Lemma 5.8 also $\Gamma; L \vdash Q \triangleright \Delta$, that is, $\Gamma; L \vdash X[\tilde{k}]\{Q/X\} \triangleright \Delta$.

Case [G-RecP]**.** We have $\Gamma, X : (A, I, \Delta'); L \vdash (\mathrm{rec}^e\, Y(i).P; R) \triangleright \Delta$. By typing we have $\Gamma, X : (A, I, \Delta'), Y : (L', \Delta); L' \vdash P \triangleright \Delta$ and $\Gamma; L' \vdash R \triangleright \Delta$ for some $L' \supseteq L$. Using $\Gamma; I \vdash Q \triangleright \Delta'$, by the induction hypothesis $\Gamma, Y : (L', \Delta); L' \vdash P\{Q/X\} \triangleright \Delta$ and $\Gamma; L' \vdash R\{Q/X\} \triangleright \Delta$, which in turn yields $\Gamma; L \vdash (\mathrm{rec}^e\, Y(i).P; R)\{Q/X\} \triangleright \Delta$.

Case [G-VAR]**.** Suppose first $X \neq Y$; then by typing we have

$$\frac{L \subseteq L' \qquad \mathsf{dom}(\Delta) = \tilde{k}}{\Gamma, Y : (A', I', \Delta), X : t; L \vdash Y[\tilde{k}] \triangleright \Delta} \ ,$$

so by [G-VAR] also

$$\Gamma, Y : (A', I', \Delta); L \vdash Y[\tilde{k}]\{Q/X\} \triangleright \Delta \ .$$

If on the other hand $X = Y$ we have by typing

$$\frac{L \subseteq I \subseteq A \qquad \mathsf{dom}(\Delta) = \tilde{k}}{\Gamma, X : (I, A, \Delta'); L \vdash X[\tilde{k}] \triangleright \Delta'} \ ;$$

where necessarily $\Delta' = \mathit{Delta}$. We have by assumption $\Gamma; I \vdash Q \triangleright \Delta'$. By Lemma 5.8 also $\Gamma; L \vdash Q \triangleright \Delta'$, that is, $\Gamma; L \vdash X[\tilde{k}]\{Q/X\} \triangleright \Delta'$.

Case [G-REC]**.** We have $\Gamma, X : (A, I, \Delta'); L \vdash \mathrm{rec}\, Y.P \triangleright \Delta$. We find by typing $\Gamma, X : (A, I, \Delta'), Y : (A', I', \Delta); L \vdash P \triangleright \Delta$ with $L \subseteq I'$, hence by the induction hypothesis $\Gamma, Y : (A', I', \Delta); L \vdash P\{Q/X\} \triangleright \Delta$, and so by [G-REC] $\Gamma; L \vdash (\mathrm{rec}\, Y.P)\{Q/X\} \triangleright \Delta$.

Case [G-COND]**.** Immediate from the induction hypothesis. □

Lemma 5.5. *If $\Gamma; L \vdash P \triangleright \Delta$ then also $\Gamma; L \vdash P\{v/x\} \triangleright \Delta$.*

Proof. Straightforward induction. □

Definition 5.6. *We define $\Gamma \leq \Gamma'$ iff $\Gamma(X) = (A, I, \Delta)$ implies $\Gamma'(X) = (A', I, \Delta)$ with $A \subseteq A'$ and $\Gamma(X) = (I, \Delta)$ implies $\Gamma'(X) = (I, \Delta)$.*

Lemma 5.7. *If $\Gamma; L \vdash P \triangleright \Delta$ and $\Gamma \leq \Gamma'$ then also $\Gamma'; L \vdash P \triangleright \Delta$.*

Proof. Straightforward induction. We report the two essential cases.

Case [G-SEL]**.** We have

$$\frac{\Gamma + l_j; (L \setminus l_j) \cup L_j \vdash P \triangleright \Delta, k : T_j}{\Gamma; L \vdash k!l_j.P \triangleright \Delta, k : \oplus\{l_i[L_i].T_i\}_{i \in I}}$$

Noting that $\Gamma \leq \Gamma'$ implies $\Gamma + l_j \leq \Gamma' + l_j$ we find by IH and [G-SEL]

$$\frac{\Gamma' + l_j; (L \setminus l_j) \cup L_j \vdash P \triangleright \Delta, k : T_j}{\Gamma'; L \vdash k!l_j.P \triangleright \Delta, k : \oplus\{l_i[L_i].T_i\}_{i \in I}} \ .$$

Case [G-REC]**.** We have

$$\frac{\Gamma, X : (\emptyset, I, \Delta); I \vdash P \triangleright \Delta \qquad L \subseteq I}{\Gamma; L \vdash \mathrm{rec}\, X.P \triangleright \Delta}$$

Noting that $\Gamma, X : (\emptyset, I, \Delta) \leq \Gamma', X : (\emptyset, I, \Delta)$ we have by IH and [G-REC]

$$\frac{\Gamma', X : (\emptyset, I, \Delta); I \vdash P \triangleright \Delta \qquad L \subseteq I}{\Gamma'; L \vdash \mathrm{rec}\, X.P \triangleright \Delta} \ .$$

Lemma 5.8. *If $\Gamma; L \vdash P \triangleright \Delta$ then also $\Gamma + L'; L \vdash P \triangleright \Delta$.*

Proof. Immediate from Lemma 5.7. □

Lemma 5.9. *If $\Gamma; L \vdash P \triangleright \Delta$ and $\Delta(k) \neq \mathtt{end}$ then $k \in \mathsf{fn}(P)$.*

Proof. Straightforward induction. □

Lemma 5.10. *Suppose $\Gamma; L \vdash P \triangleright \Delta, k : T$ with $\Delta, k : T$ balanced, $T \neq \mathtt{end}$, and $\overline{k} \notin \mathsf{fn}(P)$. Then $\overline{k} \notin \mathsf{dom}(\Delta)$.*

Proof. Supposed for a contradiction $\overline{k} \in \mathsf{dom}(\Delta)$. Because $\Delta, k : !.T$ balanced, $\Delta(\overline{k}) \neq \mathtt{end}$. By Lemma 5.9 we thus have $\overline{k} \in \mathsf{fn}(P)$; contradiction. □

We can now formulate the core lemma which will subsequently be used to prove subject reduction. For the formulation of the lemma, we will slightly abuse notation and consider the range of the $\mathsf{sel}(-)$ operator as an empty or singleton set rather than the empty or singleton string as which it was originally defined.

Lemma 5.11. *Suppose that $\Gamma; L \vdash P \triangleright \Delta$ with $P \xrightarrow{\lambda} Q$. Then there exists a type transition $\Delta \xrightarrow{\delta} \Delta'$ with $\delta \simeq \lambda$, such that $\Gamma + \mathsf{sel}(\delta); (L \setminus \mathsf{res}(\delta)) \cup \mathsf{req}(\delta) \vdash Q \triangleright \Delta'$. Moreover, if Δ balanced, then also Δ' balanced.*

Proof. By induction on the derivation of the transition.

Case [C-Out]**.** We have $k!\langle e\rangle.P \xrightarrow{k!v} P$ with $\overline{k} \notin P$ and $\Gamma; L \vdash k!\langle e\rangle.P \triangleright \Delta, k : !.T$. By typing $\Gamma; L \vdash P \triangleright \Delta, k : T$. By [F-Lift] we have $k : !.T \xrightarrow{k:!} k : T$. By [F-Par] $\Delta, k : !.T \xrightarrow{k:!} \Delta, k : T$; Observing that $k : ! \simeq k!v$ and $\mathsf{res}(k : !) = \mathsf{sel}(k : !) = \mathsf{req}(k : !) = \emptyset$ we have found the requisite type transition.

Now suppose $\Delta, k : !.T$ balanced; we must show $\Delta, k : T$ balanced. It is sufficient to show $\overline{k} \notin \mathsf{dom}(\Delta)$. But this follows from Lemma 5.10.

Case [C-In]**.** We have $k?(x).P \xrightarrow{k?v} P\{v/x\}$ with $\overline{k} \notin \mathsf{fn}(P)$ and $\Gamma; L \vdash k?(x).P \triangleright \Delta, k : ?.T$. By typing $\Gamma; L \vdash P \triangleright \Delta, k : T$. By [F-Lift] and [F-Par], $\Delta, k : ?.T \xrightarrow{k:?} \Delta, k : T$. By Lemma 5.5 we have $\Gamma; L \vdash P\{v/x\} \triangleright \Delta, k : T$. Observing that $\mathsf{req}(k : ?T[L']) = \mathsf{sel}(k : ?T[L']) = \mathsf{res}(k : ?T[L']) = \emptyset$ and that $k : ? \simeq k?v$ we have found the requisite transition and typing. Preservation of balance follows from Lemma 5.10.

Case [C-Bra]**.** We have $k?\{l_i.P_i\} \xrightarrow{k\&l_i} P_i$ and $\Gamma; L \vdash k?\{l_i.P_i\}_{i\in I} \triangleright \Delta, k : \&\{l_i[L_i].T_i\}_{i\in I}$. By typing we have $\Gamma + l_i; (L \setminus \{l_i\}) \cup L_i \vdash P_i \triangleright \Delta, k : T_i$. By [F-Lift] and [F-Par] we have $\Delta, k : \&\{l_i[L_i].T_i\}_{i\in I} \xrightarrow{k:\&l_i[L_i]} \Delta, k : T_i$. Observing that $\mathsf{req}(k : \&l_i[L_i]) = L_i$, $\mathsf{sel}(k : \&l_i[L_i]) = \mathsf{res}(k : \&l_i[L_i]) = \{l_i\}$ and that $k : \oplus l_i[L_i] \simeq k\&l_i$, we have found the requisite type transition. Preservation of balance follows from Lemma 5.10.

