# Session Typed *Parameterised* Communication Patterns

Andi Bejleri

Department of Computing, Imperial College London

**Abstract.** Communication patterns describe simple and elegant structured interactions in communication based applications. They are used in many parallel computing architectures of parallel algorithms, data exchange protocols and web-services. Communication patterns help programmers to design more efficient, structured, modular and understandable architectures, but they do not provide any automatic code validation.
We study this problem using *global session types*, a type theory that describes structured interactions from a global point of view. We then augment the syntax of global types with parameters that abstract the number of participants and an iterative construct that builds instances of parameterised communication patterns. Our formal system allows programmers to represent parameterised communication patterns by global types and then validate the code by type-checking.

## 1 Introduction

Communication patterns describe simple and elegant structured interactions in communication based applications. They are used in many parallel computing architectures of parallel algorithms [11], data exchange protocols [2] and web-services [1]. Communication patterns, as design patterns, help programmers to design more modular and more understandable system architectures. In parallel algorithms, communication patterns define the assignment of processes to regions of the problem domain. Hence, the choice of the pattern affects the performance of the algorithm. Common communication patterns are Ring, Tree, Mesh and Hypercube.

This paper studies communication patterns in the context of global session types, a type theory that addresses at static time the problem of type-safe, deadlock-free interactions among processes. Global session types [13] describe the interaction structure of several processes, that is defined by the "sending-receiving" actions in the presence of conditionals and recursion, from a global point of view. Processes are then validated by type-checking, through the projection of global types onto each participant.

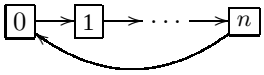
Both communication patterns and global types describe structured interactions, and the latter provide not only a blue-print of the system architecture but also a type system that guarantees type-safe and deadlock-free interactions in the system's implementation. At this point, a question arises how to specify all instances of a communication pattern in a single global type, so that programmers can benefit from the type theory.

For example, the Ring pattern of two participants is specified in the global type syntax as $\mathtt{0} \to \mathtt{1}{:}\langle U\rangle.\mathtt{1} \to \mathtt{0}{:}\langle U\rangle.\mathsf{end}$, where the causality $\mathtt{p} \to \mathtt{p}'{:}\langle U\rangle$ represents a message exchange, of type $U$, between a sender $\mathtt{p}$ and a recipient $\mathtt{p}'$ and $\mathsf{end}$ signifies the end of a conversation, while for three participants as $\mathtt{0} \to \mathtt{1}{:}\langle U\rangle.\mathtt{1} \to \mathtt{2}{:}\langle U\rangle.\mathtt{2} \to \mathtt{0}{:}\langle U\rangle.\mathsf{end}$. Building global types of other instances given the number of participants is rather easy. Unfortunately, in parallel algorithms and other communication-based applications the number of participants is known only at run-time; e.g. in parallel algorithms, the number of processes assigned to compute the answer of a problem instance

is in proportion to its size. We would like a type theory of global types that answers this research question:

> *How can programmers specify a single global type that captures all the instances of a communication pattern that has a different number of participants?*

The solution to this problem is to parameterise participants, to iterate over parameterised causalities that abstract the repetitive behavior of a pattern, and to compose sequentially global types. We use the $\mathbf{R}$ operator from Gödel's theory $T$ of primitive recursive functionals to formalise the three idioms. In the Ring pattern,




the causality that abstracts the communications from `0` to `n` is $\texttt{i} \rightarrow \texttt{i+1}{:}\langle U \rangle$, where $0 \leq i \leq n-1$ and $n \geq 1$. Given the number of participants, the $\mathbf{R}$ operator will iterate over the parameterised causality, and then the global type created will be composed with the causality $\texttt{n} \rightarrow \texttt{0} : \langle U \rangle$ to complete an instance of the Ring.

Another problem is the design of processes that implement the behavior of each participant. The behavior of `1` in the Ring pattern of two participants is: receive from `0` and then send to `0`; of three participants: receive from `0` and then send to `2`. Our formal model needs to address also a second research question:

> *How can programmers specify a single program that captures all the instances of a communication pattern that has a different number of participants?*

There are three kinds of participants in the ring pattern: the first one is `0` which sends to the participant on his left (`1`) and then receives from the last participant (`n`), the second one is `i` for $1 \leq i \leq n-1$ that receives from the participants on his right (`i−1`) and then sends to the one on his left (`i+1`) and finally, the last participant `n`, which receives from the participant on his right (`n−1`) and then sends to the first participant (`0`). Before giving a summary of the solution to this problem, we provide the definition of *role*:

**Definition 1 (Role)** Role is a *parameterised process* at implementation-time, which represents a design to create run-time processes that are ready to participate in a session. This design describes the behavior that all run-time processes of that role will share.

The solution proposed in this paper introduces the syntax of roles that includes the $\mathbf{R}$ operator, to parameterise participants, to iterate over roles and to compose in parallel roles. In the Ring, for each kind of participant corresponds a role. The $\mathbf{R}$ operator will iterate over the role of `i`, returning on each iteration processes that share the same behavior, and then will compose them in parallel with the processes of roles of `0` and `n`.

In this paper, we extend the syntax of global types introduced by Honda et al. [13], to describe parameterised communication patterns and propose a programming paradigm of roles. Programmers first define the global type of the intended pattern and then define each role of it. Roles are validated through projection of the global type onto the principals by type-checking. At run-time, roles are instantiated into processes that will generate correct sessions. More specifically, contributions of this work include:

– Notion of role, and a different definition of principal and global type are introduced to address the above questions, in coherence with our design choices (Sections 2.1 and 2.2).
– Examples that illustrate how the formal model of this work can represent various communication patterns (Section 2.3).
– Typing mechanisms for kinding of global types, projection of global types onto parameterised principals, and ordering and $\mathbf{R}$-elimination of actions, and also a static type-system that validates the specification against the description through role types (Section 3).
– Examples from parallel algorithms and key distribution protocols illustrating the practical utility of our system (Section 4).
– Auxiliary definitions of the formal system and proof of type-preservation (Appendix).

## 2 Formalising Parameterised Communication Patterns

Our system is modelled after that of Bettini et al. [4], where channels are omitted from the syntax of roles and global types, serving a simpler type system than the one introduced by Honda et al. [13]. The model is based on small-step operational semantics, which allows to use standard proofs techniques.

### 2.1 Roles

**Syntax** Figure 1 provides the syntax of our calculus. The metavariable $a$ ranges over shared channels; $\mathtt{p}$ ranges of principals; $s$ ranges over session channels; $y$ ranges over channel variables; $x$ ranges over variables; $l$ ranges over labels; $I$ ranges over index sets; $\mathbb{X}$ ranges over process names; $w$ ranges over session- and shared-channels; $i$ ranges over index variables. A program in our calculus is a function from naturals (the number of participants) to roles composed in parallel. Roles in our calculus are second-class constructs; they can not be computed by functions. Each role defines a scope that includes the subsequent behaviors. The role with the overbar $\bar{u}$ prefix represents the behavior of the first principal in the list (possibly parameterised) of all principals present in a session $\mathtt{p}_0, \mathtt{p}_1, p$ and the process of that role initiates a session with the acceptor processes of principals $\mathtt{p}_1$ and $p$. In sending and receiving constructs, the principal denotes the other end-point of the communication; the same notation is used in selection-branching of a label where the former selects one of the labels enumerated in $I$ and sends it to the later. $\nu a.R$ restricts $a$ to $R$. Parallel composition and conditional are standard. Recursion and process call define infinite behavior.

To this core, we add the $\mathbf{R}$ operator from System $T$ to parameterise principals, to iterate and compose in parallel roles. The recursive operator can be used also inside the definition of a role to iterate over a particular end-point behavior and to sequentially compose behaviors. Iteration takes place when a natural is applied to a primitive recursion term.

The message queue is part of the runtime syntax of the calculus. Identifiers $u$ can be variables or shared names. A list of principals (see Section 2.2) can be constant or parameterised using the $\mathbf{R}$ operator. Expressions include parametric mathematical expressions (see Section 2.2), values and operations such as $e = e'$, $e$ and $e'$ and not $e$.

$\mathrm{E} ::= \lambda n.\mathrm{E} \mid \mathrm{E}\ \mathtt{t} \mid R$ General expressions

| | | | |
|---|---|---|---|
| $R ::=$ | Roles | $\mid (\nu w)R$ | Hiding |
| $\mid \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R$ | Multicast request | $\mid \mathbb{X}\langle e\rangle$ | Process call |
| $\mid u[\mathtt{p}](y).R$ | Accept | $\mid \mathbf{0}$ | Inaction |
| $\mid c!\langle \mathtt{p}, e\rangle; R$ | Value sending | $\mid R \mid \mathrm{S}$ | Parallel |
| $\mid c?\langle \mathtt{p}, x\rangle; R$ | Value reception | $\mid \mathsf{if}\ e\ \mathsf{then}\ S\ \mathsf{else}\ R$ | Conditional |
| $\mid c \oplus \langle \mathtt{p}, l\rangle; R$ | Selection | $\mid \mathbf{R}\ S\ \lambda i.\lambda X.R$ | Primitive recursion |
| $\mid c\&\langle \mathtt{p}, \{l_i : R_i\}_{i\in I}\rangle$ | Branching | $\mid X$ | Process variable |
| $\mid \mathsf{def}\ \mathbb{X}(x) = S\ \mathsf{in}\ R$ | Recursive definition | $\mid R\ \mathtt{t}$ | Application |
| | | $\mid s{:}h$ | Queues |

| | | | |
|---|---|---|---|
| $u ::= x \mid a$ | Identifiers | $c ::= y \mid s[\hat{\mathtt{p}}]$ | Channels |
| $p ::= \mathtt{p}_1..\mathtt{p}_\mathrm{n} \mid \mathbf{R}\ p\ \lambda i.\lambda X.p'\ \mathtt{t} \mid X$ | List of prin. | $h ::= \epsilon \mid m \cdot h$ | Queues |
| $e ::= \mathtt{t} \mid v \mid e\ \mathsf{op}\ e'$ | Expressions | $m ::= (\hat{\mathtt{q}},\hat{\mathtt{p}},v) \mid (\hat{\mathtt{q}},\hat{\mathtt{p}},l)$ | Msgs. in transit |
| $v ::= a \mid s[\hat{\mathtt{p}}] \mid \mathrm{n} \mid \mathsf{true} \mid \mathsf{false}$ | Values | $\hat{\mathtt{p}}, \hat{\mathtt{q}} ::= \hat{\mathtt{p}}[\mathrm{n}] \mid \mathcal{N}$ | Value principals |

**Fig. 1.** Syntax for roles and run-time processes

Values are defined over shared names, session channels (higher-order communication is obtained through the value send-receive constructs), naturals and boolean values. Channels denote channel variables or session channels. Messages in queues are defined as triples, sender, receiver and data (value or label). Messages are run-time entities, therefore they are defined over value principals. Value principals include participants (`Bob`, `Alice`, ...) or indexed principals over naturals (`W`[3], `W`[2][4], ...).

**Operational Semantics** Figure 2 gives the operational semantics via the reduction relation $\longrightarrow$ where the state of the machine is defined only by terms of the calculus. The interesting features of the rules are how they invoke a program, start a session, instantiate roles, iterate over end-point behavior and exchange messages.

The rule [App] invokes a program by replacing the parameter $n$ with the argument $\mathrm{n}$. Also, it instantiates roles which are parameterised only by $n$. Rule [Zero] returns the behavior $S$ and defines the last iteration of the $\mathbf{R}$ operator. Rule [Succ] replaces each occurrence of the index $i$ in $R$ with a predecessor of $\mathrm{n}{+}1$ and replaces $X$ with instances of $R$ returned by the other iterations. When $R$ denotes roles, [Succ] instantiates them in each iteration and composes them in parallel. Otherwise $R$ denotes an end-point behavior that [Succ] iterates when the session has been established.

A session is established among processes via shared channels, that denote public points of communication. At this point, every role has been instantiated into processes and the computation follows over value principals. The rule [Link] invokes a session between $\mathrm{n}$ peers by generating a session channel and substitutes it in the processes scope. The identity of each principal within a session is represented by the label attached to the session channel. The [Send] and [Label] rules insert a message in the queue of the session. The receiving rule [Recv] removes a value message of the same sender, as the one specified in the receiving construct, from the queue, and substitutes it in the process. The [Branch] rule removes a label message of the same sender, as the ones specified in the branching construct, from the queue. The result of the rule is the process following

$$(\lambda n.E)\ \mathrm{n} \longrightarrow E\{\mathrm{n}/n\} \qquad \text{[App]}$$

$$\mathbf{R}\ S\ \lambda i.\lambda X.R\ 0 \longrightarrow S \qquad \text{[Zero]}$$

$$\mathbf{R}\ S\ \lambda i.\lambda X.R\ \mathrm{n}+1 \longrightarrow R\{\mathrm{n}/i\}\{\mathbf{R}\ S\ \lambda i.\lambda X.R\ \mathrm{n}/X\} \qquad \text{[Succ]}$$

$$\begin{array}{c}\bar{a}[\hat{\mathtt{p}}_0..\hat{\mathtt{p}}_\mathrm{n}](y_0).R_0 \mid a[\hat{\mathtt{p}}_1](y_1).R_1 \mid ...a[\hat{\mathtt{p}}_\mathrm{n}](y_\mathrm{n}).R_\mathrm{n} \\ \longrightarrow (\nu s)(R_0\{s[\hat{\mathtt{p}}_0]/y_0\} \mid ... \mid R_\mathrm{n}\{s[\hat{\mathtt{p}}_\mathrm{n}]/y_\mathrm{n}\} \mid s:\emptyset)\end{array} \qquad \text{[Link]}$$

$$s[\hat{\mathtt{p}}]!\langle\hat{\mathtt{q}}, v\rangle; R \mid s:h \longrightarrow R \mid s:h\cdot(\hat{\mathtt{p}},\hat{\mathtt{q}},v) \qquad \text{[Send]}$$

$$s[\hat{\mathtt{p}}] \oplus \langle\hat{\mathtt{q}}, l\rangle; R \mid s:h \longrightarrow R \mid s:h\cdot(\hat{\mathtt{p}},\hat{\mathtt{q}},l) \qquad \text{[Label]}$$

$$s[\hat{\mathtt{p}}]?(\hat{\mathtt{q}}, x); R \mid s:(\hat{\mathtt{q}},\hat{\mathtt{p}},v)\cdot h \longrightarrow R\{v/x\} \mid s:h \qquad \text{[Recv]}$$

$$s[\hat{\mathtt{p}}]\&(\hat{\mathtt{q}}, \{l_i:R_i\}_{i\in I}) \mid s:(\hat{\mathtt{q}},\hat{\mathtt{p}},l_{i_0})\cdot h \longrightarrow R_{i_0} \mid s:h \quad (i_0\in I) \qquad \text{[Branch]}$$

$$\mathsf{if\ true\ then}\ R\ \mathsf{else}\ S \longrightarrow R \qquad \mathsf{if\ false\ then}\ R\ \mathsf{else}\ S \longrightarrow S \qquad \text{[If-T, If-F]}$$

$$\mathsf{def}\ \mathbb{X}(x)=S\ \mathsf{in}\ (\mathbb{X}\langle v\rangle \mid R) \longrightarrow \mathsf{def}\ \mathbb{X}(x)=S\ \mathsf{in}\ (S\{v/x\} \mid R) \qquad \text{[Def]}$$

$$R \longrightarrow R' \ \Rightarrow\ (\nu r)R \longrightarrow (\nu r)R' \qquad R \longrightarrow R' \ \Rightarrow\ R \mid S \longrightarrow R' \mid S \qquad \text{[Scop,Par]}$$

$$R \longrightarrow R' \ \Rightarrow\ \mathsf{def}\ D\ \mathsf{in}\ R \longrightarrow \mathsf{def}\ D\ \mathsf{in}\ R' \qquad \text{[Defin]}$$

$$R \equiv R' \text{ and } R' \longrightarrow S' \text{ and } S \equiv S' \ \Rightarrow\ R \longrightarrow S \qquad \text{[Str]}$$

$$\mathtt{i}_i \longrightarrow \mathtt{i}_i' \ \Rightarrow\ \mathcal{N}[\mathtt{i}_0]...[\mathtt{i}_i]...[\mathtt{i}_n] \longrightarrow \mathcal{N}[\mathtt{i}_0]...[\mathtt{i}_i']...[\mathtt{i}_n] \qquad \text{[Princ]}$$

$$\mathtt{p}_i \longrightarrow \mathtt{p}_i' \ \Rightarrow\ \mathcal{R}[\mathtt{p}_0,...,\mathtt{p}_i,...,\mathtt{p}_n] \longrightarrow \mathcal{R}[\mathtt{p}_0,...,\mathtt{p}_i',...,\mathtt{p}_n] \qquad \text{[ContextP]}$$

$$p \longrightarrow p' \ \Rightarrow\ \mathcal{R}[...,p] \longrightarrow \mathcal{R}[...,p'] \qquad \text{[Request]}$$

$$e_i \longrightarrow e_i' \ \Rightarrow\ \mathcal{E}[e_0,...,e_i,...,e_n] \longrightarrow \mathcal{E}[e_0,...,e_i',...,e_n] \qquad \text{[ContextE]}$$

**Fig. 2.** Reduction rules

the label. [If-T] and [If-F] action the evaluation of e; if e evaluates to true then rule [If-T] is applied otherwise rule [If-F]. [Def] invokes the behaviour ($P$) identified by $\mathbb{X}$by binding the parameter $x$to argument $v$. [Scop] actions the reduction of the process inside the scope of the $\nu$ operator. [Par] actions the reduction of a process parallely composed with other processes. [Str] states that the reduction relation is defined on structural congruent terms, given in Figure 3. The [Context] rule define the order of execution of expressions within a process. Definition of the context is given in Figure 10. The list of parameterised principals reduces following the rules shown in Figure 4.