Case [C-Sel]**.** We have $k!l.P \xrightarrow{k\oplus l_i} P$ and $\Gamma; L \vdash k!l_i.P \triangleright \Delta, k : \oplus\{l_i[L_i].T_i\}_{i\in I}$. By typing $\Gamma + l_i; (L \setminus \{l_i\}) \cup L_i \vdash P \triangleright \Delta, k : T_i$. By [F-Lift] and [F-Par] we have $\Delta, \oplus\{l_i[L_i].\ T_i\}_{i\in I} \xrightarrow{k:\oplus l_i[L_i]} \Delta, T_i$. Observing that $\mathsf{req}(k : \oplus l_i[L_i]) = L_i$,

$\mathsf{sel}(k : \oplus l_i[L_i]) = \mathsf{res}(k : \oplus l_i[L_i]) = \{l_i\}$ and that $k : \oplus l_i[L_i] \simeq k \oplus l_i$, we have found the requisite type transition. Preservation of balance follows from Lemma 5.10.

Case [C-ParL]**.** We have $P \mid P' \xrightarrow{\lambda} Q \mid P'$ with $\overline{\mathsf{subj}(\lambda)} \notin \mathsf{fn}(P')$ and $\Gamma; L \vdash P \mid P' \triangleright \Delta$. By typing we have for some $L_1 \cup L_2 = L$ and $\Delta_1 \cup \Delta_2$ that $\Gamma; L_1 \vdash P \triangleright \Delta_1$ and $\Gamma; L_2 \vdash P' \triangleright \Delta_2$. By the induction hypothesis, we have a transition $\Delta_1 \xrightarrow{\delta} \Delta_1'$ with $\Gamma + \mathsf{sel}(\delta); L_1 \setminus \mathsf{res}(\delta) \cup \mathsf{req}(\delta) \vdash Q \triangleright \Delta_1'$ and $\delta \simeq \lambda$. By Lemma 5.8 we find also $\Gamma + \mathsf{sel}(\delta); L_2 \vdash P' \triangleright \Delta_2$. By Lemma 4.6 $\mathsf{dom}(\Delta_1) = \mathsf{dom}(\Delta_1')$ so Δ_1', Δ_2 is defined, and hence by [G-Par] we have $\Gamma + \mathsf{sel}(\delta); L_1 \setminus \mathsf{res}(\delta) \cup \mathsf{req}(\delta) \cup L_2 \vdash Q \mid P' \triangleright \Delta_1', \Delta_2$. This is not exactly the form we need, but observing that

$$(L_1 \cup L_2) \setminus \mathsf{res}(\delta) \cup \mathsf{req}(\delta) \subseteq (L_1 \setminus \mathsf{res}(\delta)) \cup \mathsf{req}(\delta) \cup L_2,$$

we find again by Lemma 5.3 that $\Gamma + \mathsf{sel}(\delta); (L_1 \cup L_2) \setminus \mathsf{res}(\delta) \cup \mathsf{req}(\delta) \vdash Q \mid P' \triangleright \Delta_1', \Delta_2$. By [F-Par] $\Delta_1, \Delta_2 \xrightarrow{\delta} \Delta_1', \Delta_2$, and we have found the requisite type transition.

Now suppose Δ_1, Δ_2 balanced. By Lemma 4.8 it is sufficient to prove that $\overline{\mathsf{subj}(\delta)} \notin \mathsf{dom}(\Delta_1, \Delta_2)$. If $\mathsf{subj}(\delta) = \tau$ this is trivial, so say $\mathsf{subj}(\delta) = k$ and suppose for a contradiction $\overline{k} \in \mathsf{dom}(\Delta_1, \Delta_2)$. We must have $\delta = k : \rho$ and because $\delta \simeq \lambda$ we must have $\mathsf{subj}(\lambda) = \mathsf{subj}(\delta) = k$. By Lemma 2.2 $\overline{k} \notin \mathsf{fn}(Q \mid P')$. By Lemma 4.7 we have $\Delta_1 = \Delta_1'', k : S$ with $S \neq \mathsf{end}$. Because Δ_1, Δ_2 balanced, $(\Delta_1, \Delta_2)(\overline{k}) \bowtie S$ and so $(\Delta_1, \Delta_2)(\overline{k}) \neq \mathsf{end}$.

Suppose first $\overline{k} \in \mathsf{dom}(\Delta_1)$. Then $\overline{k} \in \mathsf{dom}(\Delta_1'')$, so also $\Delta''(\overline{k}) \neq \mathsf{end}$, and it follows that $\Delta_1'(\overline{k}) = \Delta_1''(\overline{k}) \neq \mathsf{end}$. By Lemma 5.9 $\overline{k} \in \mathsf{fn}(Q)$, contradicting $\overline{k} \notin \mathsf{fn}(Q \mid P')$.

Suppose instead $\overline{k} \in \mathsf{dom}(\Delta_2)$. Then immediately by Lemma 5.9 $\overline{k} \in \mathsf{fn}(P')$, contradicting $\overline{k} \notin \mathsf{fn}(Q \mid P')$.

Case [C-Com1]**.** We have

$$\frac{P_1 \xrightarrow{\overline{k}!v} P_1' \qquad P_2 \xrightarrow{k?v} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'}$$

and

$$\frac{\Gamma; L_1 \vdash P_1 \triangleright \Delta_1 \qquad \Gamma; L_2 \vdash P_2 \triangleright \Delta_2}{\Gamma; L_1 \cup L_2 \vdash P_1 \mid P_2 \triangleright \Delta_1, \Delta_2}$$

By the induction hypothesis we find $\Delta_i \xrightarrow{\delta_i} \Delta_i'$ s.t. $\Gamma + \mathsf{sel}(\delta_i); L_i \setminus \mathsf{res}(\delta_i) \cup \mathsf{req}(\delta_i) \vdash P_i \triangleright \Delta_i'$ with $\delta_1 \simeq \overline{k}!v$ and $\delta_2 \simeq k?v$. It follows that $\delta_1 = k : !$ and $\delta_2 = k : ?$ whence $\mathsf{res}(\delta_1) = \mathsf{req}(\delta_2) = \mathsf{res}(\delta_2) = \mathsf{req}(\delta_1) = \emptyset$ and $\mathsf{sel}(\delta_1) = \mathsf{sel}(\delta_2) = \epsilon$. By Lemma 4.6 Δ_1', Δ_2' defined, and so by by [F-Com1] we have $\Delta_1, \Delta_2 \xrightarrow{\tau} \Delta_1', \Delta_2'$. Noting that $\tau \simeq \tau$ and that $\Gamma + \mathsf{sel}(\delta_1) + \mathsf{sel}(\delta_2) = \Gamma$, we have the required type transition. Since $\mathsf{subj}(\tau) = \tau$ and so $\overline{\mathsf{subj}(\tau)} \notin \mathsf{dom}(\Delta_1, \Delta_2)$, it follows from Lemma 4.8 that Δ_1', Δ_2' is balanced when Δ_1, Δ_2 is.

Case [C-Com2]**.** We have

$$\frac{P_1 \xrightarrow{\overline{k} \oplus l} P_1' \qquad P_2 \xrightarrow{k \& l} P_2'}{P_1 \mid P_2 \xrightarrow{\tau : l} P_1' \mid P_2'}$$

and

$$\frac{\Gamma; L_1 \vdash P_1 \triangleright \Delta_1 \qquad \Gamma; L_2 \vdash P_2 \triangleright \Delta_2}{\Gamma; L_1 \cup L_2 \vdash P_1 \mid P_2 \triangleright \Delta_1, \Delta_2}$$

By induction we find $\Delta_i \xrightarrow{\delta_i} \Delta_i'$ s.t. $\Gamma + \mathsf{sel}(\delta_i); L_i \setminus \mathsf{res}(\delta_i) \cup \mathsf{req}(\delta_i) \vdash P_i \triangleright \Delta_i'$ with $\delta_1 \simeq \overline{k} \oplus l$ and $\delta_2 \simeq k \& l$. It follows that for some L_1', L_2' we have $\delta_1 = \overline{k} : \oplus l[L_1']$ and $\delta_2 = k : \& l[L_2']$, and so $\mathsf{req}(\delta_1) = L_1'$ and $\mathsf{req}(\delta_2) = L_2'$, $\mathsf{res}(\delta_1) = \mathsf{res}(\delta_2) = \{l\}$, and $\mathsf{sel}(\delta_1) = \mathsf{sel}(\delta_2) = l$. By Lemma 4.6 Δ_1', Δ_2' defined, and so we find a transition $\Delta_1, \Delta_2 \xrightarrow{\tau : l, L_1' \cup L_2'} \Delta_1', \Delta_2'$ by [F-Com2]. By [G-Par] we find $\Gamma + l; L_1 \setminus \{l\} \cup L_1' \cup L_2 \setminus \{l\} \cup L_2' \vdash P_1' \mid P_2' \triangleright \Delta_1', \Delta_2'$. Noting that $\tau : l, L_1' \cup L_2' \simeq \tau : l$ and

$$\begin{aligned} L_1 \setminus \{l\} \cup L_1' \cup L_2 \setminus \{l\} \cup L_2' &= (L_1 \cup L_2) \setminus \{l\} \cup L_1' \cup L_2' \\ &= (L_1 \cup L_2) \setminus \mathsf{res}(\tau : l, L_1' \cup L_2') \cup \mathsf{req}(\tau : l, L_1' \cup L_2'), \end{aligned}$$

we have the required type transition. Since $\mathsf{subj}(\tau : l, L_1' \cup L_2') = \tau$ and so $\overline{\mathsf{subj}(\tau : l, L_1' \cup L_2')} \notin \mathsf{dom}(\Delta_1, \Delta_2)$, it follows from Lemma 4.8 that Δ_1', Δ_2' is balanced when Δ_1, Δ_2 is.