### 2.2 Global Types

Figure 5 gives the syntax of global types where the metavariable $T$ ranges over role types. (see Section 3.3). A message of type $U$ is exchanged between two principals. Branching is defined over labels which identify the paths of a conversation; i.e. participant $\mathtt{p}$ internally chooses one of the labels $l_i$ enumerated by $I$ and then sends it to participant $\mathtt{p}'$ and the conversation follows $G_i$. Infinite behavior is represented by recursively defined global types $\mu\mathbf{t}.G$. $\mathsf{end}$ signifies the end of a conversation.

The $\mathbf{R}$ operator is added to the syntax of global types to describe communication patterns of an arbitrary number of principals. The parameters that abstract the number

$$P \mid \mathbf{0} \equiv P \quad P \mid Q \equiv Q \mid P \quad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$$

$$(\nu r)\ P \mid Q \equiv (\nu r)\ (P \mid Q) \quad \text{if } r \notin \text{fn}(Q)$$

$$(\nu r r')\ P \equiv (\nu r' r)\ P \quad (\nu r)\ \mathbf{0} \equiv \mathbf{0} \quad \mathsf{def}\ D\ \mathsf{in}\ \mathbf{0} \equiv \mathbf{0}$$

$$\mathsf{def}\ D\ \mathsf{in}\ (\nu r)\ P \equiv (\nu r)\ \mathsf{def}\ D\ \mathsf{in}\ P \quad \text{if } r \notin \text{fn}(D)$$

$$(\mathsf{def}\ D\ \mathsf{in}\ P) \mid Q \equiv \mathsf{def}\ D\ \mathsf{in}\ (P \mid Q) \quad \text{if } \text{dpv}(D) \cap \text{fpv}(Q) = \emptyset$$

$$\mathsf{def}\ D\ \mathsf{in}\ (\mathsf{def}\ D'\ \mathsf{in}\ P) \equiv \mathsf{def}\ D\ \mathsf{and}\ D'\ \mathsf{in}\ P \quad \text{if } \text{dpv}(D) \cap \text{dpv}(D') = \emptyset$$

$$s : (\mathtt{q}, \mathtt{p}, z) \cdot (\mathtt{q}', \mathtt{p}', z') \cdot h \equiv s : (\mathtt{q}', \mathtt{p}', z') \cdot (\mathtt{q}, \mathtt{p}, z) \cdot h \quad \text{if } \mathtt{p} \neq \mathtt{p}' \text{ or } \mathtt{q} \neq \mathtt{q}'$$

**Fig. 3.** Structural equivalence

$$\mathbf{R}\ p\ \lambda i.\lambda X.p'\ 0 \longrightarrow p \qquad \text{[Zero]}$$

$$\mathbf{R}\ p\ \lambda i.\lambda X.p'\ \mathrm{n}+1 \longrightarrow p'\{\mathrm{n}/i\}\{\mathbf{R}\ p\ \lambda i.\lambda X.p'\ \mathrm{n}/X\} \qquad \text{[Succ]}$$

**Fig. 4.** Reduction rules for parameterised list of principals

of participants are bound by the binders in the lambda expressions of roles, as both global type and roles are part of the program definition. The $\mathbf{R}$ operator is defined over the tail recursion case of recursion and index variable $i$. Thus the operator preserves the existing declarative nature of global types, helping to ensure that the system developed in our model can be extended with additional features. Throughout the paper, we will refer to primitive recursive global types as *product global types*, as they abstract all instances of the parameterised global type. The infinite set of instances generated from the $\mathbf{R}$ operator can be understood through the two reduction rules:

$$\mathbf{R}\ G\ \lambda i.\lambda \mathbf{x}.G'\ 0 \longrightarrow G$$
$$\mathbf{R}\ G\ \lambda i.\lambda \mathbf{x}.G'\ \mathrm{n}+1 \longrightarrow G'\{\mathrm{n}/i\}\{(\mathbf{R}\ G\ \lambda i.\lambda \mathbf{x}.G'\ \mathrm{n})/\mathbf{x}\}$$

For each natural, we obtain a global type by applying the two rules. In each iteration, the index variable in $G'$ is substituted by a predecessor of $n{+}1$ and $\mathbf{x}$ is replaced by instances of the parameterised causalities present in $G'$, except 0 when $\mathbf{x}$ is replaced by instances of $G$.

Principals $\mathtt{p}, \mathtt{p}', \mathtt{q}, ..$ include primitive participants `Alice`, `Bob`, ... and indexed principals defined over one or multiple index expressions $\mathtt{W}[i]$, $\mathtt{W}[i{+}1][j{+}1]$,.... Index expressions are represented by parametric linear functions, where $\mathrm{n}$ ranges over naturals, $i$ ranges over index variables and $\mathtt{t}$ ranges over parametric expressions. Parametric expressions range over variables, naturals, arithmetical operations ($\mathtt{t}{+}\mathrm{n}$, $\mathtt{t}{-}\mathrm{n}$, $\mathtt{t}{*}\mathrm{n}$) and exponentiation of base natural. The design of index expressions as parametric linear functions comes from our observation that the information flow follows a line in the patterns/virtual-topologies we have studied so far [2, 11, 14]. For simplicity and without

$G$ ::= Global types

| | | |
|---|---|---|
| $\mathtt{p} \to \mathtt{p}' : \langle U \rangle . G$ | Message | $\mathbf{R}\ G\ \lambda i.\lambda \mathbf{x}.G'$ Primitive recursion |
| $\mathtt{p} \to \mathtt{p}' : \{l_i : G_i\}_{i \in I}$ | Branching | $\mathbf{x}$ Primitive rec. type var. |
| $\mu \mathbf{t}.G$ | Recursion | $G\ \mathtt{t}$ Application |
| $\mathbf{t}$ | Rec. type var. | $\mathsf{end}$ End |

| | | | |
|---|---|---|---|
| $\mathtt{p} ::= \mathtt{p}[\mathtt{i}] \mid \mathcal{N}$ | Principals | $\mathcal{N} ::= \mathtt{Alice} \mid \mathtt{Worker} \mid \ldots$ | Participants |
| $\mathtt{i} ::= \mathtt{t} \mid i \mid \mathrm{n} * \mathtt{i} \mid \mathtt{t} \pm \mathtt{i}$ | Index expr. | $U ::= S \mid T$ | Message type |
| $\mathtt{t} ::= \mathrm{n} \mid n \mid \mathtt{t}\ \mathrm{op}\ \mathrm{n} \mid \mathrm{n}^{\mathtt{t}}$ | Par. expr. | $S ::= \mathsf{bool} \mid \mathsf{nat} \mid .. \mid \langle G \rangle$ | Value type |

**Fig. 5.** Global types

reducing the practical expressiveness of our system, we have designed index expression to have at most one parameter $n$. A type $U$ ranges over primitive (bool, nat) and global types ($\langle G \rangle$), and role types ($T$); global types $\langle G \rangle$ type shared channels and role types ($T$) type session channels.

### 2.3 Ring and Tree Communication Patterns

We illustrate how the formal model of this work can represent various communication patterns such as Ring and Tree.

**Ring pattern - figure 6(a)** The Ring pattern consists of $n{+}1$ workers (named by $\mathtt{W}$) where each has exactly two neighbours: the worker $\mathtt{W}[j]$ communicates with the workers $\mathtt{W}[j{-}1]$ and $\mathtt{W}[j{+}1]$ ($1{\leq}j{\leq}\mathrm{n}{-}1$), with the exception of $\mathtt{W}[0]$ and $\mathtt{W}[\mathrm{n}]$ who communicate via a direct link. Due to the enumeration of workers in a non-modular arithmetic, the Ring has three distinct roles: *Starter*, represented by $\mathtt{W}[0]$, *Middle*, represented by $\mathtt{W}[j]$, and *Last*, represented by $\mathtt{W}[\mathrm{n}]$. The global type specifies that the first message is sent by $\mathtt{W}[0]$ to $\mathtt{W}[1]$ for $j{=}n{-}1$, and the last one is sent by $\mathtt{W}[\mathrm{n}]$ back to $\mathtt{W}[0]$ for $n{=}0$. To ensure the presence of all three roles in a session, we set the number of participants to $n{\geq}2$. Below, we provide the main program and roles of the Ring:

$$
\begin{aligned}
&\mathtt{def}\ \ \mathtt{W} = \mathbf{R}\ \mathtt{W}[n]\ \lambda i.\lambda X.(\mathtt{W}[i+2], X)\ n-2 \\
&\textit{Starter} \triangleq \bar{a}[\mathtt{W}[0], \mathtt{W}[1], \mathtt{W}](y).y!\langle \mathtt{W}[1], v\rangle; y?(\mathtt{W}[n], z); R \\
&\textit{Middle(i)} \triangleq a[\mathtt{W}[i+1]](y).y?(\mathtt{W}[i], z); y!\langle \mathtt{W}[i+2], z\rangle; R' \\
&\textit{Last} \triangleq a[\mathtt{W}[n]](y).y?(\mathtt{W}[n-1], z); y!\langle \mathtt{W}[0], z\rangle; S \\
&\mathrm{Ring} \triangleq \lambda n.(\mathbf{R}\ \textit{Starter} \mid \textit{Last}\ \lambda i.\lambda X.(\textit{Middle(i)} \mid X) \quad n-1)
\end{aligned}
$$

where *W* denotes the parameterised list of principals $\mathtt{W}[2]$, ..., $\mathtt{W}[n]$, represented mathematically through the $\mathbf{R}$ operator, and *Starter* and *Last* are parameterised by $n$ and *Middle* by $i$. *Middle* is composed in parallel with the process variable $X$ that is used as a placeholder of processes generated in each iteration; in the last iteration for $n{=}0$, $X$ will be replaced with processes of *Starter* and *Last*. The reduction steps of the Ring for $n{=}2$ are given in Appendix A.1.

**Tree pattern - figure 6(b)** The Tree pattern consists of $2^{n+1}{-}1$ workers organized in a binary tree. The global type specifies a message exchange between a parent node and its children, numbered in the Ahnentafel system. A tree has three kinds of nodes: root, internal and leaf. The principal running on the root sends a message to its children; the

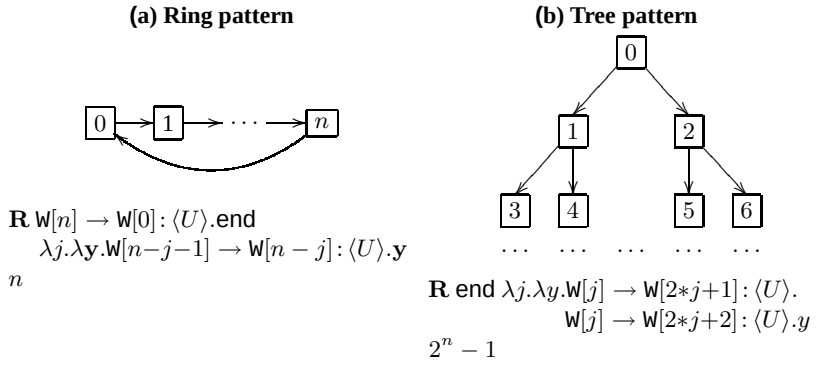



**Fig. 6.** Diagram and global type of the Ring and Tree communication patterns

ones on internal nodes send a message to their children and receive a message from their parents; the ones on leaf nodes receive a message from their parents. The three kind of nodes define three distinct roles of the Tree. An internal or leaf node is enumerated by an even or odd number, and thus the mathematical expressions that identify the parent and children of each of these nodes are different. For this reason, even and odd nodes define two distinct roles in the same kind of node, internal/leaf. Thus, we have distinguished five roles in the Tree: *Root* represented by $\mathtt{W}[0]$, *OddInt* and *EvenInt* by $\mathtt{W}[2{*}i{+}1]$ and $\mathtt{W}[2{*}i{+}2]$ ($0{\leq}i{\leq}2^{n-1}{-}2$), and, *OddLeaf* and *EvenLeaf* by $\mathtt{W}[2{*}i{+}1]$ and $\mathtt{W}[2{*}i{+}2]$ ($2^{n-1}{-}1{\leq}i{\leq}2^{n}{-}2$). To ensure the presence of all five roles in a session, we set $n \geq 2$. Below, we provide the Tree's roles and the main program:

$$\mathtt{def}\ \mathtt{W} = \mathbf{R}\ \mathtt{W}[2^{n+1}-2]\ \lambda i.\lambda X.(\mathtt{W}[i+2], X)\ 2^{n+1}-4$$

$$\textit{Root} \triangleq \bar{a}[\mathtt{W}[0], \mathtt{W}[1], \mathtt{W}](y).y!\langle \mathtt{W}[1], f(1)\rangle; y!\langle \mathtt{W}[2], f(2)\rangle; R$$

$$\textit{OddInt(i)} \triangleq a[\mathtt{W}[2{*}i{+}1]](y).y!\langle \mathtt{W}[4{*}i{+}3], f(4{*}i{+}3)\rangle; y!\langle \mathtt{W}[4{*}i{+}4], f(4{*}i{+}4)\rangle; y?(\mathtt{W}[i], z); R'$$

$$\textit{EvenInt(i)} \triangleq a[\mathtt{W}[2{*}i{+}2]](y).y!\langle \mathtt{W}[4{*}i{+}5], v_{2*i+2}\rangle; y!\langle \mathtt{W}[4{*}i{+}6], v'_{2*i+2}\rangle; y?(\mathtt{W}[i], z); Q'$$

$$\textit{OddLeaf(i)} \triangleq a[\mathtt{W}[2^{n}{-}1{+}2{*}i]](y).y?(\mathtt{W}[2^{n-1}{-}1{+}i], z); S$$

$$\textit{EvenLeaf(i)} \triangleq a[\mathtt{W}[2*(2^{n-1}+i-1)+2]](y).y?(\mathtt{W}[2^{n-1}+i-1], z); R$$

$$Tree \triangleq \lambda n.(\mathbf{R}\ (\mathbf{R}\ \textit{Root}\ \lambda i.\lambda X.(\textit{OddLeaf(i)} \mid \textit{EvenLeaf(i)} \mid X)\quad 2^{n-1})$$
$$\lambda i.\lambda X.(\textit{OddInt(i)} \mid \textit{EvenInt(i)} \mid X)2^{n-1}{-}1)$$

where $f$ is a function from naturals to $U$. It is interesting to note that index calculation in the principals of the global type is less complex than in the ones of the roles. This is a direct advantage of the global representation of interactions. The problem of index calculation in parallel computing architectures of parallel algorithms has been recognized also by the MPI community [11] as a source of program errors.

## 3 Typing

This section introduces the typing mechanisms for kinding, projection, ordering and $\mathbf{R}$-elimination, and also rules of typing.

$$\frac{C \vdash \mathtt{p}, \mathtt{p}' \quad \Theta; C \vdash G' \blacktriangleright \mathsf{Type} \quad \Theta; C \vdash U \blacktriangleright \mathsf{Type} \quad \mathsf{fprtv}(U) = \emptyset}{\Theta; C \vdash \mathtt{p} \to \mathtt{p}' : \langle U \rangle . G' \blacktriangleright \mathsf{Type}} \lfloor \text{KIO} \rfloor \qquad \frac{C \vdash \mathtt{p}, \mathtt{p}' \quad \forall i \in I,\ \Theta; C \vdash G_i \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \mathtt{p} \to \mathtt{p}' : \{l_i : G_i\}_{i \in I} \blacktriangleright \mathsf{Type}} \lfloor \text{KBRA} \rfloor$$

$$\frac{\Theta; C \vdash G \blacktriangleright \mathsf{Type} \quad \Theta, \mathbf{x} : \mathsf{Type}; C, i : \mathtt{I} \vdash G' \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \mathbf{R}\ G\ \lambda i . \lambda \mathbf{x} . G' \blacktriangleright \Pi i : \mathtt{I} . \mathsf{Type}} \lfloor \text{KPR} \rfloor \qquad \frac{C \vdash \mathtt{t} \quad \Theta; C \vdash G \blacktriangleright \Pi i : \{i | i \in \mathsf{nat}, 0 \leq i \leq \mathtt{t} - 1\} . \mathsf{Type}}{\Theta; C \vdash G\ \mathtt{t} \blacktriangleright \mathsf{Type}} \lfloor \text{KAPP} \rfloor$$

$$\frac{}{\Theta, \mathbf{t} : \mathsf{Type}; C \vdash \mathbf{t} \blacktriangleright \mathsf{Type}} \lfloor \text{KRVAR} \rfloor \qquad \frac{}{\Theta, \mathbf{x} : \mathsf{Type}; C \vdash \mathbf{x} \blacktriangleright \mathsf{Type}} \lfloor \text{KPRVAR} \rfloor$$

$$\frac{\Theta, \mathbf{t} : \mathsf{Type}; C \vdash G \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \mu \mathbf{t} . G \blacktriangleright \mathsf{Type}} \lfloor \text{KREC} \rfloor \qquad \frac{}{\Theta; C \vdash \mathsf{end} \blacktriangleright \mathsf{Type}} \lfloor \text{KEND} \rfloor$$

**Fig. 7.** Kinding rules for global types

### 3.1 Kinding of Global Types

Primitive recursion forms a new kind of global type–product kind. Kinding rules, shown in Figure 7, ensure that in a global type the indexed principals are well-formed and parametric expressions are applied only to product global types. They are of the form $\Theta; C \vdash G \blacktriangleright \kappa$. read, "In the variable typing $\Theta$ and parameter-index typing $C$, global type $G$ has kind $\kappa$". Variable and parameter-index context typing $\Theta$ and $C$, and kind $\kappa$ are defined as:

$$\Theta ::= \emptyset \mid \mathbf{x} : \mathsf{Type}, \Theta \mid \mathbf{t} : \mathsf{Type}, \Theta \quad C ::= \emptyset \mid n : \mathtt{T}, C \mid i : \mathtt{I}, C \quad \kappa ::= \mathsf{Type} \mid \Pi i : \mathtt{I} . \mathsf{Type}$$
$$\mathtt{T} ::= \{n \mid n \in \mathsf{nat}, n \geq \mathtt{n}\} \quad \mathtt{I} ::= \{i \mid i \in \mathsf{nat}, 0 \leq i \leq \mathtt{t}\}$$

The $\lfloor \text{KIO} \rfloor$ rule ensures that the principals of a causality are well-formed, the message type is kinded and that there are no free primitive recursive type variables, and that the inductive part is Type kinded. A well-formed principal is either a participant or an indexed principal where the set of values of each index expression is defined over naturals. We have defined a set of rules that ensure when subtraction can be used safely in expressions (see Appendix B.3). The $\lfloor \text{KBRA} \rfloor$ rule checks that the principals of label causality are well-formed and that the inductive parts are Type kinded. The $\lfloor \text{KPR} \rfloor$ rule ensures that the inductive parts are Type kinded, returning the kind of a product global type. The $\lfloor \text{KAPP} \rfloor$ rule checks that the argument is applied to a product global type and that it is a successor of the biggest index value, returning a Type kind. Other rules ensure that the inductive global types are Type kinded and look up for type variables in the context $\Theta$.

### 3.2 Role Types

The role types, given in Figure 8, are used to type-check the roles of parameterised communication patterns.

The role types prefixed by the output, input, selection and branching types capture the same capabilities. Infinite behavior is captured by the recursion type. The primitive recursion with application are used to capture repetitive end-point behaviors.

| | | | | |
|---|---|---|---|---|
| $T ::= !\langle \mathtt{p}, U\rangle; T$ | Output | $\mid \mathbf{R}\ T\ \lambda i.\lambda \mathbf{x}.T'$ | Primitive Recursion |
| $\mid\ ?\langle \mathtt{p}, U\rangle; T$ | Input | $\mid\ T\ \mathtt{t}$ | Application |
| $\mid\ \oplus\langle \mathtt{p}, \{l_i : T_i\}_{i\in I}\rangle$ | Selection | $\mid\ \mathbf{x}$ | Primitive Recursion Variable |
| $\mid\ \&\langle \mathtt{p}, \{l_i : T_i\}_{i\in I}\rangle$ | Branching | $\mid\ \mathbf{t}$ | Recursion Variable |
| $\mid\ \mu \mathbf{t}.T$ | Recursion | $\mid\ \mathsf{end}$ | Null |

**Fig. 8.** Role types

### 3.3 Projection, Ordering and R-elimination

**Projection** A global type's projection onto the principals of roles produces types that capture the behavior of roles.

**Definition 3.1** *Given global type $G$, principal $\mathtt{q}$, and the context $C$ of parameter variables present in $G$ and $\mathtt{q}$, and index variables present in $\mathtt{q}$, if $\emptyset; C \vdash G \blacktriangleright \kappa$ and $C \vdash \mathtt{q}$ then the projection of $G$ onto $\mathtt{q}$, denoted $G\upharpoonright \mathtt{q}$, is defined inductively on $G$:*

$$\mathtt{p} \to \mathtt{p}' : \langle U\rangle.G\upharpoonright \mathtt{q} = \begin{cases} !\langle \mathtt{p}'\{\mathtt{p}=\mathtt{q}\}, U\rangle(\mathtt{p}); ?\langle \mathtt{p}\{\mathtt{p}'=\mathtt{q}\}, U\rangle(\mathtt{p}'); (G\upharpoonright \mathtt{q}) & \text{if } C\vdash \mathtt{p}{=}\mathtt{q} \text{ and } C\vdash \mathtt{p}'{=}\mathtt{q}, \\ !\langle \mathtt{p}'\{\mathtt{p}=\mathtt{q}\}, U\rangle(\mathtt{p}); (G\upharpoonright \mathtt{q}) & \text{if } C\vdash \mathtt{p}{=}\mathtt{q}, \\ ?\langle \mathtt{p}\{\mathtt{p}'=\mathtt{q}\}, U\rangle(\mathtt{p}'); (G\upharpoonright \mathtt{q}) & \text{if } C\vdash \mathtt{p}'{=}\mathtt{q}, \\ G\upharpoonright \mathtt{q} & \text{otherwise} \end{cases}$$

$$\mathtt{p} \to \mathtt{p}' : \{l_i : G_i\}_{i\in I}\upharpoonright \mathtt{q} = \begin{cases} \oplus\langle \mathtt{p}'\{\mathtt{p}=\mathtt{q}\}, \{l_i : \&\langle \mathtt{p}\{\mathtt{p}'=\mathtt{q}\}, \{l_i : G_i\upharpoonright \mathtt{q}\}_{i\in I}\rangle(\mathtt{p}')\}_{i\in I}\rangle(\mathtt{p}) & \text{if } C\vdash \mathtt{p}{=}\mathtt{q} \text{ and } C\vdash \mathtt{p}'{=}\mathtt{q}, \\ \oplus\langle \mathtt{p}'\{\mathtt{p}=\mathtt{q}\}, \{l_i : G_i\upharpoonright \mathtt{q}\}_{i\in I}\rangle & \text{if } C\vdash \mathtt{p}{=}\mathtt{q}, \\ \&\langle \mathtt{p}\{\mathtt{p}'=\mathtt{q}\}, \{l_i : G_i\upharpoonright \mathtt{q}\}_{i\in I}\rangle & \text{if } C\vdash \mathtt{p}'{=}\mathtt{q}, \\ \sqcup_{i\in I} G_i\upharpoonright \mathtt{q} & \text{if } C\nvdash \mathtt{p}{=}\mathtt{q}, C\nvdash \mathtt{p}'{=}\mathtt{q} \\ & \forall i,j{\in}I.G_i\upharpoonright \mathtt{q} \bowtie G_j\upharpoonright \mathtt{q} \end{cases}$$

$$\mu\mathbf{t}.G\upharpoonright \mathtt{q} = \mu\mathbf{t}.(G\upharpoonright \mathtt{q}) \quad \mathbf{t}\upharpoonright \mathtt{q} = \mathbf{t} \quad \mathsf{end}\upharpoonright \mathtt{q} = \mathsf{end}$$

$$\mathbf{R}\ G\ \lambda i.\lambda \mathbf{x}.G'\upharpoonright \mathtt{q} = \mathbf{R}\ (G\upharpoonright \mathtt{q})\ \lambda i.\lambda \mathbf{x}.(G'\upharpoonright \mathtt{q}) \quad \mathbf{x}\upharpoonright \mathtt{q} = \mathbf{x} \quad G\ \mathtt{t}\upharpoonright \mathtt{q} = (G\upharpoonright \mathtt{q})\ \mathtt{t}$$

Projection is intuitive and holds some of the technical challenges of this system, which we discuss in the following paragraphs. In the role types returned, the principal in brackets, attached to an action, denotes the principal that performs that action, and is used to sort actions and eliminate the $\mathbf{R}$ operator from role types as we shall see later. The equality between a global type principal $\mathtt{p}$ and role principal $\mathtt{q}$ is defined as a relation $\vdash \mathtt{p}{=}\mathtt{q}$ over the context $C$ (see Appendix B.4), which ensures that the set of values of $\mathtt{p}$ is a subset of the set of values of $\mathtt{q}$. The intuition underlying this design originates from the knowledge that an action performed by every process of the same role is captured by the same causality in the global type.

In product global types, for different values of the index variable, an indexed principal can be present in both sides of a parameterised causality. This occurrence is covered by the first case of projection for message exchange and branching.

The index variables of principals in global types are different from the ones in roles, as they are bound by different binders. For this reason, we need to translate the role

types being expressed from global type indexes to role ones. The $\mathtt{p}'\{\mathtt{p} = \mathtt{q}\}$ operation (see Appendix B.5) substitutes the index variables in $\mathtt{p}'$ with expressions in terms of indexes of $\mathtt{q}$, obtained by the relation $\mathtt{p} = \mathtt{q}$ where $\mathtt{p}$ and $\mathtt{p}'$ have the same index variables.

In branching, in the case when $\mathtt{q}$ is not equal neither to $\mathtt{p}$ nor to $\mathtt{p}'$, all inductive projections of $\mathtt{q}$ should return an identical role type up to mergeability $\bowtie$. The notion of mergeability is introduced in [9] as an equivalence relation over role types. Intuitively, two different $\&$ role types are mergeable if the labels, they are denoted by, are different; e.g. the projection of global type:

$$\mathtt{W}[1] \to \mathtt{W}[2] : \begin{cases} true : \mathtt{W}[2] \to \mathtt{W}[3] : \{true : G, \\ false : \mathtt{W}[2] \to \mathtt{W}[3] : \{false : G' \end{cases}$$

onto $\mathtt{W}[3]$ returns $\&\langle \mathtt{W}[2], \{true{:}G \upharpoonright \mathtt{W}[3], false{:}G \upharpoonright \mathtt{W}[3]\}\rangle$ where $G \upharpoonright \mathtt{W}[3] \neq G' \upharpoonright \mathtt{W}[3]$.

**Definition 3.2 (Mergebility)** *If* $\forall i \in (I \cap J).T_i \bowtie T'_i$ *and* $\forall i \in I \setminus J.\forall j \in J \setminus I.l_i \neq l_j$ *then* $\&\langle \mathtt{p}, \{l_i : T_i\}_{i\in I}\rangle \bowtie \&\langle \mathtt{p}, \{l_j : T'_j\}_{j\in J}\rangle$.
$\&\langle \mathtt{p}, \{l_i : T_i\}\rangle \sqcup \&\langle \mathtt{p}, \{l_j : T'_j\}_{j\in J}\rangle = \&\langle \mathtt{p}, \{l_i : T_i \sqcup T'_i\}_{i\in I\cap J} \cup \{l_i : T_i\}_{i\in I\setminus J} \cup \{l_i : T'_i\}_{i\in J\setminus I}\rangle$.

**Proposition 3.3** *The relation* $C \vdash \mathtt{p} = \mathtt{q}$ *is decidable.*

**Theorem 3.4** *The projection of a global type onto principals is decidable.*

*Proof.* Straightforward from Proposition 3.3.

**Ordering and R-elimination** Actions in the role types, returned by projection, are sorted to preserve the order of appearance in all instances of a parameterised global type. We can note from the first case of projection in the message global type, that the order of actions is not preserved; i.e., the sending action is always placed before the receiving one. However, the appearance order of actions is not broken only in the projection of a causality, but also in the sequential composition of other actions returned by projection. The reason behind this is that the order of actions depends on the order of principals performing those actions.

**Definition 3.5** *The appearance order relation between two actions (*$\mathtt{order}$*) is defined as the appearance order of the principals performing those actions:*

$\mathtt{order}(!/?^1\langle \mathtt{p}_1, U\rangle(\mathtt{p}'_1), !/?\langle \mathtt{p}_2, U'\rangle(\mathtt{p}'_2))$ *if and only if* $\mathtt{order}(\mathtt{p}'_1, \mathtt{p}'_2)$ *and* $\mathtt{order}(\oplus/\&\langle \mathtt{p}_1, \{l_i{:}T_i\}_{i\in I}\rangle(\mathtt{p}'_1), \oplus/\&\langle \mathtt{p}_2, \{l_i{:}T'_i\}_{i\in I'}\rangle(\mathtt{p}'_2))$ *if and only if* $\mathtt{order}(\mathtt{p}'_1, \mathtt{p}'_2)$.

**Definition 3.6** *The appearance order between principals is defined as a lexicographical order over the index expressions that define them:*

$\mathtt{order}(\mathcal{N}[\mathtt{i}_1]...[\mathtt{i}_i]...[\mathtt{i}_n], \mathcal{N}[\mathtt{i}'_1]...[\mathtt{i}'_i]...[\mathtt{i}'_n])$ *if and only if* $\mathtt{order}(\mathtt{i}_i, \mathtt{i}'_i)$ *for* $1 \leq i \leq n$ *and* $\forall j.1 \leq j \leq i-1.\ C \vdash \mathtt{i}_j = \mathtt{i}'_j$ *and* $C \nvdash \mathtt{i}_i = \mathtt{i}'_i$,
*where the appearance order between index expressions in their canonical form is defined as:*

$\mathtt{order}(\mathtt{t}{-}\mathrm{n}{*}i, \mathtt{t}'{-}\mathrm{n}'{*}i)$ *if and only if* $C \vdash \mathtt{t}{-}\mathrm{n}{*}i \geq \mathtt{t}'{-}\mathrm{n}'{*}i$ *and*
$\mathtt{order}(\mathtt{t}{+}\mathrm{n}{*}i, \mathtt{t}'{+}\mathrm{n}'{*}i)$ *if and only if* $C \vdash \mathtt{t}{+}\mathrm{n}{*}i \leq \mathtt{t}'{+}\mathrm{n}'{*}i$.

[1] $!/?$ denotes either ! or ?.

The order of index expression is defined on the basis that the value of $i$ decreases in each iteration of the $\mathbf{R}$ global type, resulting in the increase of values for expressions $\mathtt{t}-\mathrm{n}*i$ and the decrease for $\mathtt{t}+\mathrm{n}*i$. Thus, in two expressions of the form $\mathtt{t}-\mathrm{n}*i$, a value will appear first in the bigger expression for bigger value of $i$ and then in the smaller one for smaller value of $i$. And, in two expressions of the form $\mathtt{t}+\mathrm{n}*i$, a value will appear first in the smaller expression for bigger value of $i$ and then in the bigger one for smaller value of $i$. No ordering can be defined for expressions of opposite monotonicity, e.g. $\mathtt{t}-\mathrm{n}*i$ and $\mathtt{t}'+\mathrm{n}'*i$, as some values will appear first in the former and second in latter, whilst some others vice versa.

The $\mathbf{R}$ operator in global types iterates over parameterised causalities and defines a repetitive behavior for non index-parameterised principals. For these principals, we keep the $\mathbf{R}$ operator and the argument applied in the role types, otherwise we eliminate it by composing the two sub-types, and then later the argument. *fivr* denotes the free index variables in the bracket-principals of role types.