Case [C-Rec]**.** We have

$$\frac{P\{\mathrm{rec}\, X.P/X\} \xrightarrow{\lambda} Q}{\mathrm{rec}\, X.P \xrightarrow{\lambda} Q}$$

and

$$\frac{\Gamma, X : (\emptyset, I, \Delta); I \vdash P \triangleright \Delta \qquad L \subseteq I}{\Gamma; L \vdash \mathrm{rec}\, X.P \triangleright \Delta}$$

It follows by [G-Rec] that also $\Gamma; I \vdash \mathrm{rec}\, X.P \triangleright \Delta$ and by Lemma 5.8 that $\Gamma, X : (\emptyset, I, \Delta); L \vdash P \triangleright \Delta$. It then follows by Lemma 5.4 that

$$\Gamma; L \vdash P\{\mathrm{rec}\, X.P/X\} \triangleright \Delta.$$

By the induction hypothesis we find a balance-preserving type transition $\Delta \xrightarrow{\delta} \Delta'$ with $\delta \simeq \lambda$ and $\Gamma + \mathsf{sel}(\delta); L \setminus \mathsf{res}(\delta) \cup \mathsf{req}(\delta) \vdash Q \triangleright \Delta'$.

Case [C-Prec0]**.** We have

$$\frac{e \Downarrow 0 \qquad Q \xrightarrow{\lambda} R}{(\mathrm{rec}^e\, X(i).P; Q) \xrightarrow{\lambda} R}$$

and

$$\frac{\Gamma, X : (L', \Delta); L' \vdash P \triangleright \Delta \qquad \Gamma; L' \vdash Q \triangleright \Delta \qquad L \subseteq L'}{\Gamma; L \vdash (\mathrm{rec}^e\, X(i).P; Q) \triangleright \Delta}\,.$$

By Lemma 5.8 we have also $\Gamma; L \vdash Q \triangleright \Delta$, and so by the induction hypothesis we find the required balance-preserving type transition.

Case [C-PrecN]**.** We have

$$\frac{e \Downarrow n+1 \qquad P\{n/i\}\{(\mathrm{rec}^n\, X(i).P; Q)/X\} \xrightarrow{\lambda} R}{(\mathrm{rec}^e\, X(i).P; Q) \xrightarrow{\lambda} R}$$

and again

$$\frac{\Gamma, X : (L', \Delta); L' \vdash P \triangleright \Delta \qquad \Gamma; L' \vdash Q \triangleright \Delta \qquad L \subseteq L'}{\Gamma; L \vdash (\mathrm{rec}^e\, X(i).P; Q) \triangleright \Delta}\ .$$

By [G-RecP] it follows that $\Gamma; L' \vdash (\mathrm{rec}^n\, X(i).P; Q) \triangleright \Delta$. By Lemmas 5.5 and 5.3 we have $\Gamma, X : (L', \Delta); L \vdash P\{n/i\} \triangleright \Delta$. Finally, by Lemma 5.4 we have

$$\Gamma; L \vdash P\{n/i\}\{(\mathrm{rec}^n\, X(i).P; Q)/X\} \triangleright \Delta,$$

and the requisite balance-preserving type transition follows by the induction hypothesis.

Case [C-CondT] and [C-CondF]**.** We have,

$$\frac{e \Downarrow \mathsf{true} \qquad P \xrightarrow{\lambda} P'}{\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\lambda} P'}$$

and

$$\frac{\Gamma; L \vdash P \triangleright \Delta \qquad \Gamma; L \vdash Q \triangleright \Delta}{\Gamma; L \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta}\ ,$$

and the requisite balance-preserving type transition follows from the induction hypothesis. The other case is the same. □

Theorem 5.12 (Subject reduction). *Suppose that* $\cdot; L \vdash P \triangleright \Delta$ *with and* $P \xrightarrow{\lambda} Q$*. Then there exists a type transition* $\Delta \xrightarrow{\delta} \Delta'$ *with* $\delta \simeq \lambda$*, such that* $\cdot; (L \setminus \mathsf{res}(\delta)) \cup \mathsf{req}(\delta) \vdash Q \triangleright \Delta'$*. Moreover, if* Δ *balanced then also* Δ' *balanced.*

Proof. Immediate from the Lemma 5.11. □

Example 5.13. Wrt. the typing system of Figure G, the process $P(D)$ is typable wrt. the types we postulated for it in Example 3.4. The process $P(D_0)$ on the other hand is not. That is, we have $\cdot; \emptyset \vdash P(D) \triangleright k : T_P, o^+ : T_D, o^- : \overline{T_D}$, but the same does *not* hold for $P(D_0)$. We also exemplify a typing judgment with non-trivial guaranteed responses. The process D, the order-fulfillment part of $P(D)$, can in fact be typed

$$\cdot; \{\mathsf{SI}\} \vdash D \triangleright k : \mu t'.\, \oplus\{\mathsf{DI}.!.t', \mathsf{SI}.!.\mathtt{end}\},\ o^- : \overline{T_D}$$

Note the left-most $\{\mathsf{SI}\}$, indicating intuitively that this process will eventually select SI in *any* execution. The process D has this property essentially because it is implemented by bounded recursion. □

6 Liveness

We now prove that a lock-free process well-typed under our liveness typing system is indeed live as defined in Def. 4.4.

Liveness refers to the notion of maximal transition sequences, which is defined below and in turn relies on (weak) fairness. For defining lock-freedom and fairness, we must track occurrences of prefixes across transitions. This is straightforward in the absence of a structural congruence; refer to [12] for a formal treatment.

Definition 6.1. *A* prefix *M is a process on one of the forms $k!\langle e\rangle.P$, $k?(x).P$, $k?\{l_i.P_i\}$, or $k!l.P$. An* occurence *of a prefix M in a process P is a path in the abstract syntax tree of P to a subterm on the form M (see [12] for details).*

Definition 6.2. *An occurrence of a prefix P in M where $P \xrightarrow{\lambda} Q$ is* preserved *by the latter if M has the same occurrence in Q;* executed *otherwise. It is* enabled *if it is executed by some transition, and* top-level *if it is not nested in another prefix. Suppose we have an occurrence of a prefix M in P and a transition $P \xrightarrow{\lambda} Q$. This transition*

1. preserves *the occurence of M in P if M has the same occurrence in Q, and*
2. executes *the occurrence of M in P iff it does not preserve it.*

The occurrence of M in P is

3. enabled *iff P has a transition executing it, and*
4. top-level *if it is nested inside no other prefix.*

Lemma 6.3. *Occurrences have the following properties.*

1. *If an occurrence is enabled, it is also top-level.*
2. *If $P \xrightarrow{\lambda} Q$ preserves a top-level occurrence of a prefix M in P, then that occurence is also top-level in Q.*
3. *If $P \xrightarrow{\lambda} Q$ then there exists an occurrence of a prefix M in P which is executed by that transition.*

Definition 6.4. *An infinite transition sequence $s = (P_i, \lambda_i)_{i\in\mathbb{N}}$ is* fair *iff whenever a prefix M occurs enabled in P_n then some $m \geq n$ has $P_m \xrightarrow{\lambda_m} P_{m+1}$ executing that occurence.*

Definition 6.5. *A transition sequence s is* terminated *iff it has finite length n and $P_n \not\longrightarrow$. It is* maximal *iff it is finite and terminated or infinite and fair.*

We define lock-freedom in the spirit of [18]; notice that the present definition strictly generalises fairness.

Definition 6.6. *A maximal transition sequence (P_i, λ_i) is lock-free iff whenever there is a top-level occurence of a prefix M in P_i, then there exists some $j \geq i$ s.t. $P_j \xrightarrow{\lambda_j} P_{j+1}$ executes that occurrence. A process is lock-free iff all its transition sequences are.*

The central liveness result hinges on the following proposition, which links the typing judgment to the semantics of a process.

Definition 6.7. *For a process transition label λ, define* $\mathsf{sel}(\lambda)$ *by*

$$\mathsf{sel}(k!v) = \mathsf{sel}(k?v) = \mathsf{sel}(\tau) = \emptyset \, \mathsf{sel}(k\&l) = \mathsf{sel}(k \oplus l) = \mathsf{sel}(\tau : l) = l$$

Given a trace α we lift $\mathsf{sel}(-)$ *pointwise, that is,* $\mathsf{sel}(\alpha) = \{\mathsf{sel}(\lambda) \mid \alpha = \phi\lambda\alpha'\}$.

Note that $\delta \simeq \lambda$ implies $\mathsf{sel}(\lambda) = \mathsf{sel}(\delta)$.

Lemma 6.8. *For any transition sequence s of $P \mid Q$, there exists transition sequences $p = (P_i, \beta_i)_{i \in |p|}$ and $q = (Q_i, \delta_i)_{i \in |q|}$ and monotone surjective maps $u : |s| \to |p|$ and $v : |s| \to |q|$ such that $s = (P_{u(i)} | Q_{v(i)}), \alpha_i)_{i \in |s|}$ and $\mathsf{sel}(\beta) \cup \mathsf{sel}(\delta) = \mathsf{sel}(\alpha)$.*

Proof. We prove the existence of such functions for finite s; the result for infinite s follows. So suppose s finite and write it $s = (S_i, \alpha_i)_{|s|}$. We proceed by induction on the length of s. First, a bit of notation: when $\alpha = \alpha_1 \dots \alpha_n$ we define $\mathsf{cut}(\alpha) = \alpha_1 \dots \alpha_{n-1}$. Now, for $|s| = 1$, the identity functions suffice. Suppose instead $|s| = n + 1$, and consider the last transition $S_n \xrightarrow{\alpha_n} S_{n+1}$. By the induction hypothesis we have transition sequences p, q with labels β, δ and maps u, v such that $S_n = P_{u(n)} \mid Q_{v(n)}$ and $\mathsf{sel}(\beta) \cup \mathsf{sel}(\delta) = \mathsf{sel}(\mathsf{cut}(\alpha))$ etc. Notice that because u, v surjective and monotone, p, q must have lengths $u(n)$ and $v(n)$, respectively. We proceed by cases on the derivation of this last transition.