**Definition 3.7** *Sorting of actions, defined over* `order` *relation, and* $\mathbf{R}$*-elimination are introduced in the function* $\xi$*, that also removes the principals in brackets, defined as:*

- $\xi(!\langle \mathtt{p}, U\rangle(\mathtt{p}'); T)) = !\langle \mathtt{p}, U\rangle; \xi(T) \quad \xi(?\langle \mathtt{p}, U\rangle(\mathtt{p}'); T) = ?\langle \mathtt{p}, U\rangle; \xi(T)$
- $\xi(\oplus\langle \mathtt{p}, \{l_i : T_i\}_{i\in I}\rangle(\mathtt{p}')) = \oplus\langle \mathtt{p}, \{l_i : \xi(T_i)\}_{i\in I}\rangle$
- $\xi(\&\langle \mathtt{p}, \{l_i : T_i\}_{i\in I}\rangle(\mathtt{p}')) = \&\langle \mathtt{p}, \{l_i : \xi(T_i)\}_{i\in I}\rangle$
- $\xi(\mathbf{R}\ T\ \lambda i.\lambda \mathbf{x}.T') = \begin{cases} \mathbf{R}\ \xi(\mathtt{sort}(T))\ \lambda i.\lambda \mathbf{x}.\xi(\mathtt{sort}(T')) & i \notin \mathit{fivr}(T') \\ \xi(\mathtt{sort}(T'))\{\xi(\mathtt{sort}(T))/\mathbf{x}\} & \mathit{otherwise} \end{cases}$
- $\xi(T\ \mathtt{t}) = \begin{cases} \xi(T)\ \mathtt{t} & \mathit{if}\ \Theta; C \vdash \xi(T) \blacktriangleright \Pi i{:}\mathtt{I}.\mathsf{Type} \\ \xi(T) & \mathit{otherwise} \end{cases}$
- $\xi(\mu \boldsymbol{t}.T) = \mu \boldsymbol{t}.\xi(T) \quad \xi(\boldsymbol{t}) = \boldsymbol{t} \quad \xi(\mathbf{x}) = \mathbf{x} \quad \xi(\mathsf{end}) = \mathsf{end}$

where $\Theta; C \vdash T \blacktriangleright \kappa$ is the kinding judgment of role types (see Appendix B.1).

### 3.4 Type System

Figure 9 describes the program typing rules. The typing judgment is of the form $\Gamma; C \vdash E \triangleright \tau$, read, "In the context $\Gamma$ and $C$ program $E$ has type $\tau$". $\Gamma$ maps shared names, process names and type variables to types, while $\tau$ represents channel and product types, defined as:

$$\tau ::= \Delta \mid \Pi n{:}\mathtt{T}.\tau \mid \Pi i{:}\mathtt{I}.\tau \quad \Delta ::= \emptyset \mid \Delta, c{:}T \quad \Gamma ::= \emptyset \mid \Gamma, u{:}S \mid \Gamma, \mathbb{X}{:}S\ T \mid \Gamma, X{:}\Delta$$

The rules of appealing interest are those for program and session initiation. Rule $\lfloor$TFUN$\rfloor$ augments the context $C$ with mapping for parameter variables and ensures that the subterm is typed. Rule $\lfloor$TAPPF$\rfloor$ checks if the argument applied to the lambda abstraction falls in the set of values $\mathtt{T}$, where $\min(\mathtt{T})$ represents the minimum value $\mathrm{n}$. For primitive recursion, we ensure that the sub-terms are well-typed in the augmented contexts $\Gamma$ and $C$. If primitive recursion, specifies a repetitive behavior of a role, then $\Delta\ 0$ and $\Delta\ i{+}1$ return the sub-role type for type-checking of the respective sub-terms. Otherwise, $\Delta\ 0$ and $\Delta\ i{+}1$ return $\Delta$.

**Definition 3.8** *Given* $\Delta$*=* $\Delta$*', c:*$T$*. We define*

$$\Delta\ i{=} \begin{cases} \Delta'\ i, c : T\ i & \text{if } \emptyset; C \vdash T \blacktriangleright \Pi j : \mathtt{I}.\mathsf{Type} \\ \Delta'\ i, c : T & \text{otherwise} \end{cases}$$

$$\frac{\Gamma; C, n : \mathtt{T} \vdash E \rhd \tau}{\Gamma; C \vdash \lambda n.E \rhd \Pi n{:}\mathtt{T}.\tau}\ \lfloor \textsc{TFun} \rfloor \qquad \frac{\Gamma; C \vdash E \rhd \Pi n{:}\mathtt{T}.\tau \quad C \vdash \mathtt{t} \geq \mathsf{min}(\mathtt{T})}{\Gamma; C \vdash E\ \mathtt{t} \rhd \tau}\ \lfloor \textsc{TAppF} \rfloor$$

$$\frac{\begin{array}{c}\Gamma; C \vdash S \rhd \Delta\ 0 \\ \Gamma, X : \Delta\ i; C, i : \mathtt{I} \vdash R \rhd \Delta\ i+1\end{array}}{\Gamma; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X.R \rhd \Pi i{:}\mathtt{I}.\Delta}\ \lfloor \textsc{TPRec} \rfloor \qquad \frac{C \vdash \mathtt{t} \quad \Gamma; C \vdash R \rhd}{\Pi i{:}\{i\,|i{\in}\mathsf{nat}, 0{\leq}i{\leq}\mathtt{t}{-}1\}.\Delta}\ \lfloor \textsc{TAppR} \rfloor$$

$$\frac{\begin{array}{c}\Gamma \vdash u : \langle G \rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type} \\ C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G) = \{\mathtt{p}_0, \mathtt{p}_1, p\} \\ \Gamma; C \vdash R \rhd \Delta, y : \xi(G \upharpoonright \mathtt{p}_0)\end{array}}{\Gamma; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \rhd \Delta}\ \lfloor \textsc{TAcc} \rfloor \qquad \frac{\begin{array}{c}\emptyset; C \vdash G \blacktriangleright \mathsf{Type} \\ \Gamma \vdash u : \langle G \rangle \quad C \vdash \mathtt{p} \\ \Gamma; C \vdash R \rhd \Delta, y : \xi(G \upharpoonright \mathtt{p})\end{array}}{\Gamma; C \vdash u[\mathtt{p}](y).R \rhd \Delta}\ \lfloor \textsc{TReq} \rfloor$$

$$\frac{\Gamma; C \vdash e \rhd S \quad \Gamma \vdash R \rhd \Delta, c : T}{\Gamma; C \vdash c!\langle \mathtt{p}, e \rangle; R \rhd \Delta, c : !\langle \mathtt{p}, S \rangle; T}\ \lfloor \textsc{TOut} \rfloor \qquad \frac{\Gamma, x : S; C \vdash R \rhd \Delta, c : T}{\Gamma; C \vdash c?\langle \mathtt{p}, x \rangle; R \rhd \Delta, c : ?\langle \mathtt{p}, S \rangle; T}\ \lfloor \textsc{TIn} \rfloor$$

$$\frac{\Gamma; C \vdash R \rhd \Delta, c : T_j \quad j \in K}{\Gamma; C \vdash c \oplus \langle \mathtt{p}, l_j \rangle; R \rhd \Delta, c : \oplus\langle \mathtt{p}, \{l_i : T_k\}_{i \in I} \rangle}\ \lfloor \textsc{TSel} \rfloor$$

$$\frac{\forall k \in K, \Gamma; C \vdash R_k \rhd \Delta, c : T_k}{\Gamma; C \vdash c\&\langle \mathtt{p}, \{l_i : R_i\}_{i \in I} \rangle \rhd \Delta, c : \&\langle \mathtt{p}, \{l_i : T_k\}_{i \in I} \rangle}\ \lfloor \textsc{TBra} \rfloor$$

$$\frac{\Gamma, a : U; C \vdash R \rhd \Delta}{\Gamma; C \vdash (\nu a)R \rhd \Delta}\ \lfloor \textsc{TNu} \rfloor \qquad \frac{\Gamma; C \vdash R \rhd \Delta \quad \Gamma \vdash S \rhd \Delta'}{\Gamma; C \vdash R \mid S \rhd \Delta, \Delta'}\ \lfloor \textsc{TPar} \rfloor$$

$$\frac{\Gamma, X : \Delta; C \vdash \mathsf{Env}}{\Gamma, X : \Delta; C \vdash X \rhd \Delta}\ \lfloor \textsc{TVar} \rfloor \qquad \frac{\Gamma; C \vdash \Delta \quad \Delta\ \mathsf{end\ only}}{\Gamma; C \vdash \mathbf{0} \rhd \Delta}\ \lfloor \textsc{TNull} \rfloor$$

**Fig. 9.** Program and role typing

**Definition 3.9** *A structural congruence of an application of an index variable to a primitive recursive end-point type is defined as:*

$\mathbf{R}\ T\ \lambda j.\lambda x.T'\ i{+}1 \equiv T'\{i/j\}\{\mathbf{R}\ T\ \lambda j.\lambda x.T'\ i/x\}$

The rule of applying a parametric expression to primitive recursion is similar to $\lfloor$TAppF$\rfloor$, but it also ensures that the argument applied is a successor of the biggest index value. Roles are type-checked by the role types, returned by projection, sorting and $\mathbf{R}$-elimination. All the conditions to invoke projection are ensured by $\lfloor$TAcc$\rfloor$ and $\lfloor$TReq$\rfloor$. Rule $\lfloor$TAcc$\rfloor$ checks also if the set of principals, present in the session, is the same as the one of global type (see Appendix B.6).

Rules $\lfloor$TOut$\rfloor$ and $\lfloor$TIn$\rfloor$ ensure that the sub-terms are typed and check if the principal in the primitives is the same as the one in the role-types. Other standard rules lookup for type variables in $\Gamma$, and type primitives such as branching, delegation, hiding, inaction and parallel composition. Typing rules for the run-time are the ones in [4].

**Properties.** In this paragraph, we state type preservation for the formal system presented in this paper, i.e. if a term is well-typed and it reduces to a new term, then the new term is also well-typed. The full proof is given in Appendix C.

**Theorem 3.10 (Type Preservation)** *If $\Gamma; C \vdash E \rhd \tau$, and $E \rightarrow^{\star} E'$, then there exists $\tau'$ where $\tau \Rightarrow \tau'$, such that $\Gamma; C \vdash E' \rhd \tau'$.*

*Proof.* By induction over the derivation of $E \rightarrow^{\star} E'$. The proof relies on standard substitution lemmas.

Although, we do not have a formal proof, we believe the system holds progress, defined as in [13]. We leave the prove of progress for future work.

### 3.5 Typing Parameterised Communication Patterns

We type the programs of the Ring and Tree, given in Section 2.3, with focus on sorting and $\mathbf{R}$-elimination in the former, and projection and principals set equality in the latter.

**Ring - figure 6(a)** The main program is typed by $\Pi n{:}\{n \mid n{\in}\mathsf{nat}, n{\geq}2\}.\Delta$, application by $\Delta$, where $\Delta\ n{-}1$ equals $\Delta$, and the primitive recursion is typed by $\Pi i{:}\{i \mid i{\in}\mathsf{nat}, 0{\leq}i{\leq}n{-}2\}.\Delta$. Projection of the global type onto the principals $\mathtt{W}[0], \mathtt{W}[i{+}1]$ and $\mathtt{W}[n]$ returns the following role-types:

$\mathbf{R}\ ?\langle \mathtt{W}[n], U\rangle(\mathtt{W}[0]); \mathsf{end}\ \lambda j.\lambda y.!\langle \mathtt{W}[1], U\rangle(\mathtt{W}[n-j-1]); y\ n,$
$\mathbf{R}\ \mathsf{end}\ \lambda j.\lambda y.!\langle \mathtt{W}[i+2], U\rangle(\mathtt{W}[n-j-1]); ?\langle \mathtt{W}[i], U\rangle(\mathtt{W}[n-j]); y\ n$ and
$\mathbf{R}\ !\langle \mathtt{W}[0], U\rangle(\mathtt{W}[n]); \mathsf{end}\ \lambda j.\lambda y.?\langle \mathtt{W}[n-1], U\rangle(\mathtt{W}[n-j-1]); y\ n.$

$\mathtt{W}[0]$ and $\mathtt{W}[n]$ contain one action in each sub-type and so, no sorting is performed on them. The $\mathbf{R}$ operator carried from the global type is eliminated, as it does not define a repetitive behavior for any principal; i.e. all participant in the lambda global type are parameterised by the index variable. Thus, the role types returned for type-checking of $\mathtt{W}[0]$ and $\mathtt{W}[n]$ are $!\langle \mathtt{W}[1], U\rangle; ?\langle \mathtt{W}[n], U\rangle; \mathsf{end}$ and $?\langle \mathtt{W}[n{-}1], U\rangle; !\langle \mathtt{W}[0], U\rangle; \mathsf{end}$, respectively. The role type of $\mathtt{W}[j{+}1]$ has a more sophisticated structure than $\mathtt{W}[0]$ and $\mathtt{W}[n]$. It has two actions in the lambda role type, that are sorted to preserve the order of appearance in all the instances of the global type. The order of actions is defined over the order of the principals that perform them. The action $?\langle \mathtt{W}[i], U\rangle$ comes before $!\langle \mathtt{W}[i{+}2], U\rangle$ ($?\langle \mathtt{W}[i], U\rangle; !\langle \mathtt{W}[i{+}2], U\rangle; \mathsf{end}$), as $\mathtt{W}[n{-}j]$ comes before $\mathtt{W}[n{-}j{-}1]$, as a participant in $\mathtt{W}[n{-}j]$ will appear first for bigger value of $j$ and then in $\mathtt{W}[n{-}j{-}1]$ for a smaller one. Type-checking of the roles with the role types is straightforward.

**Tree - figure 6(b)** For space's sake, we limit the description of typing in the roles of $\mathtt{W}[0]$ and $\mathtt{W}[2{*}i{+}1]$. Type-checking of role of $\mathtt{W}[0]$ by the type $!\langle \mathtt{W}[1], U\rangle; !\langle \mathtt{W}[2], U\rangle; \mathsf{end}$ is straightforward. The interesting part of typing is checking if the relation $C \vdash \{\mathtt{W}[j], \mathtt{W}[2{*}j{+}1], \mathtt{W}[2{*}j{+}2]\} = \{\mathtt{W}[0], \mathtt{W}[1], \mathtt{W}[2^{n+1}{-}2], \mathtt{W}[i{+}2]\}$ holds or not, in the rule $\lfloor \mathrm{TA}_{\mathrm{CC}} \rfloor$. Two principal sets are equal if each set is a subset of the other. The subset relation $C \vdash A{\subseteq}B$ holds if each element of $A$ is an element of $B$. The membership relation of a principal $\mathtt{p}$ in a set $A$ holds if the set of values of the principal is contained in the sets of values of all the members of $A$. The set of values of $\mathtt{W}[\mathtt{j}]$ ($0{\leq}j{\leq}2^n{-}1$) is contained in the sets of values of $\{\mathtt{W}[0], \mathtt{W}[1], \mathtt{W}[i{+}2]\}$ ($0{\leq}i{\leq}\ 2^{n+1}{-}5$), $\mathtt{W}[2{*}\mathtt{j}{+}1]$ in $\{\mathtt{W}[1], \mathtt{W}[\mathtt{i}{+}2]\}$ and $\mathtt{W}[2{*}\mathtt{j}{+}2]$

in $\{\mathtt{W}[\mathrm{i}{+}2], \mathtt{W}[2^{n+1}{-}2]\}$. The check of $C \vdash \{\mathtt{W}[0], \mathtt{W}[1], \mathtt{W}[i{+}2], \mathtt{W}[2^{n+1}{-}2]\} \subseteq \{\mathtt{W}[j], \mathtt{W}[2{*}j{+}1], \mathtt{W}[2{*}j{+}2]\}$ is similar but tedious so we leave it to the curious reader.

Projection of the global type onto $\mathtt{W}[2{*}i{+}1]$ checks on each causality if any principal is equal to $\mathtt{W}[2{*}i{+}1]$. For example, $C \vdash \mathtt{W}[j]{=}\mathtt{W}[2{*}i{+}1]$ holds because $j$ can be represented as an index expression in terms of $i$ and $C \vdash \max(2{*}i{+}1) \leq \max(j)$ holds, which in turn holds because $C \vdash 2{*}(2^{n-1}{-}2){+}1 \leq 2^n{-}2$ holds, where $\max(i){=}2^{n-1}{-}2$ and $\max(j){=}2^n{-}2$. Thus, the action obtained is $!\langle \mathtt{W}[2{*}j{+}1], U\rangle\{\mathtt{W}[j]{=}\mathtt{W}[2{*}i{+}1]\}$ which is translated in terms of $i$ by substituting $j$ in $2{*}j{+}1$ with $2{*}i{+}1$, returning $!\langle \mathtt{W}[4{*}i{+}3], U\rangle$. Principal $\mathtt{W}[2{*}j{+}2]$ is not equal to $\mathtt{W}[2{*}i{+}1]$ as $j$ cannot be represented as an index expressions in terms of $i$ ($j{=}i{-}1/2$ is not an index expression $\mathtt{i}$). The role type returned following projection, sorting and $\mathbf{R}$-elimination is $!\langle \mathtt{W}[4{*}i{+}3], U\rangle; !\langle \mathtt{W}[4{*}i{+}4], U\rangle; ?\langle \mathtt{W}[i], U\rangle; \mathsf{end}$.