Case [C-ParL]**.** We must in this case have

$$\frac{P_{u(n)} \xrightarrow{\alpha_n} R}{P_{u(n)} \mid Q_{v(n)} \xrightarrow{\alpha_n} R \mid Q_{v(n)} = S_{n+1}} \, .$$

Extend p to p' by taking $P_{u(n)+1} = R$ and $\beta_{u(n)} = \alpha_n$; and extend u, v by taking $u(n+1) = u(n) + 1$ and $v(n+1) = v(n)$ and we have found the requisite transition sequences and maps. It is now sufficient to note that

$$\begin{aligned} \mathsf{sel}(\alpha) &= \mathsf{sel}(\alpha_n) \cup \mathsf{sel}(\mathsf{cut}(\alpha)) \\ &= \mathsf{sel}(\beta_{u(n)}) \cup \mathsf{sel}(\beta) \cup \mathsf{sel}(\delta) \\ &= \mathsf{sel}(\beta') \cup \mathsf{sel}(\delta) \, . \end{aligned}$$

Case [C-Com1]**.** We must have in this case

$$\frac{P_{u(n)} \xrightarrow{\overline{k}!v} P' \qquad Q_{v(n)} \xrightarrow{k?v} Q'}{P_{u(n)} \mid Q_{v(n)} \xrightarrow{\tau = \alpha_n} P' \mid Q' = S_{n+1}} \, .$$

Extend p to p' by taking $P_{u(n)+1} = P'$ and $\beta_{u(n)} = \overline{k}!v$; and similarly extend q to q' by taking $Q_{v(n)+1} = Q'$ and $\delta_{v(n)} = k?v$. Extending also u, v by $u(n+1) =$

$u(n)+1$ and $v(n+1) = v(n)+1$ we have found the requisite transition sequences and maps. It is now sufficient to note that

$$\begin{aligned}\mathsf{sel}(\alpha) &= \mathsf{sel}(\tau) \cup \mathsf{sel}(\mathsf{cut}(\alpha)) \\ &= \mathsf{sel}(\beta) \cup \mathsf{sel}(\delta) \\ &= \mathsf{sel}(\overline{k}!v) \cup \mathsf{sel}(\beta') \cup \mathsf{sel}(k?v) \cup \mathsf{sel}(\delta') \\ &= \mathsf{sel}(\beta') \cup \mathsf{sel}(\delta') \ .\end{aligned}$$

□

Case [C-Com2]**.** We must have in this case

$$\frac{P_{u(n)} \xrightarrow{\overline{k}\oplus l} P' \qquad Q_{v(n)} \xrightarrow{k\&l} Q'}{P_{u(n)} \mid Q_{v(n)} \xrightarrow{\tau:l=\alpha_n} P' \mid Q' = S_{n+1}} \ .$$

Extend p to p' by taking $P_{u(n)+1} = P'$ and $\beta_{u(n)} = \overline{k} \oplus l$; and similarly extend q to q' by taking $Q_{v(n)+1} = Q'$ and $\delta_{v(n)} = k\&l$. Extending also u, v by $u(n+1) = u(n)+1$ and $v(n+1) = v(n)+1$ we have found the requisite transition sequences and maps. It is now sufficient to note that

$$\begin{aligned}\mathsf{sel}(\alpha) &= \mathsf{sel}(\tau : l) \cup \mathsf{sel}(\mathsf{cut}(\alpha)) \\ &= \{l\} \cup \mathsf{sel}(\beta) \cup \mathsf{sel}(\delta) \\ &= \mathsf{sel}(\overline{k} \oplus l) \cup \mathsf{sel}(\beta) \cup \mathsf{sel}(k\&l) \cup \mathsf{sel}(\delta) \\ &= \mathsf{sel}(\beta') \cup \mathsf{sel}(\delta') \ .\end{aligned}$$

Lemma 6.9. *If s is a maximal lock-free transition sequence of $P \mid Q$ with trace α, then there exists maximal lock-free transition sequences p, q of P, Q with traces β, δ, respectively, such that* $\mathsf{sel}(\alpha) = \mathsf{sel}(\beta) \cup \mathsf{res}(\delta)$.

Proof. By Lemma 6.8 we find transition sequences $p = (P_i, \beta_i)_{i\in|p|}$ and $q = (Q_i, \delta_i)_{i\in|q|}$ and maps u, v such that s can be written $s = (s_i, \alpha_i)_{i\in|s|} = (P_{u(i)} \mid Q_{v(i)},\ \alpha_i)_{i\in|s|}$ and $\mathsf{sel}(\alpha) = \mathsf{sel}(\beta) \cup \mathsf{sel}(\delta)$. It remains to prove that these p, q are maximal and lock-free. Suppose for a contradiction that p is not; the case for q is similar. Then either (A) p is maximal but not lock-free, or (B) p is not maximal.

We consider first (A); p maximal but not lock-free. Then some top-level occurrence of a prefix M sits in each P_i when $i \geq n$ for some n. But then for $j \geq u^{-1}(n)$ we must have $s_j = (P_{u(n)}, Q_{v(n)})$ contradicting s lock-free.

Consider now (B); p not maximal. Then either (1) p is finite and can be extended by a transition λ, or (2) p is infinite but not fair.

Suppose (1) that is, p of finite length n and $P_n \xrightarrow{\lambda}$. By Lemma 6.3(3) P_n must have an occurrence of an enabled prefix M. By Lemma 6.3(1) this occurrence is top-level. But for $i \geq u^{-1}(n)$, $s_i = (P_{u(i)} \mid Q_{v(i)})$ and so there is a top-level occurrence of M in each such s_i, contradicting $P \mid Q$ lock-free.

Suppose instead (2), that is, p infinite but not fair. Then there exists a P_n and an occurrence of an enabled prefix M in P_n s.t. no β_j with $j \geq n$ executes that

occurrence. By definition, every $P_j \xrightarrow{\beta_j} P_{j+1}$ then preserves that occurrence. By Lemma 6.3(1) the occurrence in P_n is top-level, and so by Lemma 6.3(2) it also is in every P_j. But for $j \geq u^{-1}(n)$, $s_j = (P_{u(j)} \mid Q_{v(j)})$, and so we have found a top-level occurrence of M in each such s_j, contradicting $P \mid Q$ lock-free. □

Definition 6.10. *A process P is* simple for X *iff*

1. *no process variable but X occurs free in P, and*
2. $\mathbf{0}$ *is not a sub-term of P, and*
3. *neither* $\mathrm{rec}\, Y.Q$ *nor* $(\mathrm{rec}^e\, Y(i).Q; R)$ *is a sub-term of P,*
4. *$Q \mid R$ is not a sub-term of P.*

Observe that by convention, in $(\mathrm{rec}^e\, X(i).P; Q)$, P is simple for X.

Lemma 6.11. *If P simple for X and $s = (P_i, \lambda_i)_i$ is a maximal lock-free transition sequence of $P\{Q/X\}$, then $Q \xrightarrow{\lambda_{j-1}} P_j$ for some $j > 1$.*

Proof. By induction on P.
Case "$\mathbf{0}$"**.** Impossible: not simple for X.
Case "$k!\langle e\rangle.P$"**.** Clearly $(P_{i+1}, \lambda_{i+1})_i$ is a maximal lock-free transition sequence of $P\{\tilde{v}/\tilde{x}\}$. By the induction hypothesis $Q \xrightarrow{\lambda_j} P_j$ for some $j > 2$.
Case "$k?(x).P$"**.** Clearly $s' = (P_{i+1}, \lambda_{i+1})_i$ is a maximal lock-free transition sequence of $P\{Q/X\}\{\tilde{v}/\tilde{x}\}$. Because x bound, it is fresh for Q, so $P\{\tilde{v}/\tilde{x}\}\{Q/X\} = P\{Q/X\}\{\tilde{v}/\tilde{x}\}$ and s' is a maximal lock-free transition sequence of the latter. But then by the induction hypothesis $Q \xrightarrow{\lambda_j} P_j$ for some $j > 2$.
Case "$k?\{l_i.P_i\}_{j\in J}$"**.** Like $k!\langle e\rangle.P$. **Case** "$k!l.P$"**.** Like $k!\langle e\rangle.P$
Case "$P \mid R$"**.** Impossible: not simple for X.
Case "$\mathrm{rec}\, X.P$"**.** Impossible: not simple for X.
Case "$(\mathrm{rec}^e\, Y(i).P; R)$"**.** Impossible: not simple for X.
Case "$Y[\tilde{k}]$"**.** By P simple for X we must have $X = Y$ whence s is a transition sequence of $X\{Q/X\} = Q$; clearly $Q \xrightarrow{\lambda_1} P_2$.
Case "$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ R$"**.** Like $k?\{l_i.P_i\}_{j\in J}$. □

Lemma 6.12. *If $s = (P_i, \lambda_i)_i$ is a maximal lock-free transition sequence of $(\mathrm{rec}^e\, X(i).P; Q)$ then $Q \xrightarrow{\lambda_{j-1}} P_j$ for some $j > 1$.*