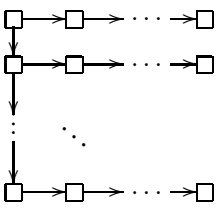
## 4 Real-World Examples

**Jacobi Solution of the Discrete Poisson Equation [11]** Poisson's equation is widely used in many areas of the natural sciences, including electrostatics and climate computations. The discrete two-dimensional Poisson equation $(\nabla^2 u)_{ij}$ for a $n \times m$ grid can be written

$$u_{ij} = \frac{1}{4}(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - dx^2 g_{i,j})$$

where $2 \leq i \leq m-1$, $2 \leq j \leq n-1$, and $dx = 1/(n+1)$. Jacobi's method converges on a solution by repeatedly replacing each element of the input grid by an adjusted average of its four neighbouring values. The grid can be divided up and the algorithm is performed on each subgrid in separate processes. Neighbouring processes must exchange their subgrid boundary values (ghost-points) as they are updated. We illustrate a two-dimensional (mesh) decomposition of the grid into $n{*}m$ processes, where $n, m{\geq}2$. The process on the $(n, m)$ subgrid, top right corner, controls the termination condition for all processes and sends the first message in the mesh. The global type for the said interactions is:

$Jacobi \triangleq \mu\mathbf{t}.\mathtt{W}[n][m] \to \mathtt{W}[n][m-1], \mathtt{W}[n-1][m] : \{\mathtt{true} : iterate, \mathtt{false} : return\}$.
The stopping condition is propagated in the processes following the pattern of the diagram below. Next to it, the global type (*iterate*) for propagating the `true` label.

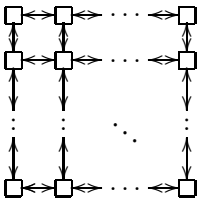


1. $\mathbf{R}$
2. $\mathbf{R}\ (\mathbf{R}\ ghst{-}pnts\ \lambda k{:}\mathtt{I}'.\lambda\mathbf{z}.\mathtt{W}[0][k{+}1] \to \mathtt{W}[0][k]{:}\{\mathtt{true}{:}\mathbf{z}\}\ m)$
3. $\quad \lambda i : \mathtt{I}.\lambda\mathbf{x}.$
4. $\quad\quad (\mathbf{R}\ \mathtt{W}[i{+}1][m] \to \mathtt{W}[i][m]{:}\{\mathtt{true}{:}\mathbf{x}\}$
5. $\quad\quad\quad \lambda j{:}\mathtt{I}'.\lambda\mathbf{y}.\mathtt{W}[i{+}1][j{+}1] \to \mathtt{W}[i{+}1][j]{:}\{\mathtt{true}{:}\mathbf{y}\}$
6. $\quad\quad\quad m)$
7. $\quad n{-}1$
8. $\lambda l{:}\mathtt{I}.\lambda\mathbf{w}.\mathtt{W}[n][l{+}1] \to \mathtt{W}[n][l]{:}\{\mathtt{true}{:}\mathbf{w}\}\ m{-}1$

Propagation of the label in the top row, is described in the causality of line 8, in all the rows, except top and bottom, line 5, in the leftmost column in line 4 and in the bottom row, line 2.

Each process maintains a copy of the boundary values of its neighbours and exchanges them on each iteration of the algorithm. The diagram below portrays how these

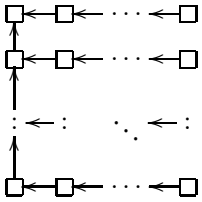
values are exchanged between the processes, followed by the global type (*ghst–pnts*).



1. $\mathbf{R}$ $(\mathbf{R}\ conv\text{–}data\ \lambda k{:}\mathtt{I}'.\lambda\mathbf{z}.\mathtt{W}[0][k+1] \leftrightarrow \mathtt{W}[0][k]{:}\langle U\rangle\ m)$
2. $\quad\lambda i{:}\mathtt{I}.\lambda\mathbf{x}.$
3. $\quad\quad(\mathbf{R}\ \mathtt{W}[i+1][0] \leftrightarrow \mathtt{W}[i][0]{:}\langle U\rangle.\mathbf{x}$
4. $\quad\quad\quad\lambda j{:}\mathtt{I}'.\lambda\mathbf{y}.\mathtt{W}[i+1][j+1] \leftrightarrow \mathtt{W}[i][j+1]{:}\langle U\rangle.$
5. $\quad\quad\quad\quad\quad\mathtt{W}[i+1][j+1] \leftrightarrow \mathtt{W}[i+1][j]{:}\langle U\rangle.\mathbf{y}$
6. $\quad\quad m)$
7. $\quad n$

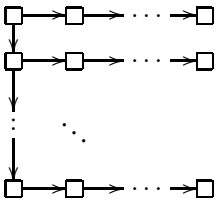
$\mathtt{p} \leftrightarrow \mathtt{p}'{:}\langle U\rangle$ is a shortcut for $\mathtt{p} \rightarrow \mathtt{p}'{:}\langle U\rangle.\mathtt{p}' \rightarrow \mathtt{p}{:}\langle U\rangle$. The exchange of ghost-points in all the rows and columns, except the leftmost column line 3 and bottom row line 1, is described in the causalities of line 4 and 5.

The convergence data are gathered at the root processes following the pattern of the diagram below and next to it, the global type (*conv–data*).



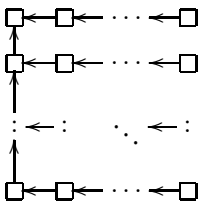
$\mathbf{R}$
$(\mathbf{R}\ \mathbf{t}\ \lambda k : \mathtt{I}'.\lambda\mathbf{z}.\mathtt{W}[n][m-k-1] \rightarrow \mathtt{W}[n][m-k]{:}\langle U'\rangle.\mathbf{z}\ m)$
$\lambda i : \mathtt{I}.\lambda\mathbf{x}.$
$\quad(\mathbf{R}\ (\mathtt{W}[n-i-1][m] \rightarrow \mathtt{W}[n-i][m]{:}\langle U'\rangle.\mathbf{x})$
$\quad\quad\lambda j : \mathtt{I}'.\lambda\mathbf{y}.$
$\quad\quad\quad\mathtt{W}[n-i-1][m-j-1] \rightarrow \mathtt{W}[n-i-1][m-j]{:}\langle U'\rangle.\mathbf{y}$
$\quad m)$
$n$

The stopping condition is propagated in the processes following the pattern of the diagram below. Next to it, the global type (*return*) for propagating the `false` label.



1. $\mathbf{R}$
2. $\mathbf{R}$ $(\mathbf{R}\ result\ \lambda k{:}\mathtt{I}'.\lambda\mathbf{z}.\mathtt{W}[0][k+1] \rightarrow \mathtt{W}[0][k]{:}\{\mathtt{false}{:}\mathbf{z}\}\ m)$
3. $\quad\lambda i : \mathtt{I}.\lambda\mathbf{x}.$
4. $\quad\quad(\mathbf{R}\ \mathtt{W}[i+1][m] \rightarrow \mathtt{W}[i][m]{:}\{\mathtt{false}{:}\mathbf{x}\}$
5. $\quad\quad\quad\lambda j{:}\mathtt{I}'.\lambda\mathbf{y}.\mathtt{W}[i+1][j+1] \rightarrow \mathtt{W}[i+1][j]{:}\{\mathtt{false}{:}\mathbf{y}\}$
6. $\quad\quad\quad m)$
7. $\quad n-1$
8. $\lambda l{:}\mathtt{I}.\lambda\mathbf{w}.\mathtt{W}[n][l+1] \rightarrow \mathtt{W}[n][l]{:}\{\mathtt{false}{:}\mathbf{w}\}\ m-1$

The final result is gathered at the root processes following the pattern of the diagram below and next to it, the global type (*result*).



$\mathbf{R}$
$(\mathbf{R}\ \mathsf{end}\ \lambda k : \mathtt{I}'.\lambda \mathbf{z}.\mathtt{W}[n][m-k-1] \to \mathtt{W}[n][m-k] : \langle Grid \rangle.\mathbf{z}\ m)$
$\lambda i : \mathtt{I}.\lambda \mathbf{x}.$
$(\mathbf{R}\ (\mathtt{W}[n-i-1][m] \to \mathtt{W}[n-i][m] : \langle Grid \rangle.\mathbf{x})$
$\lambda j : \mathtt{I}'.\lambda \mathbf{y}.$
$\mathtt{W}[n-i-1][m-j-1] \to \mathtt{W}[n-i-1][m-j] : \langle Grid \rangle.\mathbf{y}$
$m)$
$n$

The definition roles is given in Appendix D.2.

**Group Diffie-Hellman with Complete Key Authentication Protocol [2].** The Diffie-Hellman protocol is used in password-authentication key agreement and public key infrastructure. Every group $M_i$ ($0<i<n$) generates and encrypts a random exponent, that together with the data received from $M_{i-1}$ is then sent to $M_{i+1}$. Lastly, $M_n$ receives the data from $M_{n-1}$, computes the group key and broadcasts it to all other parties. The global type of the protocol is defined as:

$\mathbf{R}$ $(\mathbf{R}\ \mathsf{end}\ \lambda k : \mathtt{I}.\lambda \mathbf{z}.\mathtt{M}[n] \to \mathtt{M}[k] : \langle key \rangle.\mathbf{z}\ n)$ //Broadcasting the group key
$\lambda i : \mathtt{I}.\lambda \mathbf{x}.\mathtt{M}[n-i-1] \to \mathtt{M}[n-i] : \langle data \rangle.\mathbf{x}\ n$ //Exchanging data on a line pattern

This protocol is modelled in a system of contracts [10]. In that model, an extra private channel is used by the last group $M_n$ to send the key to every other group. The private channel is forwarded between the groups through delegation. A condition is added to the protocol description to check whether the key is sent to every group or not. In our model, we do not need an extra private channel to send the key, as communications between parties of a session are always defined over private channels. Also, we do not need to add a condition that checks whether the key is sent to every group as this is granted by the semantics of the $\mathbf{R}$ operator.

## 5 Related Work

The idea of parameterised session types originated from our previous research on investigating the expressivity of session types for parallel algorithms [3]. This idea has been modelled recently in our work [17]. The systems share the similarity of using the $\mathbf{R}$ operator in the syntax of global types and processes/roles, and differ in the programming paradigm and typing mechanisms. Principals in [17] are more theoretical expressive than in this work as they allow more than one parameter per index expression. In [17] there is no notion of role programming, i.e., both processes and functions are first-class constructs. The programming burden is increased at the type level in [17], as programmers have to specify processes local types, making the same errors when writing principals for processes, in addition to global type. In this work, programmers specify only the global type. In [17], the coherence of the local types with respect to the global type is ensured by a type-equivalence relation for every instance of a parameterised global type; to compute all instances, the system is restricted to finite sets of

values for parameters. In this system, the coherence is ensured by an efficient projection algorithm, sorting and $\mathbf{R}$-elimination that allow infinite sets of values for parameters.

Our formal system is modelled after [4], a simpler version of [13], which difference was discussed at the beginning of Section 2; none of these systems are expressive enough to model parameterised communication patterns. The $\mathbf{R}$ operator used in this work, is introduced by Gödel in System *T*. The idea of using the $\mathbf{R}$ operator comes from Nelson's work on adding primitive recursion to the lambda calculus [15]. As a result, his system can type functions previously untyped in ML. Our use of the $\mathbf{R}$ operator models parameterised communication patterns.

Session types have been first introduced by Honda et al. [12, 16] to capture the interaction structure of two processes. Their type system checks whether for each "send" on one process corresponds a "receive" on the other and vice versa. In [13], Honda et al. have extended their system from two-parties to $\mathrm{n}$-parties. With an intuitive syntax, they have introduced a notion of global type to describe the interaction structure of $\mathrm{n}$ processes from a global viewpoint. Multiparty session types have been studied also by Bonelli and Compagnoni [5]. Their type system is defined over binary session types, obtained by projecting processes local types onto principals. Session types have been used to type service-oriented multiparty communications [6]. The calculus proposed permits communications inside and outside a session to model merging of two running sessions. Type safety and progress properties are not provided for the formal model.

Contracts [10] are another typing model of mobile processes, defined over processes as behavioral types and not over channels as session types. Consequently, they can well-type more correct programs than session types. However, the expressiveness of the type system comes at a practical cost. Contracts have no intuitive syntax as global types and no iterative construct as our system. Thus, they do not provide a practical model to design a programming language that supports communication and elegantly expresses parameterised communication patterns; e.g. the key exchange protocol in Section 4 has been augmented with additional communications to check the end of a send-iteration.

The conversation calculus presented in [8] is based on boxed ambients [7] and not in the $\pi$-calculus as session types. Typing is similar with the one of contracts and thus, the system carries the same disadvantages when compared to session types. The calculus models dynamic joining and leaving of participants within a session. We plan to add such features to our system, as the next step, after adding parameters.

**Acknowledgements** I thank Nobuko Yoshida for technical discussions on this material, and Iain Phillips and Raymond Hu for comments on a previous version of this paper.

## References

1. Web Services Choreography Description Language: Primer 1.0. http://www.w3.org/TR/ws-cdl-10-primer/.
2. G. Ateniese, M. Steiner, and G. Tsudik. Authenticated group key agreement and friends. In *CCS*, pages 17–26. ACM, 1998.
3. A. Bejleri, R. Hu, and N. Yoshida. Session-based programming for parallel algorithms: Expressiveness and performance. In *PLACES'09*.
4. L. Bettini and al. Global progress in dynamically interleaved multiparty sessions. In *CONCUR*, volume 5201 of *LNCS*, pages 418–433, 2008.

5. E. Bonelli and A. Compagnoni. Multipoint session types for a distributed calculus. In *TGC'07*, volume 4912 of *LNCS*, 2008.
6. R. Bruni, I. Lanese, H. Melgratti, and E. Tuosto. Multiparty Sessions in SOC. In *COORDINATION'08*, volume 5052 of *LNCS*, pages 67–82. Springer, 2008.
7. M. Bugliesi, G. Castagna, and S. Crafa. Access control for mobile agents: The calculus of boxed ambients. *ACM Trans. Program. Lang. Syst.*, 26(1):57–124, 2004.
8. L. Caires and H. T. Vieira. Conversation types. In *ESOP*, volume 5502 of *LNCS*, pages 285–300. Springer, 2009.
9. M. Carbone, N. Yoshida, and K. Honda. Asynchronous session types: Exceptions and multiparty interactions. In *SFM'09*, volume 5569 of *LNCS*, pages 187–212. Springer, 2009.
10. G. Castagna and L. Padovani. Contracts for mobile processes. In *CONCUR*, volume 5710 of *LNCS*, pages 211–228. Springer, 2009.
11. W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 1999.
12. K. Honda and al. Language primitives and type disciplines for structured communication-based programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998.
13. K. Honda, N. Yoshida, and M. Carbone. Multiparty Asynchronous Session Types. In *POPL'08*, pages 273–284. ACM, 2008.
14. F. T. Leighton. *Introduction to parallel algorithms and architectures: arrays, trees, hypercubes*. Morgan Kaufmann, 1991.
15. N. Nelson. Primitive recursive functionals with dependent types. In *MFPS*, volume 598 of *LNCS*, pages 125–143, 1991.
16. K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In *PARLE'94*, volume 817 of *LNCS*, pages 398–413. Springer, 1994.
17. N. Yoshida, P.-M. Denielou, A. Bejleri, and R. Hu. Parameterised multiparty session types. In *FOSSACS*, LNCS, 2010. To appear.

# A Evaluation Context

Definition of the context is given in Figure 10.

| | | |
|---|---|---|
| $\mathcal{R}[\_,\ldots,\_]$ ::= | | Principals evaluation contexts |
| | $\bar{a}[\_,...,\_](y).R$ | Request |
| \| | $a[\_](y).R$ | Accept |
| \| | $s[\hat{\mathtt{p}}]!\langle\_,e\rangle;R$ | Send |
| \| | $s[\hat{\mathtt{p}}]\oplus\langle\_,l\rangle;R$ | Selection |
| \| | $s[\hat{\mathtt{p}}]?(\_,x);R$ | Receive |
| \| | $s[\hat{\mathtt{p}}]\&(\_,\{l_i:R_i\}_{i\in I})$ | Branching |
| $\mathcal{E}[\_,\ldots,\_]$ ::= | | Expressions evaluation contexts |
| \| | _ op _ | Expression |
| \| | (R _) | Application |
| \| | $c!\langle \mathtt{p},\_\rangle;R$ | Send |
| \| | (E _) | Applicaiton Expr. |

**Fig. 10.** Evaluation contexts

### A.1 Reduction of a Ring instance

The reduction steps of the Ring for $n = 2$ are given below, where on the reduction relation $\longrightarrow$ is attached the name of the rule applied.

$Ring\ 2 \rightarrow_{App}$

$\mathbf{R}\ \bar{a}[\mathsf{W}[0], \mathsf{W}[1], \mathsf{W}\{2/n\}](y).y!\langle \mathsf{W}[1], v\rangle; y?(\mathsf{W}[2], z); P \mid a[\mathsf{W}[2]](y).y?(\mathsf{W}[1], z); y!\langle \mathsf{W}[0], z\rangle; Q$
$\lambda i.\lambda X.(\textit{Middle(i)} \mid X)\ 1 \rightarrow_{Context}$

$\mathbf{R}\ \bar{a}[\mathsf{W}[0], \mathsf{W}[1], \mathsf{W}[2]](y).y!\langle \mathsf{W}[1], v\rangle; y?(\mathsf{W}[2], z); P \mid a[\mathsf{W}[2]](y).y?(\mathsf{W}[1], z); y!\langle \mathsf{W}[0], z\rangle; Q$
$\lambda i.\lambda X.(\textit{Middle(i)} \mid X)\ 1 \rightarrow_{Succ}$

$a[\mathsf{W}[1]](y).y?(\mathsf{W}[0], z); y!\langle \mathsf{W}[2], z\rangle;$
$\mid X\{\mathbf{R}\ \bar{a}[\mathsf{W}[0], \mathsf{W}[1], \mathsf{W}[2]](y).y!\langle \mathsf{W}[1], v\rangle; y?(\mathsf{W}[2], z); P \mid a[\mathsf{W}[2]](y).y?(\mathsf{W}[1], z); y!\langle \mathsf{W}[0], z\rangle; Q$
$\lambda i.\lambda X.(\textit{Middle(i)} \mid X)\ 0/X\} \rightarrow_{Zero}$

$a[\mathsf{W}[1]](y).y?(\mathsf{W}[0], z); y!\langle \mathsf{W}[2], z\rangle;\ \mid \bar{a}[\mathsf{W}[0], \mathsf{W}[1], \mathsf{W}[2]](y).y!\langle \mathsf{W}[1], v\rangle; y?(\mathsf{W}[2], z); P$
$\mid a[\mathsf{W}[2]](y).y?(\mathsf{W}[1], z); y!\langle \mathsf{W}[0], z\rangle; Q \rightarrow_{Link}$

$(\nu s)(s[\mathsf{W}[1]]?(\mathsf{W}[0], z); s[\mathsf{W}[1]]!\langle \mathsf{W}[2], z\rangle;\ \mid s[\mathsf{W}[0]]!\langle \mathsf{W}[1], v\rangle; s[\mathsf{W}[0]]?(\mathsf{W}[2], z); P$
$\mid s[\mathsf{W}[2]?(\mathsf{W}[1], z); s[\mathsf{W}[2]!\langle \mathsf{W}[0], z\rangle; Q \mid s : \emptyset) \rightarrow_{Scop,Send}$

$(\nu s)(s[\mathsf{W}[1]]?(\mathsf{W}[0], z); s[\mathsf{W}[1]]!\langle \mathsf{W}[2], z\rangle;\ \mid s[\mathsf{W}[0]]?(\mathsf{W}[2], z); P$
$\mid s[\mathsf{W}[2]?(\mathsf{W}[1], z); s[\mathsf{W}[2]!\langle \mathsf{W}[0], z\rangle; Q \mid s : \langle \mathsf{W}[0], \mathsf{W}[1], v\rangle) \rightarrow_{Scop,Recv}$

$(\nu s)(s[\mathsf{W}[1]]!\langle \mathsf{W}[2], v\rangle;\ \mid s[\mathsf{W}[0]]?(\mathsf{W}[2], z); P \mid s[\mathsf{W}[2]?(\mathsf{W}[1], z); s[\mathsf{W}[2]!\langle \mathsf{W}[0], z\rangle; Q \mid s : \emptyset) \rightarrow_{Scop,Send}$

$(\nu s)(s[\mathsf{W}[0]]?(\mathsf{W}[2], z); P \mid s[\mathsf{W}[2]?(\mathsf{W}[1], z); s[\mathsf{W}[2]!\langle \mathsf{W}[0], z\rangle; Q \mid s : \langle \mathsf{W}[1], \mathsf{W}[2], v\rangle) \rightarrow_{Scop,Recv}$

$(\nu s)(s[\mathsf{W}[0]]?(\mathsf{W}[2], z); P \mid s[\mathsf{W}[2]!\langle \mathsf{W}[0], v\rangle; Q \mid s : \emptyset) \rightarrow_{Scop,Send}$

$(\nu s)(s[\mathsf{W}[0]]?(\mathsf{W}[2], z); P \mid Q \mid s : \langle \mathsf{W}[2], \mathsf{W}[0], v\rangle) \rightarrow_{Scop,Recv}$

$(\nu s)(P\{v/z\} \mid Q \mid s : \emptyset)$

# B Typing

## B.1 Kinding of Global Types

In Figure 11 and 12 are given the auxiliary definitions for kinding of value-types and role-types.

In [17], the kinding system allows kinded subterms in the primitive recursive global types. We do not allow kinded subterms in primtive recursive term as it breaks the soundness of the system. We illustrate the problem with a global type which is kinded by Type in the system of [17] and which after two reductions is not kinded.

$\mathbf{R}\ \mathbf{R}$ end $\lambda i.\lambda \mathbf{x}.\mathtt{W}[i] \to \mathtt{W}[i+1] : \langle U \rangle.\mathbf{x}$
$j.\lambda y.\mathbf{R}$ end $\lambda k.\lambda z.\mathtt{W}[j] \to \mathtt{W}[j+1] : \{true : y, false : z\}\ 1\ 1 \longrightarrow$
$\mathbf{R}$ end $\lambda k.\lambda z.\mathtt{W}[1] \to \mathtt{W}[2] : \{true : \mathbf{R}$ end $\lambda i.\lambda \mathbf{x}.\mathtt{W}[i] \to \mathtt{W}[i+1] : \langle U \rangle.\mathbf{x}, false : z\}\ 1 \longrightarrow$
$\mathtt{W}[1] \to \mathtt{W}[2] : \{true : \mathbf{R}$ end $\lambda i.\lambda \mathbf{x}.\mathtt{W}[i] \to \mathtt{W}[i+1] : \langle U \rangle.\mathbf{x}, false :$ end$\}$

$$\frac{}{\Theta; C \vdash \mathsf{bool} \blacktriangleright \mathsf{Type}}\ \lfloor \text{KBOOL} \rfloor$$

$$\frac{}{\Theta; C \vdash \mathsf{nat} \blacktriangleright \mathsf{Type}}\ \lfloor \text{KNAT} \rfloor$$

$$\frac{\Theta; C \vdash G \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \langle G \rangle \blacktriangleright \mathsf{Type}}\ \lfloor \text{KMAR} \rfloor$$

**Fig. 11.** Kinding rules for value types

$$\frac{C \vdash \mathtt{p} \quad \Theta; C \vdash U \blacktriangleright \mathsf{Type} \quad \mathsf{fprtv}(U) = \emptyset \quad \Theta; C \vdash T \blacktriangleright \mathsf{Type}}{\Theta; C \vdash !\langle \mathtt{p}, U\rangle; T \blacktriangleright \mathsf{Type}} \lfloor \text{KROUT} \rfloor$$

$$\frac{C \vdash \mathtt{p} \quad \Theta; C \vdash U \blacktriangleright \mathsf{Type} \quad \mathsf{fprtv}(U) = \emptyset \quad \Theta; C \vdash T \blacktriangleright \mathsf{Type}}{\Theta; C \vdash ?\langle \mathtt{p}, U\rangle; T \blacktriangleright \mathsf{Type}} \lfloor \text{KRIN} \rfloor$$

$$\frac{C \vdash \mathtt{p} \quad \forall i \in I. \Theta; C \vdash T_i \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \oplus\langle \mathtt{p}, \{l_i : T_i\}_{i\in I}\rangle \blacktriangleright \mathsf{Type}} \lfloor \text{KRSEL} \rfloor \quad \frac{C \vdash \mathtt{p} \quad \forall i \in I. \Theta; C \vdash T_i \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \&\langle \mathtt{p}, \{l_i : T_i\}_{i\in I}\rangle \blacktriangleright \mathsf{Type}} \lfloor \text{KRBRA} \rfloor$$

$$\frac{\Theta, \mathbf{t} : \mathsf{Type}; C \vdash T \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \mu \mathbf{t}.T \blacktriangleright \mathsf{Type}} \lfloor \text{KRREC} \rfloor \quad \frac{}{\Theta, \mathbf{t} : \mathsf{Type}; C \vdash \mathbf{t} \blacktriangleright \mathsf{Type}} \lfloor \text{KRVAR} \rfloor$$

$$\frac{\Theta; C \vdash T \blacktriangleright \mathsf{Type} \quad \Theta, \mathbf{x} : \mathsf{Type}; C, i : \mathtt{I} \vdash T' \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \mathbf{R}\, T\, \lambda i.\lambda \mathbf{x}.T' \blacktriangleright \Pi i : \mathtt{I}.\mathsf{Type}} \lfloor \text{KRPREC} \rfloor$$

$$\frac{}{\Theta, \mathbf{t} : \mathsf{Type}; C \vdash \mathbf{x} \blacktriangleright \mathsf{Type}} \lfloor \text{KRTVAR} \rfloor \quad \frac{}{\Theta; C \vdash \mathsf{end} \blacktriangleright \mathsf{Type}} \lfloor \text{KREND} \rfloor$$

$$\frac{C \vdash \mathtt{t} \quad \Theta; C \vdash T \blacktriangleright \Pi i{:}\{i | i \in \mathsf{nat}, 0 \leq i \leq \mathtt{t} - 1\}.\mathsf{Type}}{\Theta; C \vdash T\ \mathtt{t} \blacktriangleright \mathsf{Type}} \lfloor \text{KAPP} \rfloor$$

**Fig. 12.** Kinding rules for role types

### B.2 Auxiliary Definitions

In this section, we give the definition of free primitive recursion variables in global types, Figure 13, principals in a global type Figure 14, free index variable of the principal that performs each action on role types, Figure 15 and free index variables in principals, Figure 16. The $\lfloor$KREC$\rfloor$ rule checks only that the inductive part is not a $\mathbf{R}$ global type. The $\lfloor$KRVAR$\rfloor$ and $\lfloor$KPRVAR$\rfloor$ looks up for type variables in the context of variable typing. The $\lfloor$KEND$\rfloor$ rule cheks if the typing context is well-formed.

$$\mathsf{fprtv}(\mathtt{p} \to \mathtt{p}' : \langle U \rangle.G) = \mathsf{fprtv}(G) \cup \mathsf{fprtv}(U)$$
$$\mathsf{fprtv}(\mathtt{p} \to \mathtt{p}' : \{l_i : G_i\}_{i \in I}) = \bigcup_{i \in I} \mathsf{fprtv}(G_i)$$
$$\mathsf{fprtv}(\mu \mathbf{t}.G) = \mathsf{fprtv}(G) \quad \mathsf{fprtv}(\mathbf{t}) = \emptyset$$
$$\mathsf{fprtv}(\mathbf{R}\ G\ \lambda i.\lambda \mathbf{x}.G') = \mathsf{fprtv}(G) \cup \mathsf{fprtv}(G') \setminus \{\mathbf{x}\}$$
$$\mathsf{fprtv}(G\ \mathtt{t}) = \mathsf{fprtv}(G) \quad \mathsf{fprtv}(\mathbf{x}) = \{\mathbf{x}\} \quad \mathsf{fprtv}(\mathsf{end}) = \emptyset$$

$$\mathsf{fprtv}(\mathsf{bool}) = \emptyset \quad \mathsf{fprtv}(\langle G \rangle) = \mathsf{fprtv}(G)$$

$$\mathsf{fprtv}(!\langle \mathtt{p}, U \rangle; T) = \mathsf{fprtv}(T) \cup \mathsf{fprtv}(U)$$
$$\mathsf{fprtv}(?\langle \mathtt{p}, U \rangle; T) = \mathsf{fprtv}(T) \cup \mathsf{fprtv}(U)$$
$$\mathsf{fprtv}(\oplus\langle \mathtt{p}, \{l_i : T_i\}_{i \in I} \rangle) = \bigcup_{i \in I} \mathsf{fprtv}(T_i)$$
$$\mathsf{fprtv}(\&\langle \mathtt{p}, \{l_i : T_i\}_{i \in I} \rangle) = \bigcup_{i \in I} \mathsf{fprtv}(T_i)$$
$$\mathsf{fprtv}(\mu \mathbf{t}.T) = \mathsf{fprtv}(T) \quad \mathsf{fprtv}(\mathbf{t}) = \emptyset$$
$$\mathsf{fprtv}(\mathbf{R}\ T\ \lambda i.\lambda \mathbf{x}.T') = \mathsf{fprtv}(T) \cup \mathsf{fprtv}(T') \setminus \{\mathbf{x}\}$$
$$\mathsf{fprtv}(T\ \mathtt{i}) = \mathsf{fprtv}(T) \quad \mathsf{fprtv}(\mathbf{x}) = \{\mathbf{x}\} \quad \mathsf{fprtv}(\mathsf{end}) = \emptyset$$

**Fig. 13.** Free primitive recursive type variables

$$\mathsf{pid}(\mathtt{p} \to \mathtt{p}' : \langle U \rangle.G) = \{\mathtt{p}, \mathtt{p'}\} \cup \mathsf{pid}(G)$$
$$\mathsf{pid}(\mathtt{p} \to \mathtt{p}' : \{l_i : G_i\}_{i \in I}) = \{\mathtt{p}, \mathtt{p'}\} \cup \bigcup_{i \in I} \mathsf{pid}(G_i)$$
$$\mathsf{pid}(\mu \mathbf{t}.G) = \mathsf{pid}(G) \quad \mathsf{pid}(\mathbf{t}) = \emptyset$$
$$\mathsf{pid}(\mathbf{R}\ G\ \lambda i.\lambda \mathbf{x}.G') = \mathsf{pid}(G) \cup \mathsf{pid}(G')$$
$$\mathsf{pid}(G\ \mathtt{t}) = \mathsf{pid}(G) \quad \mathsf{pid}(\mathsf{end}) = \emptyset \quad \mathsf{pid}(\mathbf{x}) = \emptyset$$

**Fig. 14.** Principal identifiers of a global type

$$\mathrm{fivr}(!\langle \mathtt{p}, U \rangle(\mathtt{p}'); T) = \mathsf{fiv}(\mathtt{p}') \cup \mathrm{fivr}(T)$$
$$\mathrm{fivr}(?\langle \mathtt{p}, U \rangle(\mathtt{p}'); T) = \mathsf{fiv}(\mathtt{p}') \cup \mathrm{fivr}(T)$$
$$\mathrm{fivr}(\oplus\langle \mathtt{p}, \{l_i : T_i\}_{i \in I} \rangle(\mathtt{p}')) = \mathsf{fiv}(\mathtt{p}') \bigcup_{i \in I} \mathrm{fivr}(T_i)$$
$$\mathrm{fivr}(\&\langle \mathtt{p}, \{l_i : T_i\}_{i \in I} \rangle(\mathtt{p}')) = \mathsf{fiv}(\mathtt{p}') \bigcup_{i \in I} \mathrm{fivr}(T_i)$$
$$\mathrm{fivr}(\mu \mathbf{t}.T) = \mathrm{fivr}(T) \quad \mathrm{fivr}(\mathbf{t}) = \emptyset$$
$$\mathrm{fivr}(\mathbf{R}\ T\ \lambda i.\lambda x.T') = \mathrm{fivr}(T) \cup \mathrm{fivr}(T') \setminus \{i\}$$
$$\mathrm{fivr}(T\ \mathtt{t}) = \mathrm{fivr}(T) \quad \mathrm{fivr}(\mathbf{x}) = \emptyset \quad \mathrm{fivr}(\mathsf{end}) = \emptyset$$

**Fig. 15.** Free index variables of the principal that perform each action of roles

$$\mathsf{fiv}(\mathcal{N}) = \emptyset$$
$$\mathsf{fiv}(\mathtt{p}[\mathtt{i}]) = \mathsf{fiv}(\mathtt{i}) \cup \mathsf{fiv}(\mathtt{p})$$

$$\mathsf{fiv}(\mathtt{t}) = \emptyset$$
$$\mathsf{fiv}(i) = \{i\}$$
$$\mathsf{fiv}(\mathrm{n} * \mathtt{i}) = \mathsf{fiv}(\mathtt{i})$$
$$\mathsf{fiv}(\mathtt{t} \pm \mathtt{i}) = \mathsf{fiv}(\mathtt{i})$$

**Fig. 16.** Free index variables of principals

### B.3 Well-formedness of Principals

A well-formed principal is either a participant or an indexed principal where the set of values of each index expression is defined over naturals.We have defined a set of rules that ensure when subtraction can be used safely in expressions. Auxiliary definition of $\max \mathtt{I}$ and $\min \mathtt{T}$ are given below. Figure 19 defines well-formedness of principal lists.