Proof. By induction on n. If $n = 0$ then s is a transition sequence of Q iff it is of $(\mathrm{rec}^0\, X(i).P; Q)$, so clearly $Q \xrightarrow{\lambda_1} P_2$. If instead $n = m + 1$ observe that

$$(\mathrm{rec}^{m+1}\, X(i).P; Q) \xrightarrow{\lambda_1} R \qquad \text{iff} \qquad P\{m/i\}\{(\mathrm{rec}^m\, X(i).P; Q)/X\} \xrightarrow{\lambda_1} R\,.$$

Take s' to be the same as s except $P_1 = P\{m/i\}\{(\mathrm{rec}^m\, X(i).P; Q)/X\}$. Note that s' is maximal and lock-free. By convention P and so $P\{m/i\}$ is simple for X. Then by Lemma 6.12 for some j we have $Q \xrightarrow{\lambda_{j-1}} P_j$. □

Definition 6.13. *When P is a process, we define $\mathcal{A}(P)$ inductively as follows.*

$$\begin{aligned}
\mathcal{A}(\mathbf{0}) &= \emptyset \\
\mathcal{A}(k!\langle e\rangle.P) &= \mathcal{A}(P) \\
\mathcal{A}(k?(x).P) &= \mathcal{A}(P) \\
\mathcal{A}(k?\{l_i.P_i\}_{i\in I}) &= \bigcap_{i\in I}(\{l_i\} \cup \mathcal{A}(P_i)) \\
\mathcal{A}(k!l.P) &= \{l\} \cup \mathcal{A}(P) \\
\mathcal{A}(P \mid Q) &= \mathcal{A}(P) \cup \mathcal{A}(Q) \\
\mathcal{A}(\mathrm{rec}\, X.P) &= \mathcal{A}(P) \\
\mathcal{A}(\mathrm{rec}^e\, X(i).P; Q) &= \mathcal{A}(Q) \\
\mathcal{A}(X[\tilde{k}]) &= \emptyset \\
\mathcal{A}(\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q) &= \mathcal{A}(P) \cap \mathcal{A}(Q)
\end{aligned}$$

Proposition 6.14. *If $s = (P_i, \alpha_i)_i$ is a maximal lock-free transition sequence of $P\{Q/X\}$ $\mathcal{A}(P) \subseteq \mathsf{sel}(\alpha)$.*

Proof. First, notation: if α is a sequence $\alpha_1\alpha_2\cdots$ we define $\mathsf{shift}(\alpha) = \alpha_2\cdots$. We proceed by induction on P.

Case "$\mathbf{0}$"**.** Immediate from $\mathcal{A}(\mathbf{0}) = \emptyset$.

Case "$k!\langle e\rangle.P$"**.** Clearly $(P_{i+1}, \alpha_{i+1})_i$ is a maximal lock-free transition sequence of $P\{\tilde{Q}/\tilde{X}\}$. By the induction hypothesis $\mathcal{A}(P) \subseteq \mathsf{sel}(\mathsf{shift}(\alpha)) = \mathsf{sel}(\alpha)$.

Case "$k?(x).P$"**.** Clearly $(P_{i+1}, \alpha_{i+1})_i$ is a maximal lock-free transition sequence of $P\{\tilde{Q}/\tilde{X}\}\{v/x\}$ for some v. As x is bound $P\{\tilde{Q}/\tilde{X}\}\{v/x\} = P\{v/x\}\{\tilde{Q}/\tilde{X}\}$. Using the induction hypothesis $\mathcal{A}(P) = \mathcal{A}(P\{v/x\}) \subseteq \mathsf{sel}(\mathsf{shift}(\alpha)) = \mathsf{res}(()\alpha)$.

Case "$k?\{l_i.P_i\}_{i\in I}$"**.** Like $k!\langle e\rangle.P$.

Case "$k!l.P$"**.** Like $k!\langle e\rangle.P$.

Case "$P \mid R$"**.** By Lemma 6.9 there exists traces maximal lock-free transition sequences p, q of $P\{\tilde{Q}/\tilde{X}\}, R\{\tilde{Q}/\tilde{X}\}$ with traces β, δ s.t. $\mathsf{sel}(\beta) \cup \mathsf{sel}(\delta) = \mathsf{sel}(\alpha)$. Using the induction hypothesis we find $\mathcal{A}(P) \cup \mathcal{A}(R) \subseteq \mathsf{sel}(\beta) \cup \mathsf{sel}(\delta) = \mathsf{sel}(\alpha)$.

Case "$\mathrm{rec}\, Y.P$"**.** s is lock-free maximal transition sequence of $\mathrm{rec}\, Y.P\{\tilde{Q}/\tilde{X}\}$. Then taking s' to be the same as s except $P_1 = P\{\tilde{Q}/\tilde{X}\}\{\mathrm{rec}\, Y.(P\{\tilde{Q}/\tilde{X}\})/Y\}$ we have a maximal lock-free transition sequence of the latter, also with trace α. Using the induction hypothesis $\mathcal{A}(\mathrm{rec}\, X.P) = \mathcal{A}(P) \subseteq \mathsf{sel}(\alpha)$.

Case "$(\mathrm{rec}^e\, Y(i).P; R)$"**.** Again, s is a lock-free maximal transition sequence of $(\mathrm{rec}^e\, Y(I).P; R)\{\tilde{Q}/\tilde{X}\}$. By Lemma 6.12 for some $j > 1$ we have $R \xrightarrow{\lambda_{j-1}} P_j$, and so

$$R \xrightarrow{\lambda_{j-1}} P_j \xrightarrow{\lambda_j} P_{j+1}\cdots$$

is a lock-free maximal transition sequence of R. By the induction hypothesis, $\mathcal{A}((\mathrm{rec}^e\, Y(i).P; R)) = \mathcal{A}(R) \subseteq \mathsf{sel}(\mathsf{shift}^{j-2}(\alpha)) \subseteq \mathsf{sel}(\alpha)$.

Case "$Y[\tilde{k}]$"**.** Immediate from $\mathcal{A}(Y[\tilde{k}]) = \emptyset$.

Case "$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ R$"**.** Like $k?\{l_i.P_i\}_{j\in J}$.

Lemma 6.15. *Suppose that* $\Gamma; L \vdash P \triangleright \Delta$. *Define mappings* $M((A, I, \Delta)) = A$ *and* $M((I, \Delta)) = I$, *and*

$$M(\Gamma) = \bigcup_{X \in \mathsf{dom}(\Gamma)} M(\Gamma(X)) \, .$$

Then $L \setminus M(\Gamma) \subseteq \mathcal{A}(P)$.

Proof. By induction on the derivation of $\Gamma; L \vdash P \triangleright \Delta$.

Case [G-Inact]**.** By typing, $L = \emptyset$.

Case [G-Out]**.** $\mathcal{A}(k!\langle e\rangle.P) = \mathcal{A}(P) \supseteq L \setminus M(\Gamma)$, the latter by typing and the induction hypothesis.

Case [G-In]**.** Ditto.

Case [G-Bra]**.** By the induction hypothesis for $i \in I$

$$(L \setminus \{l_i\}) \cup L_i) \setminus (M(\Gamma + l_i) \subseteq \mathcal{A}(P_i) \, .$$

Observe that $M(\Gamma + l_i) = M(\Gamma) \cup \{l_i\}$. We compute:

$$\begin{aligned}
L \setminus M(\Gamma) &= \cap_{i \in I}(L \setminus M(\Gamma)) \\
&\subseteq \cap_{i \in I}(\{l_i\} \cup (L \setminus (M(\Gamma)))) \\
&= \cap_{i \in I}(\{l_i\} \cup (L \setminus (M(\Gamma) \cup l_i))) \\
&= \cap_{i \in I}(\{l_i\} \cup ((L \setminus \{l_i\}) \setminus (M(\Gamma + l_i)))) \\
&\subseteq \cap_{i \in I}(\{l_i\} \cup ((L \setminus \{l_i\}) \cup L_i) \setminus (M(\Gamma + l_i))) \\
&\subseteq \cap_{i \in I}(\{l_i\} \cup \mathcal{A}(P_i)) \\
&= \mathcal{A}(k?\{l_i.P_i\}_{i \in I})
\end{aligned}$$

Case [G-Sel]**.** Similar to [G-Bra].

Case [G-Par]**.** By typing, we find Δ_1, Δ_2 and L_1, L_2 s.t. $\Gamma; L_i \vdash P_i \triangleright \Delta_i$. By the induction hypothesis we then find that $L_i \setminus M(\Gamma) \subseteq \mathcal{A}(P_i)$. We now compute:

$$\begin{aligned}
L \setminus M(\Gamma) &= (L_1 \cup L_2) \setminus M(\Gamma) \\
&= L_1 \setminus M(\Gamma) \cup L_2 \setminus M(\Gamma) \\
&\subseteq \mathcal{A}(P_1) \cup \mathcal{A}(P_2) \\
&= \mathcal{A}(P_1 \mid P_2)
\end{aligned}$$

Case [G-VarP]**.** We have $\Gamma, X : (L', \Delta); L \vdash X[\tilde{k}] \triangleright \Delta$. By typing $L \subseteq L'$; by definition $L' \subseteq M(\Gamma)$. But then $L \setminus M(\Gamma) = \emptyset$.

Case [G-RecP]**.** We have $\Gamma; L \vdash (\mathrm{rec}^e\, X(i).P; Q) \triangleright \Delta$. By typing we have $\Gamma \vdash Q \triangleright \Delta$ and by definition $\mathcal{A}((\mathrm{rec}^e\, X(i).P; Q)) = \mathcal{A}(Q) \supseteq L \setminus M(\Gamma)$, the latter by the induction hypothesis.