**Definition B.1** *The minimum value of a parameter range is defined as:*

$$\mathsf{min}(\{n \mid n \in \mathsf{nat}, n \geq \mathrm{n}\}) = \mathrm{n}$$

**Definition B.2** *The maximum value of an index range is defined as:*

$$\mathsf{max}(\{i \mid i \in \mathsf{nat}, 0 \leq i \leq \mathtt{t}\}) = \mathtt{t}$$

$$\frac{}{C \vdash \mathtt{Alice}} \qquad \frac{C \vdash \mathtt{i} \quad C \vdash \mathtt{p}}{C \vdash \mathtt{p}[\mathtt{i}]}$$

**Fig. 17.** Well-formedness of principals

$$\frac{i \in \mathit{dom}(C)}{C \vdash i} \qquad \frac{C \vdash \mathtt{i}}{C \vdash \mathrm{n} * \mathtt{i}} \qquad \frac{C \vdash \mathtt{t} \quad C \vdash \mathtt{i}}{C \vdash \mathtt{t} + \mathtt{i}}$$

$$\frac{C \vdash \mathtt{t} \geq \max(\mathtt{i})}{C \vdash \mathtt{t} - \mathtt{i}}$$

$$\frac{}{C \vdash \mathrm{n}} \qquad \frac{n \in \mathit{dom}(C)}{C \vdash n} \qquad \frac{C \vdash \mathtt{t}}{C \vdash \mathtt{t} * \mathrm{n}} \qquad \frac{C \vdash \mathtt{t}}{C \vdash \mathtt{t} + \mathrm{n}}$$

$$\frac{C \vdash \mathtt{t} \geq \mathrm{n}}{C \vdash \mathtt{t} - \mathrm{n}} \qquad \frac{C \vdash \mathtt{t}}{C \vdash \mathrm{n}^{\mathtt{t}}}$$

**Fig. 18.** Well-formedness of index expressions

$$\frac{C \vdash \mathtt{p}_1 .... C \vdash \mathtt{p}_n}{C \vdash \mathtt{p}_1, ..., \mathtt{p}_n} \qquad \frac{C \vdash p \quad C \vdash \mathtt{t} \quad C, i : \{i | 0 \leq i \leq \mathtt{t} - 1\} \vdash p'}{C \vdash \mathbf{R}\ p\ \lambda i.\lambda X.p'\ \mathtt{t}}$$

**Fig. 19.** Well-formedness of principals list

### B.4 Principals Equality

Figure 20 defines the equality relation between a global type and a role type principal. Following, in Figure 24 equality between index expressions, Figure 21 equality between parametric expressions, Figure 22 minimum and maximum values of an index expression, Figure 23 inequality between parametric expressions.

$$C \vdash \mathtt{Alice} = \mathtt{Alice}$$

$$\frac{C \vdash \mathtt{p} = \mathtt{p}' \quad C \vdash \mathtt{i} = \mathtt{i}'}{C \vdash \mathtt{p}[\mathtt{i}] = \mathtt{p}'[\mathtt{i}']}$$

**Fig. 20.** Equality between principals

$$\frac{}{C \vdash \mathrm{n} = \mathrm{n}} \qquad \frac{C \vdash n : \mathtt{T}}{C \vdash n = n}$$

$$\frac{C \vdash \mathtt{t} = \mathtt{t}'}{C \vdash \mathtt{t}\ \mathtt{op}\ \mathrm{n} = \mathtt{t}'\ \mathtt{op}\ \mathrm{n}} \qquad \frac{C \vdash \mathtt{t} = \mathtt{t}'}{C \vdash \mathrm{n}^{\mathtt{t}} = \mathrm{n}^{\mathtt{t}'}}$$

**Fig. 21.** Equality between parameters

$$C \vdash \min(\mathtt{t}) = \mathtt{t} \quad C \vdash \min(i) = 0$$
$$C \vdash \min(\mathrm{n} * \mathtt{i}) = \mathrm{n} * \mathcal{C} \vdash \min(\mathtt{i})$$
$$C \vdash \min(\mathtt{t} \pm \mathtt{i}) = \mathtt{t} \pm C \vdash \min(\mathtt{i})$$

$$C \vdash \max(\mathtt{t}) = \mathtt{t} \quad C \vdash \max(i) = \max(C(i))$$
$$C \vdash \max(\mathrm{n} * \mathtt{i}) = \mathrm{n} * C \vdash \max(\mathtt{i})$$
$$C \vdash \max(\mathtt{t} \pm \mathtt{i}) = \mathtt{t} \pm C \vdash \max(\mathtt{i})$$

**Fig. 22.** Minimum and maximum value of an index expression

$$\frac{}{C \vdash \mathrm{n} \geq 0} \quad \frac{C \vdash n : \{n | n \geq \mathrm{n}\}}{C \vdash n \geq 0} \quad \frac{C \vdash \mathtt{t} \geq 0}{C \vdash \mathtt{t} * \mathrm{n} \geq 0}$$

$$\frac{C \vdash \mathtt{t} \geq \mathrm{n}}{C \vdash \mathtt{t} - \mathrm{n} \geq 0} \quad \frac{C \vdash \mathtt{t} \geq 0}{C \vdash \mathtt{t} + \mathrm{n} \geq 0} \quad \frac{C \vdash \mathtt{t} \geq 0}{C \vdash n^{\mathtt{t}} \geq 0}$$

$$\frac{\mathrm{n} \geq \mathrm{n}'}{C \vdash \mathrm{n} \geq \mathrm{n}'} \quad \frac{C \vdash n : \{n | n \geq \mathrm{n}'\} \quad \mathrm{n}' \geq \mathrm{n}}{C \vdash n \geq \mathrm{n}} \quad \frac{C \vdash \mathtt{t} \geq \mathrm{n}'/\,\mathrm{n} \quad \mathrm{n}'\%\mathrm{n} = 0}{C \vdash \mathtt{t} * \mathrm{n} \geq \mathrm{n}'}$$

$$\frac{C \vdash \mathtt{t} \geq \mathrm{n}' - \mathrm{n}}{C \vdash \mathtt{t} + \mathrm{n} \geq \mathrm{n}'} \quad \frac{C \vdash \mathtt{t} \geq \mathrm{n}' + \mathrm{n}}{C \vdash \mathtt{t} - \mathrm{n} \geq \mathrm{n}'} \quad \frac{C \vdash \mathtt{t} \geq \lg_{\mathrm{n}} \mathrm{n}' \quad \mathrm{n}' = \mathrm{n}^{\mathrm{n}''}}{C \vdash \mathrm{n}^{\mathtt{t}} \geq \mathrm{n}'}$$

$$\frac{C \vdash \mathtt{t}' - \mathtt{t} \geq 0}{C \vdash \mathtt{t} \leq \mathtt{t}'}$$

**Fig. 23.** Inequality between parameters

$$\frac{C \vdash i : \{i | 0 \leq i \leq \mathtt{t}'\} \quad C \vdash \mathtt{t} \leq \mathtt{t}'}{C \vdash \mathtt{t} = i} \qquad \frac{C \vdash \mathtt{t} = \mathtt{i}}{C \vdash \mathtt{t} * \mathrm{n} = \mathtt{i} * \mathrm{n}}$$

$$\frac{C \vdash \mathtt{t} - \mathtt{t}' = \mathtt{i}}{C \vdash \mathtt{t} = \mathtt{t}' + \mathtt{i}} \quad \frac{C \vdash \mathtt{t}' - \mathtt{t} = \mathtt{i}}{C \vdash \mathtt{t} = \mathtt{t}' - \mathtt{i}}$$

$$\frac{C \vdash i : \{i | 0 \leq i \leq \mathtt{t}'\} \quad C \vdash \mathtt{t} \leq \mathtt{t}'}{C \vdash i = \mathtt{t}}$$

$$\frac{C \vdash i : \{i | 0 \leq i \leq \mathtt{t}\} \quad C \vdash \max(j) \leq \mathtt{t}}{C \vdash i = j}$$

$$\frac{C \vdash i : \{i | 0 \leq i \leq \mathtt{t}'\} \quad C \vdash \min(\mathtt{t} + \mathtt{j}) \geq 0 \quad C \vdash \max(\mathtt{t} + \mathtt{j}) \leq \mathtt{t}'}{C \vdash i = \mathtt{t} + \mathtt{j}}$$

$$\frac{C \vdash i : \{i | 0 \leq i \leq \mathtt{t}'\} \quad C \vdash \min(\mathtt{t} - \mathtt{j}) \geq 0 \quad C \vdash \max(\mathtt{t} - \mathtt{j}) \leq \mathtt{t}'}{C \vdash i = \mathtt{t} - j}$$

$$\frac{C \vdash \mathtt{i} = \mathtt{t}}{C \vdash n * \mathtt{i} = n * \mathtt{t}} \qquad \frac{C \vdash \mathtt{i} = \mathtt{t} + \mathtt{j}}{C \vdash n * \mathtt{i} = n * \mathtt{t} + n * \mathtt{j}} \qquad \frac{C \vdash \mathtt{i} = \mathtt{t} - \mathtt{j}}{C \vdash n * \mathtt{i} = n * \mathtt{t} - n * \mathtt{j}}$$

$$\frac{C \vdash \mathtt{i} = \mathtt{i}' - \mathtt{t}}{C \vdash \mathtt{t} + \mathtt{i} = \mathtt{i}'} \quad \frac{C \vdash \mathtt{i} = j - \mathtt{t}}{C \vdash \mathtt{t} + \mathtt{i} = j} \qquad \frac{C \vdash \mathtt{t} + \mathtt{i} = \mathtt{j}}{C \vdash n * \mathtt{t} + n * \mathtt{i} = n * \mathtt{j}}$$

$$\frac{C \vdash \mathtt{i} = \mathtt{t}' - \mathtt{t} + \mathtt{j}}{C \vdash \mathtt{t} + \mathtt{i} = \mathtt{t}' + \mathtt{j}} \quad \frac{C \vdash \mathtt{i} = \mathtt{t}' - \mathtt{t} - \mathtt{j}}{C \vdash \mathtt{t} + \mathtt{i} = \mathtt{t}' - \mathtt{j}}$$

$$\frac{C \vdash \mathtt{i} = \mathtt{t} - \mathtt{i}'}{C \vdash \mathtt{t} - \mathtt{i} = \mathtt{i}'} \quad \frac{C \vdash \mathtt{i} = \mathtt{t} - j}{C \vdash \mathtt{t} - \mathtt{i} = j} \qquad \frac{C \vdash \mathtt{t} - \mathtt{i} = \mathtt{j}}{C \vdash n * \mathtt{t} - n * \mathtt{i} = n * \mathtt{j}}$$

$$\frac{C \vdash \mathtt{i} = \mathtt{t} - \mathtt{t}' - \mathtt{j}}{C \vdash \mathtt{t} - \mathtt{i} = \mathtt{t}' + \mathtt{j}} \quad \frac{C \vdash \mathtt{i} = \mathtt{t} - \mathtt{t}' + \mathtt{j}}{C \vdash \mathtt{t} - \mathtt{i} = \mathtt{t}' - \mathtt{j}}$$

**Fig. 24.** Equality between indexes

### B.5 Substitution of Principals

$$\mathtt{W}\{\mathtt{W}{=}\mathtt{W}\} = \mathtt{W}$$

$$\mathtt{p[i]}\{\mathtt{p'[i']{=}q[j]}\} = \mathtt{p}\{\mathtt{p'{=}q}\}[\mathtt{i}\{\mathtt{i'{=}j}\}]$$

**Fig. 25.** Substitution on principals

$$\mathtt{i}\{\mathtt{i'{=}j}\} = \begin{cases} \mathtt{i}\{\mathtt{j}'/i\} & \text{if } i \in \mathsf{fiv}(\mathtt{i}) \text{ and } \mathtt{i}' = \mathtt{j} \rightarrow^{\star} i = \mathtt{j}' \\ \mathtt{i} & \text{otherwise} \end{cases}$$

**Fig. 26.** Substitution on indexes

$$\begin{array}{c} i{=}\,\mathtt{j} \\ \mathtt{n{*}i{=}\ n{*}j} \rightarrow \mathtt{i{=}\ j} \\ \mathtt{t{+}\ i{=}\ j} \rightarrow \mathtt{i{=}\ j{-}\ t} \\ \mathtt{t{-}\ i{=}\ j} \rightarrow \mathtt{i{=}\ t{-}\ j} \end{array}$$

**Fig. 27.** Transformation on index expressions equalities

### B.6 Principal Set Equality

**Definition B.3** $C \vdash \mathsf{pid}(G) = \{\mathtt{p}_0, \mathtt{p}_1, p\}$ if $C \vdash \mathsf{pid}(G) \subseteq \{\mathtt{p}_0, \mathtt{p}_1, p\}$ and $C \vdash \{\mathtt{p}_0, \mathtt{p}_1, p\} \subseteq \mathsf{pid}(G)$.

**Definition B.4** $C \vdash \{\mathtt{p}'_0, \mathtt{p}'_1, ..., \mathtt{p}'_n\} \subseteq \{\mathtt{p}_0, \mathtt{p}_1, p\}$ if $\forall \mathtt{p} \in \{\mathtt{p}'_0, \mathtt{p}'_1, ..., \mathtt{p}'_n\}.C; \mathtt{range}(\mathtt{p}) \vdash \mathtt{p} \in \{\mathtt{p}_0, \mathtt{p}_1, p\}$.

**Definition B.5** $C \vdash \{\mathtt{p}_0, \mathtt{p}_1, p\} \subseteq \{\mathtt{p}'_0, \mathtt{p}'_1, ..., \mathtt{p}'_n\}$ if $C \vdash \mathtt{p}_0, \mathtt{p}_1, p \in \{\mathtt{p}'_0, \mathtt{p}'_1, ..., \mathtt{p}'_n\}$.

**Definition B.6** $C \vdash \mathbf{R}\ p\ \lambda i.\lambda x.p'\ \mathtt{t} \in \{\mathtt{p}'_0, \mathtt{p}'_1, ..., \mathtt{p}'_n\}$ if $C, i : \{i | 0 \leq i \leq \mathtt{t} - 1\} \vdash \{p, p'\} \subseteq \{\mathtt{p}'_0, \mathtt{p}'_1, ..., \mathtt{p}'_n\}$.

**Definition B.7** $\mathtt{intersection}(C, [a_1..b_1]..[a_j..b_j], \mathtt{p}, \mathtt{p}') = [a_1..b_1]..[a_j..b_j] \backslash [a'_1..b'_1]...[a'_k..b'_k]$ if $C \vdash \mathtt{p} = \mathtt{p}'$ where $\mathtt{range}(\mathtt{p}') = [a''..b'']$ and $[a'_1..b'_1]...[a'_k..b'_k] = [a_1..b_1]..[a_j..b_j] \cap [a''..b'']$.

$$\frac{\texttt{intersection}(C, [a..b], \mathtt{p}, \mathtt{p}_1) = [a..b]}{C; [a..b] \vdash \mathtt{p} \in \{\mathtt{p}_1, ..., \mathtt{p}_n\}}$$

$$\frac{\texttt{intersection}(C, [a_1..b_1]..[a_j..b_j], \mathtt{p}, \mathtt{p}_1) = [a'_1..b'_1]..[a'_i..b'_i] \quad C; [a'_1..b'_1]..[a'_i..b'_i] \vdash \mathtt{p} \in \{..., \mathtt{p}_n\}}{C; [a_1..b_1]..[a_j..b_j] \vdash \mathtt{p} \in \{\mathtt{p}_1, ..., \mathtt{p}_n\}}$$

$$\frac{C, i : \{i | 0 \leq i \leq \mathtt{t} - 1\}; [a_1..b_1]..[a_j..b_j] \vdash \mathtt{p} \in \{p, p'\}}{C; [a_1..b_1]..[a_j..b_j] \vdash \mathtt{p} \in \{\mathbf{R}\ p\ \lambda i.\lambda x.p'\ \mathtt{t}\}}$$

**Fig. 28.** Membership of a set

## C Proofs

In this section, we give the full proof of type preservation for the formal system presented in this paper; i.e. if a term is well-typed and it reduces to a new term, then the new term is also well-typed. The proof relies on standard substitution lemmas.