Case [G-Var]**.** We have $\Gamma, X : (A, I, \Delta); L \vdash X \triangleright \Delta$. By definition, we find $A \subseteq M(\Gamma)$, so by typing $L \subseteq I \subseteq A = M(\Gamma)$. But then $L \setminus M(\Gamma) = \emptyset$.

Case [G-Rec]**.** We have $\Gamma; L \vdash \operatorname{rec} X.P \triangleright \Delta$. By typing we must have $\Gamma, X : (\emptyset, I, \Delta); I \vdash P \triangleright \Delta$. We compute.

$$\begin{aligned} L \setminus M(\Gamma) &\subseteq I \setminus (M(\Gamma) \cup \emptyset) \\ &= I \setminus M(\Gamma, X : (\emptyset, I, \Delta)) \\ &\subseteq \mathcal{A}(P) && \text{by IH} \end{aligned}$$

Case [G-Cond]**.** By typing and the induction hypothesis we have $L \setminus \Gamma(M) \subseteq \mathcal{A}(P)$ and $L \setminus \Gamma(M) \subseteq \mathcal{A}(Q)$. But then also $L \setminus \Gamma(M) \subseteq \mathcal{A}(P) \cap \mathcal{A}(Q) = \mathcal{A}(\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q)$. □

Proposition 6.16. *Suppose $\cdot\,; L \vdash P \triangleright \Delta$ with P lock-free, and let $s = (P_i, \alpha_i)_i$ be a maximal transition sequence of P. Then $L \subseteq \mathsf{sel}(\alpha)$.*

Proof. Observe that necessarily s lock-free. We compute:

$$\begin{aligned} L &\subseteq \mathcal{A}(P) && \text{By Lemma 6.15} \\ &\subseteq \mathsf{sel}(\alpha) && \text{By Proposition 6.14} \end{aligned}$$

□

Example 6.17. We saw in Example 5.13 that the process D of Example 2.1 is typable $\cdot; \{\mathsf{SI}\} \vdash D \triangleright \cdots$. By Proposition 6.16 above, noting that D is clearly lock-free, every maximal transition sequence of D must eventually select SI.

Theorem 6.18. *Suppose $\cdot\,; L \vdash P \triangleright \Delta$ with P lock-free. Then P is live for $\cdot\,, \Delta$.*

Proof. Consider a maximal transition sequence (P_i, α_i) of P. By Definition 4.4 we must find a live type transition sequence (Δ_i, δ_i) of Δ with $((P_i, \Delta_i), (\alpha_i, \delta_i))$ a typed transition sequence of $\cdot \vdash P \triangleright \Delta$.

By induction and Theorem 5.12 there exists a sequence $(\Delta_i, L_i, \delta_i)_i$ with $\cdot\, L_i \vdash P_i \triangleright \Delta_i$ and $\Delta_i \xrightarrow{\delta_i} \Delta_{i+1}$ and $\delta_i \simeq \alpha_i$, and moreover $L_{i+1} = L_i \setminus \mathsf{res}(\delta_i) \cup \mathsf{req}(\delta_i)$. Suppose $l \in \mathsf{req}(\delta_n)$. Then $l \in L_{n+1}$. Clearly P_{n+1} also lock-free, so by Proposition 6.14, $l \in \mathsf{sel}(\mathsf{shift}^n(\alpha))$. That means there exists $j > n$ with $l \in \mathsf{sel}(\alpha_j)$. But $\alpha_j \simeq \delta_j$ so $l \in \mathsf{res}(\delta_j)$. □

Example 6.19. We saw in Example 5.13 that $P(D)$ is typable as $\cdot; \emptyset \vdash P(D) \triangleright k : T_P, o^+ : T_D, o^- : \overline{T_D}$. Noting $P(D)$ lock-free, by the above Theorem it is live, and so will uphold the liveness guarantee in T_P: if CO is selected, then eventually also SI is selected. Or in the intuition of the example: If the buyer performs "Checkout", he is guaranteed to subsequently receive an invoice.

7 Conclusion and Future Work

We introduced a conservative generalization of binary session types to *session types with responses*, which allows to specify response liveness properties. We showed that session types with responses are strictly more expressive (wrt. the

classes of behaviours they can express) than standard binary session types. We provided a typing system for a process calculus similar to a non-trivial subset of collaborative BPMN processes with possibly infinite loops and bounded iteration and proved that lock-free, well typed processes are live.

We have identified several interesting directions for future work: Firstly, the present techniques could be lifted to multi-party session types, which guarantees lock-freedom. Secondly, investigate more general liveness properties. Thirdly, channel passing is presently omitted for simplicity of presentation and not needed for our expressiveness result (Theorem 3.10). Introducing it, raises the question of wether one can delegate the responsibility for doing responses or not? If *not*, then channel passing does not affect the liveness properties of a lock-free process, and so is not really interesting for the present paper. If one *could*, it must be ensured that responses are not forever delegated without ever being fulfilled, which is an interesting challenge for future work. Finally, and more speculatively, we plan to investigate relations to fair subtyping [22] and Live Sequence Charts [6].

A Subject reduction proof for the standard session typing system

Lemma A.1 (Process variable substitution). *Suppose that* $\Theta, X : \Delta' \vdash_{\mathsf{std}} P \triangleright \Delta$. *Suppose moreover that* $\Theta \vdash_{\mathsf{std}} Q \triangleright \Delta'$ *with* X *is not free in* Q. *Then also* $\Theta \vdash_{\mathsf{std}} P\{Q/X\} \triangleright \Delta$

Proof. By induction on the typing derivation.

Case [G-Inact]**.** We have $\Theta, X : \Delta' \vdash_{\mathsf{std}} \mathbf{0} \triangleright \Delta$. By typing Δ completed, so by [G-Inact], we have $\Theta \vdash_{\mathsf{std}} \mathbf{0}\{Q/X\} \triangleright \Delta$.

Case [G-Out]**.** Immediate from the induction hypothesis.

Case [G-In]**.** Immediate from the induction hypothesis.

Case [G-Bra]**.** We have

$$\frac{\forall i \in I : \quad \Theta, X : \Delta' \vdash_{\mathsf{std}} P_i \triangleright \Delta, k : T_i}{\Theta, X : \Delta' \vdash_{\mathsf{std}} \ k?\{l_i.P_i\}_{i\in I} \triangleright \Delta, k : \&\{l_i[L_i].T_i\}_{i\in I}}$$

By the induction hypothesis and [G-Bra] we have

$$\frac{\Theta \vdash_{\mathsf{std}} P_i\{Q/X\} \triangleright \Delta, k : T_i}{\Theta \vdash_{\mathsf{std}} \ k?\{l_i.P_i\{Q/X\}\}_{i\in I} \triangleright \Delta, k : \&\{l_i[L_i].T_i\}_{i\in I}} \quad .$$

Case [G-Sel]**.** Similar to [G-Bra].

Case [G-Par]**.** We have $\Theta, X : \Delta' \vdash_{\mathsf{std}} P_1 \mid P_2 \triangleright \Delta$. By typing we find some $\Delta_1, \Delta_2 = \Delta$ such that $\Theta, X : \Delta \vdash_{\mathsf{std}} P_i \triangleright \Delta_i$. By induction we find $\Theta \vdash_{\mathsf{std}} P_i\{Q/X\} \triangleright \Delta_i$, which in turn yields $\Theta \vdash_{\mathsf{std}} (P_1 \mid P_2)\{Q/X\} \triangleright \Delta_1, \Delta_2$.

Case [G-VarP]**.** Suppose first $X \neq Y$; then we have

$$\frac{\mathsf{dom}(\Delta) = \tilde{k}}{\Theta, Y : \Delta, X : \Delta' \vdash_{\mathsf{std}} Y[\tilde{k}] \triangleright \Delta} \ ,$$

so by [G-VarP] also

$$\Theta, Y : \Delta \vdash_{\mathsf{std}} Y[\tilde{k}]\{Q/X\} \triangleright \Delta \ .$$

If on the other hand $X = Y$ we have by typing

$$\frac{\mathsf{dom}(\Delta) = \tilde{k}}{\Theta, X : \Delta' \vdash_{\mathsf{std}} X[\tilde{k}] \triangleright \Delta} \ ;$$

and it must be the case that $\Delta = \Delta'$. We have by assumption $\Theta \vdash_{\mathsf{std}} Q \triangleright \Delta'$, that is $\Theta \vdash_{\mathsf{std}} X[\tilde{k}]\{Q/X\} \triangleright \Delta$.

Case [G-RecP]**.** We have $\Theta, X : \Delta' \vdash_{\mathsf{std}} (\mathrm{rec}^e\, Y(i).P; R) \triangleright \Delta$. By typing we have $\Theta, X : \Delta', Y : \Delta \vdash_{\mathsf{std}} P \triangleright \Delta$ and $\Theta, X : \Delta' \vdash_{\mathsf{std}} R \triangleright \Delta$. Using $\Theta \vdash_{\mathsf{std}} Q \triangleright \Delta'$, by induction $\Theta, Y : \Delta \vdash_{\mathsf{std}} P\{Q/X\} \triangleright \Delta$ and $\Theta \vdash_{\mathsf{std}} R\{Q/X\} \triangleright \Delta$, which in turn yields $\Theta \vdash_{\mathsf{std}} (\mathrm{rec}^e\, Y(i).P; R)\{Q/X\} \triangleright \Delta$.

Case [G-Var]**.** Suppose first $X \neq Y$; then by typing we have

$$\frac{\mathsf{dom}(\Delta) = \tilde{k}}{\Theta, Y : \Delta, X : \Delta' \vdash_{\mathsf{std}} Y[\tilde{k}] \triangleright \Delta} \ ,$$

so by [G-Var] also

$$\Theta, Y : \Delta \vdash_{\mathsf{std}} Y[\tilde{k}]\{Q/X\} \triangleright \Delta \ .$$

If on the other hand $X = Y$ we have by typing

$$\frac{\mathsf{dom}(\Delta) = \tilde{k}}{\Theta, X : \Delta' \vdash_{\mathsf{std}} X[\tilde{k}] \triangleright \Delta} \ ,$$

where necessarily $\Delta = \Delta'$, so $\Theta \vdash_{\mathsf{std}} X[\tilde{k}]\{Q/X\} \triangleright \Delta'$.

Case [G-Rec]**.** We have $\Theta, X : \Delta' \vdash_{\mathsf{std}} \mathrm{rec}\, Y.P \triangleright \Delta$. We find by typing $\Theta, X : \Delta', Y : \Delta \vdash_{\mathsf{std}} P \triangleright \Delta$, hence by the induction hypothesis $\Theta, Y : \Delta' \vdash_{\mathsf{std}} P\{Q/X\} \triangleright \Delta$, and so by [G-Rec] $\Theta \vdash_{\mathsf{std}} (\mathrm{rec}\, Y.P)\{Q/X\} \triangleright \Delta$.

Case [G-Cond]**.** Immediate from the induction hypothesis. □

Lemma A.2. *If $\Theta \vdash_{\mathsf{std}} P \triangleright \Delta$ then also $\Theta \vdash_{\mathsf{std}} P\{v/x\} \triangleright \Delta$.*

Proof. Straightforward induction. □

Lemma A.3. *If $\Theta \vdash_{\mathsf{std}} P \triangleright \Delta$ and $\Delta(k) \neq \mathtt{end}$ then $k \in \mathsf{fn}(P)$.*

Proof. Straightforward induction. □

Lemma A.4. *Suppose $\Theta \vdash_{\mathsf{std}} P \triangleright \Delta, k : T$ with $\Delta, k : T$ balanced, $T \neq \mathtt{end}$, and $\overline{k} \notin \mathsf{fn}(P)$. Then $\overline{k} \notin \mathsf{dom}(\Delta)$.*

Proof. Supposed for a contradiction $\overline{k} \in \mathsf{dom}(\Delta)$. Because $\Delta, k : !.T$ balanced, $\Delta(\overline{k}) \neq \mathtt{end}$. By Lemma A.3 we thus have $\overline{k} \in \mathsf{fn}(P)$; contradiction. □

Lemma A.5. *Suppose that $\Theta \vdash_{\mathsf{std}} P \triangleright \Delta$ with $P \xrightarrow{\lambda} Q$. Then there exists a type transition $\Delta \xrightarrow{\delta} \Delta'$ with $\delta \simeq \lambda$, such that $\Theta \vdash_{\mathsf{std}} Q \triangleright \Delta'$. Moreover, if Δ balanced, then also Δ' balanced.*

Proof. By induction on the derivation of the transition.

Case [C-Out]**.** We have $k!\langle e\rangle.P \xrightarrow{k!v} P$ with $\overline{k} \notin P$ and $\Theta \vdash_{\mathsf{std}} k!\langle e\rangle.P \triangleright \Delta, k : !.T$. By typing $\Theta \vdash_{\mathsf{std}} P \triangleright \Delta, k : T$. By [F-Lift] we have $k : !.T \xrightarrow{k:!} k : T$. By [F-Par] $\Delta, k : !.T \xrightarrow{k:!} \Delta, k : T$. Observing that $k : ! \simeq k!v$ we have found the requisite type transition.

Now suppose $\Delta, k : !.T$ balanced; we must show $\Delta, k : T$ balanced. It is sufficient to show $\overline{k} \notin \mathsf{dom}(\Delta)$. But this follows from Lemma A.4.

Case [C-In]**.** We have $k?(x).P \xrightarrow{k?v} P\{v/x\}$ with $\overline{k} \notin \mathsf{fn}(P)$ and $\Theta \vdash_{\mathsf{std}} k?(x).P \triangleright \Delta, k : ?.T$. By typing $\Theta \vdash_{\mathsf{std}} P \triangleright \Delta, k : T$. By [F-Lift] and [F-Par], $\Delta, k : ?.T \xrightarrow{k:?} \Delta, k : T$. By Lemma A.2 we have $\Theta \vdash_{\mathsf{std}} P\{v/x\} \triangleright \Delta, k : T$. Observing $k : ? \simeq k?v$ we have found the requisite transition and typing. Preservation of balance follows from Lemma A.4.

Case [C-Bra]**.** We have $k?\{l_i.P_i\} \xrightarrow{k\&l_i} P_i$ and $\Theta \vdash_{\mathsf{std}} k?\{l_i.P_i\}_{i\in I} \triangleright \Delta, k : \&\{l_i[L_i].T_i\}_{i\in I}$. By typing we have $\Theta + l_i \vdash_{\mathsf{std}} P_i \triangleright \Delta, k : T_i$. By [F-Lift] and [F-Par] we have $\Delta, k : \&\{l_i[L_i].T_i\}_{i\in I} \xrightarrow{k:\&l_i[L_i]} \Delta, k : T_i$. Observing that $k : \oplus l_i[L_i] \simeq k\&l_i$, we have found the requisite type transition. Preservation of balance follows from Lemma A.4.

Case [C-Sel]**.** We have $k!l.P \xrightarrow{k\oplus l_i} P$ and $\Theta \vdash_{\mathsf{std}} k!l_i.P \triangleright \Delta, k : \oplus\{l_i[L_i].T_i\}_{i\in I}$. By typing $\Theta + l_i \vdash_{\mathsf{std}} P \triangleright \Delta, k : T_i$. By [F-Lift] and [F-Par] we have

$$\Delta, \oplus\{l_i[L_i].\, T_i\}_{i\in I} \xrightarrow{k:\oplus l_i[L_i]} \Delta, T_i.$$

Observing that $k : \oplus l_i[L_i] \simeq k \oplus l_i$, we have found the requisite type transition. Preservation of balance follows from Lemma A.4.

Case [C-ParL]**.** We have $P \mid P' \xrightarrow{\lambda} Q \mid P'$ with $\overline{\mathsf{subj}(\lambda)} \notin \mathsf{fn}(P')$ and $\Theta \vdash_{\mathsf{std}} P \mid P' \triangleright \Delta$. By typing we have for some $L_1 \cup L_2 = L$ and $\Delta_1 \cup \Delta_2$ that $\Theta_1 \vdash_{\mathsf{std}} P \triangleright \Delta_1$ and $\Theta \vdash_{\mathsf{std}} P' \triangleright \Delta_2$. By the induction hypothesis, we have a transition $\Delta_1 \xrightarrow{\delta} \Delta_1'$ with $\Theta \vdash_{\mathsf{std}} Q \triangleright \Delta_1'$ and $\delta \simeq \lambda$. By Lemma 4.6 $\mathsf{dom}(\Delta_1) = \mathsf{dom}(\Delta_1')$ so Δ_1', Δ_2 is defined, and hence by [G-Par] we have $\Theta \vdash_{\mathsf{std}} Q \mid P' \triangleright \Delta_1', \Delta_2$ and thus the requisite transition.

Now suppose Δ_1, Δ_2 balanced. By Lemma 4.8 it is sufficient to prove that $\overline{\mathsf{subj}(\delta)} \notin \mathsf{dom}(\Delta_1, \Delta_2)$. If $\mathsf{subj}(\delta) = \tau$ this is trivial, so say $\mathsf{subj}(\delta) = k$ and suppose for a contradiction $\overline{k} \in \mathsf{dom}(\Delta_1, \Delta_2)$. We must have $\delta = k : \rho$ and because $\delta \simeq \lambda$ we must have $\mathsf{subj}(\lambda) = \mathsf{subj}(\delta) = k$. By Lemma 2.2 $\overline{k} \notin \mathsf{fn}(Q \mid P')$. By Lemma 4.7 we have $\Delta_1 = \Delta_1'', k : S$ with $S \neq \mathsf{end}$. Because Δ_1, Δ_2 balanced, $(\Delta_1, \Delta_2)(\overline{k}) \bowtie S$ and so $(\Delta_1, \Delta_2)(\overline{k}) \neq \mathsf{end}$.

Suppose first $\overline{k} \in \mathsf{dom}(\Delta_1)$. Then $\overline{k} \in \mathsf{dom}(\Delta_1'')$, so also $\Delta''(\overline{k}) \neq \mathsf{end}$, and it follows that $\Delta_1'(\overline{k}) = \Delta_1''(\overline{k}) \neq \mathsf{end}$. By Lemma 5.9 $\overline{k} \in \mathsf{fn}(Q)$, contradicting $\overline{k} \notin \mathsf{fn}(Q \mid P')$.

Suppose instead $\overline{k} \in \mathsf{dom}(\Delta_2)$. Then immediately by Lemma 5.9 $\overline{k} \in \mathsf{fn}(P')$, contradicting $\overline{k} \notin \mathsf{fn}(Q \mid P')$.

Case [Com-1]**.** We have

$$\frac{P_1 \xrightarrow{\overline{k}!v} P_1' \qquad P_2 \xrightarrow{k?v} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'}$$

and

$$\frac{\Theta \vdash_{\mathsf{std}} P_1 \triangleright \Delta_1 \qquad \Theta \vdash_{\mathsf{std}} P_2 \triangleright \Delta_2}{\Theta \vdash_{\mathsf{std}} P_1 \mid P_2 \triangleright \Delta_1, \Delta_2}$$

By induction we find $\Delta_i \xrightarrow{\delta_i} \Delta_i'$ s.t. $\Theta \vdash_{\mathsf{std}} P_I \triangleright \Delta_i'$ with $\delta_1 \simeq \overline{k}!v$ and $\delta_2 \simeq k?v$. It follows that $\delta_1 = k : !$ and $\delta_2 = k : ?$. By [F-Com1] we thus have $\Delta_1, \Delta_2 \xrightarrow{\tau} \Delta_1', \Delta_2'$. Noting that $\tau \simeq \tau$, we have the required type transition. Since $\mathsf{subj}(\tau) = \tau$ and so $\overline{\mathsf{subj}(\tau)} \notin \mathsf{dom}(\Delta_1, \Delta_2)$, it follows from Lemma 4.8 that Δ_1', Δ_2' is balanced when Δ_1, Δ_2 is.

Case [C-Com2]**.** We have

$$\frac{P_1 \xrightarrow{\overline{k}\oplus l} P_1' \qquad P_2 \xrightarrow{k\&l} P_2'}{P_1 \mid P_2 \xrightarrow{\tau : l} P_1' \mid P_2'}$$

and

$$\frac{\Theta \vdash_{\mathsf{std}} P_1 \triangleright \Delta_1 \qquad \Theta \vdash_{\mathsf{std}} P_2 \triangleright \Delta_2}{\Theta \vdash_{\mathsf{std}} P_1 \mid P_2 \triangleright \Delta_1, \Delta_2}$$

By induction we find $\Delta_i \xrightarrow{\delta_i} \Delta_i'$ s.t. $\Theta \vdash_{\mathsf{std}} P_i \triangleright \Delta_i'$ with $\delta_1 \simeq \overline{k} \oplus l$ and $\delta_2 \simeq k \& l$. It follows that $\delta_1 = \overline{k} : \oplus l[L_1']$, and so we find a transition $\Delta_1, \Delta_2 \xrightarrow{\tau : l, L_1' \cup L_2'} \Delta_1', \Delta_2'$ by [F-Com2]. By [G-Par] we find $\Theta \vdash_{\mathsf{std}} P_1' \mid P_2' \triangleright \Delta_1', \Delta_2'$. Noting that $\tau : l, L_1' \cup L_2' \simeq \tau : l$ we have the required type transition. Since $\mathsf{subj}(\tau : l, L_1' \cup L_2') = \tau$ and so $\overline{\mathsf{subj}(\tau : l, L_1' \cup L_2')} \notin \mathsf{dom}(\Delta_1, \Delta_2)$, it follows from Lemma 4.8 that Δ_1', Δ_2' is balanced when Δ_1, Δ_2 is.

Case [C-Rec]**.** We have

$$\frac{P\{\mathrm{rec}\, X.P/X\} \xrightarrow{\lambda} Q}{\mathrm{rec}\, X.P \xrightarrow{\lambda} Q}$$

and

$$\frac{\Theta, X : \Delta \vdash_{\mathsf{std}} P \triangleright \Delta}{\Theta \vdash_{\mathsf{std}} \mathrm{rec}\, X.P \triangleright \Delta}$$

It then follows by Lemma A.1 that

$$\Theta \vdash_{\mathsf{std}} P\{\mathrm{rec}\, X.P/X\} \triangleright \Delta,$$

and so by induction we find the required balance-preserving type transition.

Case [C-Prec0]**.** We have

$$\frac{e \Downarrow 0 \qquad Q \xrightarrow{\lambda} R}{(\mathrm{rec}^e\, X(i).P; Q) \xrightarrow{\lambda} R}$$

and

$$\frac{\Theta, X : \Delta \vdash_{\mathsf{std}} P \triangleright \Delta \qquad \Theta \vdash_{\mathsf{std}} R \triangleright \Delta}{\Theta \vdash_{\mathsf{std}} (\mathrm{rec}^{e}\, X(i).P; Q) \triangleright \Delta}\ ,$$

and so by the induction hypothesis we find the required balance-preserving type transition.

Case [C-PRECN]**.** We have

$$\frac{e \Downarrow n+1 \qquad P\{n/i\}\{(\mathrm{rec}^{n}\, X(i).P; Q)/X\} \xrightarrow{\lambda} R}{(\mathrm{rec}^{e}\, X(i).P; Q) \xrightarrow{\lambda} R}$$

and again

$$\frac{\Theta, X : \Delta \vdash_{\mathsf{std}} P \triangleright \Delta \qquad \Theta \vdash_{\mathsf{std}} Q \triangleright \Delta}{\Theta \vdash_{\mathsf{std}} (\mathrm{rec}^{e}\, X(i).P; Q) \triangleright \Delta}\ .$$

By Lemma A.2 we have $\Theta, X : (L', \Delta) \vdash_{\mathsf{std}} P\{n/i\} \triangleright \Delta$. By Lemma A.1 we have

$$\Theta \vdash_{\mathsf{std}} P\{n/i\}\{(\mathrm{rec}^{n}\, X(i).P; Q)/X\} \triangleright \Delta,$$

and the requisite balance-preserving type transition follows by the induction hypothesis.

Case [C-CONDT] and [C-CONDF]**.** We have

$$\frac{e \Downarrow \mathsf{true} \qquad P \xrightarrow{\lambda} P'}{\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\lambda} P'}$$

and

$$\frac{\Theta \vdash P \triangleright \Delta \qquad \Theta \vdash Q \triangleright \Delta}{\Theta \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta}\ ,$$

and the requisite balance-preserving type transition follows from the induction hypothesis. □

References

1. Bettini, L., M. Coppo, L. D'Antoni, M. D. Luca, M. Dezani-Ciancaglini and N. Yoshida, *Global progress in dynamically interleaved multiparty sessions*, in: *CONCUR*, 2008, pp. 418–433.
2. Brill, M., W. Damm, J. Klose, B. Westphal and H. Wittke, *Live sequence charts: An introduction to lines, arrows, and strange boxes in the context of formal verification*, in: *SoftSpez Final Report*, LNCS **3147** (2004), pp. 374–399.
3. Carbone, M. and S. Debois, *A graphical approach to progress for structured communication in web services*, in: *ICE*, 2010, pp. 13–27.
4. Cheung, S.-C., D. Giannakopoulou and J. Kramer, *Verification of liveness properties using compositional reachability analysis*, in: *ESEC/SIGSOFT FSE*, Lecture Notes in Computer Science **1301** (1997), pp. 227–243.
5. Coppo, M., M. Dezani-Ciancaglini, L. Padovani and N. Yoshida, *Inference of global progress properties for dynamically interleaved multiparty sessions*, in: *COORDINATION*, 2013, pp. 45–59.
6. Damm, W. and D. Harel, *Lscs: Breathing life into message sequence charts*, Formal Methods in System Design **19** (2001), pp. 45–80.
7. Dardha, O., E. Giachino and D. Sangiorgi, *Session types revisited*, in: *PPDP*, 2012, pp. 139–150.
8. Deniélou, P.-M. and N. Yoshida, *Multiparty session types meet communicating automata*, in: *ESOP*, 2012, pp. 194–213.
9. Deniélou, P.-M. and N. Yoshida, *Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types*, in: *ICALP*, 2013, pp. 174–186.
10. Dezani-Ciancaglini, M., U. de'Liguoro and N. Yoshida, *On progress for structured communications*, in: *TGC*, 2007, pp. 257–275.
11. Dezani-Ciancaglini, M., S. Drossopoulou, D. Mostrous and N. Yoshida, *Objects and session types*, Inf. Comput. **207** (2009), pp. 595–641.
12. Fossati, L., K. Honda and N. Yoshida, *Intensional and extensional characterisation of global progress in the π-calculus*, in: *CONCUR*, 2012, pp. 287–301.
13. Gay, S. J. and M. Hole, *Subtyping for session types in the pi calculus*, Acta Inf. **42** (2005), pp. 191–225.
14. Honda, K., A. Mukhamedov, G. Brown, T.-C. Chen and N. Yoshida, *Scribbling interactions with a formal foundation*, in: *ICDCIT*, 2011, pp. 55–75.
15. Honda, K., V. Vasconcelos and M. Kubo, *Language primitives and type discipline for structured communication-based programming*, in: *ESOP*, 1998, pp. 122–138.
16. Honda, K., N. Yoshida and M. Carbone, *Multiparty asynchronous session types*, in: *POPL*, 2008, pp. 273–284.
17. Hu, R., N. Yoshida and K. Honda, *Session-based distributed programming in Java*, in: J. Vitek, editor, *ECOOP '08*, LNCS **5142**, 2008 pp. 516–541.
18. Kobayashi, N., *A type system for lock-free processes*, I&C **177** (2002), pp. 122 – 159.
19. Kobayashi, N. and C.-H. L. Ong, *A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes*, in: *LICS* (2009), pp. 179–188.
20. Mostrous, D. and V. T. Vasconcelos, *Session typing for a featherweight Erlang*, in: *COORDINATION*, 2011, pp. 95–109.
21. Object Management Group BPMN Technical Committee, *Business Process Model and Notation, v2.0*, Webpage (2011), `http://www.omg.org/spec/BPMN/2.0/PDF`.
22. Padovani, L., *Fair subtyping for open session types*, in: *ICALP*, 2013, pp. 373–384.

23. Roa, J., O. Chiotti and P. D. Villarreal, *A verification method for collaborative business processes*, in: *Business Process Management Workshops (1)*, Lecture Notes in Business Information Processing **99** (2011), pp. 293–305.
24. Vasconcelos, V., *Fundamentals of session types*, I&C **217** (2012), pp. 52–70.
25. Vieira, H. T. and V. T. Vasconcelos, *Typing progress in communication-centred systems*, in: *COORDINATION*, 2013, pp. 236–250.
26. Yoshida, N. and V. T. Vasconcelos, *Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication*, ENTCS **171** (2007), pp. 73–93.