**Lemma C.1 (Type Preservation Under Substitution)** *1. If* $\Gamma; C, n : \mathtt{T} \vdash E \rhd \tau$ *and* $C \vdash \mathrm{n} \geq \min(\mathtt{T})$*, then* $\Gamma; C\{\mathrm{n}/n\} \vdash E\{\mathrm{n}/n\} \vdash \tau\{\mathrm{n}/n\}$*,*

*2. If* $\Gamma; C, i : \mathtt{I} \vdash R \rhd \Delta$ *and* $\mathrm{n} \leq \max(\mathtt{I})$*, then* $\Gamma\{\mathrm{n}/i\}; C \vdash R\{\mathrm{n}/i\} \rhd \Delta\{\mathrm{n}/i\}$*,*

*3. If* $\Gamma, X : \Delta; C \vdash R \rhd \Delta'$ *and* $\Gamma; C \vdash S \rhd \Delta$*, then* $\Gamma; C \vdash R\{S/X\} \rhd \Delta'$*.*

*4. If* $\Gamma, C \vdash R \rhd \Delta, y : T$ *then* $\Gamma; C \vdash R\{s[\hat{\mathtt{p}}]/y\} \rhd \Delta, s[\hat{\mathtt{p}}] : T$*.*

*Proof.* (1) is by induction on the typing judgement $\Gamma; C, n : \mathtt{T} \vdash E \rhd \tau$. We present the cases appealing to the contributions to this work.

| | |
|---|---|
| $\Gamma; C, n : \mathtt{T} \vdash \lambda \mathrm{m}.E \rhd \Pi m : \mathtt{T}'.\tau$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ | By assumption |
| $\Gamma; C, n : \mathtt{T}, m : \mathtt{T}' \vdash \lambda E \rhd \tau$ $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ | By inversion |
| $\Gamma; C\{\mathrm{n}/n\}, m : \mathtt{T}' \vdash E\{\mathrm{n}/n\} \vdash \tau\{\mathrm{n}/n\}$ | By i.h. |
| $\Gamma; C\{\mathrm{n}/n\}, \vdash \lambda m.E\{\mathrm{n}/n\} \vdash Pim : \mathtt{T}.\tau\{\mathrm{n}/n\}$ | By rule |

where $(\Pi m : \mathtt{T}.\tau)\{\mathrm{n}/n\} = \Pi m : \mathtt{T}.(\tau\{\mathrm{n}/n\})$.

| | |
|---|---|
| $\Gamma; C, n : \mathtt{T} \vdash \lambda E\ \mathtt{t} \rhd \tau$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ | By assumption |
| $\Gamma; C, n : \mathtt{T} \vdash \lambda E \rhd \Pi m : \mathtt{T}' : \tau$, $\mathcal{C} \vdash \mathtt{t} \geq \min(\mathtt{T}')$, $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ | By inversion |
| $\Gamma; C\{\mathrm{n}/n\} \vdash E\{\mathrm{n}/n\} \vdash \Pi m : \mathtt{T}'.\tau\{\mathrm{n}/n\}$, $\mathcal{C} \vdash \mathtt{t} \geq \min(\mathtt{T}')$ | By i.h. |
| $\Gamma; C\{\mathrm{n}/n\}, \vdash E\{\mathrm{n}/n\}\ \mathtt{t} \vdash \tau\{\mathrm{n}/n\}$ | By rule |

where $(\Pi m : \mathtt{T}.\tau)\{\mathrm{n}/n\} = \Pi m : \mathtt{T}.(\tau\{\mathrm{n}/n\})$.

| | |
|---|---|
| $\Gamma; C, n : \mathtt{T} \vdash \mathbf{R}\ S \lambda i.\lambda X.R \rhd \Pi i : \mathtt{I}.\Delta$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ | By assumption |
| $\Gamma; C, n : \mathtt{T} \vdash S \rhd \Delta\ 0$, $\Gamma, X : \Delta\ i; C, n : \mathtt{T}, i : \mathtt{I} \vdash R \rhd \Delta\ i + 1$, $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ | By inversion |
| $\Gamma; C\{\mathrm{n}/n\} \vdash S\{\mathrm{n}/n\} \rhd \Delta\{\mathrm{n}/n\}\ 0$, $\Gamma, X : \Delta\ i; C\{\mathrm{n}/n\} i : \mathtt{I} \vdash R\{\mathrm{n}/n\} \rhd$ $\Delta\{\mathrm{n}/n\}\ i + 1)$ | By i.h. |
| $\Gamma; C\{\mathrm{n}/n\} \vdash \mathbf{R}\ S\{\mathrm{n}/n\} \lambda i.\lambda X.R\{\mathrm{n}/n\} \rhd \Pi i : \mathtt{I}.(\Delta\{\mathrm{n}/n\})$ | By rule |

The remaining rules are similar. The proof of (2) is similar to (1). The proofs of (3) and (4) are the same as in [4] as for the rules of our contributions they do not apply.

**Definition C.2** *We generate* $\Delta \Rightarrow \Delta'$ *by the following rules:*

$$\{s[\hat{\mathtt{p}}] :! \langle \hat{\mathtt{q}}, U \rangle; T, s[\hat{\mathtt{q}}] :? \langle \hat{\mathtt{p}}, U \rangle; T'\} \ \Rightarrow\ \{s[\hat{\mathtt{p}}] : T, s[\hat{\mathtt{q}}] : T'\}$$

$$\{s[\hat{\mathtt{p}}] : \oplus \langle \hat{\mathtt{q}}, \{l_j : T_j\}_{j \in I} \rangle, s[\hat{\mathtt{q}}] : \& \langle \hat{\mathtt{p}}, \{l_k : T'_k\}_{k \in J} \rangle\} \ \Rightarrow\ \{s[\hat{\mathtt{p}}] : T_i, s[\hat{\mathtt{q}}] : T'_i\}$$

$$\{s[\hat{\mathtt{p}}] : T, ...\} \Rightarrow \{s[\hat{\mathtt{p}}] : T\{\mathrm{n}/n\}, ...\}$$

$$\Delta \Rightarrow \Delta' \ \textit{if}\ \{s[\hat{\mathtt{p}}] : T_1, s[\hat{\mathtt{q}}] : T_2\} \Rightarrow \{s[\hat{\mathtt{p}}] : T'_1, s[\hat{\mathtt{q}}] : T'_2\}\ \textit{where}\ \{s[\hat{\mathtt{p}}] : T_1, s[\hat{\mathtt{q}}] : T_2\} \subseteq \Delta$$

**Theorem C.3 (Type Preservation)** *If* $\Gamma; C \vdash E \rhd \tau$*, and* $E \rightarrow^\star E'$*, then there exists* $\tau'$ *where* $\tau \Rightarrow \tau'$*, such that* $\Gamma; C \vdash E' \rhd \tau'$*.*

*Proof.* By induction over the derivation of $E \rightarrow^\star E'$.

**Case**

$$(\lambda n.E)\ \mathrm{n} \longrightarrow E\{\mathrm{n}/n\}$$

| | |
|---|---|
| $\Gamma; C \vdash (\lambda n.E)\ \mathrm{n} \rhd \tau$ | Assumption |
| $\Gamma; C \vdash \lambda n.E \rhd \Pi n : \mathtt{T}.\tau$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ | By inversion |
| $\Gamma; C, n : \mathtt{T} \vdash E \rhd \tau$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ | By inversion |
| $\Gamma; C\{\mathrm{n}/n\} \vdash E\{\mathrm{n}/n\} \rhd \tau\{\mathrm{n}/n\}$ | By substitution lemma (1) |

**Case**

$$\mathbf{R}\ S\ \lambda i.\lambda X.R\ 0 \longrightarrow S$$

| | |
|---|---|
| $\Gamma; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X.R\ 0 \rhd \Delta\ 0$ | Assumption |
| $\Gamma; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X.R \rhd \Pi i : \mathtt{I}.\Delta$ and $C \vdash 0$ | By inversion |
| $\Gamma; C \vdash S \rhd \Delta\ 0$ | By inversion |

**Case**

$$\mathbf{R}\ S\ \lambda i.\lambda X.R\ \mathrm{n}+1 \longrightarrow R\{\mathrm{n}/i\}\{\mathbf{R}\ S\ \lambda i.\lambda X.R\ \mathrm{n}/X\}$$

| | |
|---|---|
| $\Gamma; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X.R\ \mathrm{n}+1 \rhd \Delta\ \mathrm{n}+1$ | Assumption |
| $\Gamma; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X.R \rhd \Pi i : \mathtt{I}.\Delta$ and $C \vdash \mathrm{n}+1$ n+1 = $\max(\mathtt{I})$ | By inversion |
| $\Gamma; C \vdash S \rhd \Delta\ 0$ and $\Gamma, X : \Delta\ i; C, i : \mathtt{I} \vdash R \rhd \Delta\ i+1$ | By inversion |
| $\Gamma, X : \Delta\ \mathrm{n}; C \vdash Q\{\mathrm{n}/i\} \rhd \Delta\mathrm{n}+1$ | By substitution lemma (2) |
| $\Gamma; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X.R\ \mathrm{n} \rhd \Delta\ \mathrm{n}$ | By induction hyp. |
| $\Gamma; C \vdash R\{\mathrm{n}/i\}\{\mathbf{R}\ S\ \lambda i.\lambda X.R\ \mathrm{n}/X\} \rhd \Delta\mathrm{n}+1$ | By substitution lemma (3) |

**Case**

$$\bar{a}[\hat{\mathrm{p}}_0..\hat{\mathrm{p}}_\mathrm{n}](y_0).R_0 \mid a[\hat{\mathrm{p}}_1](y_1).R_1 \mid ...a[\hat{\mathrm{p}}_\mathrm{n}](y_\mathrm{n}).R_\mathrm{n} \longrightarrow (\nu s)(R_0\{s[\hat{\mathrm{p}}_0]/y_0\} \mid ... \mid R_\mathrm{n}\{s[\hat{\mathrm{p}}_\mathrm{n}]/y_\mathrm{n}\} \mid s : \emptyset)$$

| | |
|---|---|
| $\Gamma; C \vdash \bar{a}[\hat{\mathrm{p}}_0..\hat{\mathrm{p}}_\mathrm{n}](y_0).R_0 \mid a[\hat{\mathrm{p}}_1](y_1).R_1 \mid ...a[\hat{\mathrm{p}}_\mathrm{n}](y_\mathrm{n}).R_\mathrm{n} \rhd \Delta$ | Assumption |
| $\Gamma; C \vdash \bar{a}[\hat{\mathrm{p}}_0..\hat{\mathrm{p}}_\mathrm{n}](y_0).R_0 \rhd \Delta_1$, ..., $\Gamma; C \vdash a[\hat{\mathrm{p}}_\mathrm{n}](y_\mathrm{n}).R_\mathrm{n} \rhd \Delta_{\mathrm{n}+1}$, where $\Delta = \Delta_1, ..., \Delta_{n+1}$ | By inversion |
| $\Gamma; C \vdash R_0 \rhd \Delta_1, y : \xi(G \upharpoonright \hat{\mathrm{p}}_0)$ | By inversion |
| $\Gamma; C \vdash R_0\{s[\hat{\mathrm{p}}_0]/y_0\} \rhd \Delta_1, s[\hat{\mathrm{p}}_0] : \xi(G \upharpoonright \hat{\mathrm{p}}_0)$ | By substitution lemma (4) |
| ... | |
| $\Gamma; C \vdash R_\mathrm{n}\{s[\hat{\mathrm{p}}_\mathrm{n}]/y_\mathrm{n}\} \rhd \Delta_{\mathrm{n}+1}, s[\hat{\mathrm{p}}_0] : \xi(G \upharpoonright \hat{\mathrm{p}}_0)$ | By substitution lemma (4) |
| $\Gamma; C \vdash R_0\{s[\hat{\mathrm{p}}_0]/y_0\} \mid ... \mid R_\mathrm{n}\{s[\hat{\mathrm{p}}_\mathrm{n}]/y_\mathrm{n}\} \rhd$ $\Delta_1, ..., \Delta_{\mathrm{n}+1}, s[\hat{\mathrm{p}}_0] : \xi(G \upharpoonright \hat{\mathrm{p}}_0), ..., s[\hat{\mathrm{p}}_0] : \xi(G \upharpoonright \hat{\mathrm{p}}_0)$ | By rule |
| $\Gamma; C \vdash (\nu s)(R_0\{s[\hat{\mathrm{p}}_0]/y_0\} \mid ... \mid R_\mathrm{n}\{s[\hat{\mathrm{p}}_\mathrm{n}]/y_\mathrm{n}\} \mid s : \emptyset) \rhd \Delta$ | By rule |

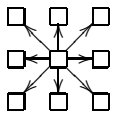
Other cases are the same as in [4].

# D Examples

## D.1 Star Communication Pattern

The Star pattern consists of n+1 workers (named by W) and that every worker W[i] ($1 \leq i \leq n$) is connected to W[0]. The Start has two distinct roles: Center, represented by W[0] and Worker, represented by W[i]. The global type specifies that the first message for j=n?1 is sent by W[n] to W[0], and the last one for n=0 is sent by W[1] to W[0]. In this pattern, we observe the use of the $\mathbf{R}$ operator to define also repetitive end-point behavior, in the Center role. Below, we provide the main program and roles of the Star:

**Star**



$\mathbf{R}$ end
$\lambda j.\lambda \mathbf{y}.\mathtt{W}[0] \rightarrow \mathtt{W}[j+1] : \langle U \rangle.\mathbf{y}$
$n+1$

$\mathtt{def}\ \ \mathtt{W} = \mathbf{R}\ \mathtt{W}[n]\ \lambda i.\lambda X.(\mathtt{W}[i+2], X)\ n-2$
*Center* $\triangleq \bar{a}[\mathtt{W}[0], \mathtt{W}[1], \mathtt{W}](y).\mathbf{R}$ end $\lambda j.\lambda Y.y!\langle \mathtt{W}[j+1], f(j+1)\rangle; Y\ n$
*Worker(i)* $\triangleq a[\mathtt{W}[i+1]](y).y?(\mathtt{W}[0], z); P$
Start $\triangleq \lambda n.(\mathbf{R}$ *Center* $\lambda i.\lambda X.($*Worker(i)* $\mid X) \quad n)$

## D.2 Jacobi Solution of the Poisson Equation

$$\lambda n.\lambda m.\mathbf{R}\ P_{\text{start}}(n,m) \mid P_{\text{RBC}}(n) \mid P_{\text{LTC}}(m) \mid P_{\text{LBC}}$$
$$\lambda i.\lambda X.\mathbf{R}\ (P_{\text{LC}}(i) \mid P_{\text{RC}}(i) \mid X)\lambda j.\lambda Y.(P_{\text{centre}}(i,j) \mid P_{\text{TR}}(j) \mid P_{\text{BR}}(j) \mid Y)\ n\ m$$

$$\begin{aligned}
P_{\text{centre}}(i,j) \ = a[\mathtt{W}[i][j]](y).\mu\mathbf{t}.y\&\langle \mathtt{W}[i][j+1], \{true : y \oplus \langle \mathtt{W}[i][j-1]\{true :\\
& y?(\mathtt{W}[i+1][j], z_1); y!\langle \mathtt{W}[i+1][j], f(i+1,j)\rangle;\\
& y?(\mathtt{W}[i][j+1], z_2); y!\langle \mathtt{W}[i][j+1], f(i,j+1)\rangle;\\
& y?(\mathtt{W}[i][j-1], z_3); y!\langle \mathtt{W}[i][j-1], f(i,j-1)\rangle;\\
& y?(\mathtt{W}[i-1][j], z_4); y!\langle \mathtt{W}[i-1][j], f(i-1,j)\rangle;\\
& y?(\mathtt{W}[i][j-1], z_5); y!\langle \mathtt{W}[i][j+1], f'(z_5)\rangle; \mathbf{t}\},\\
& \qquad false : y \oplus \langle \mathtt{W}[i][j-1], \{false :\\
& y?(\mathtt{W}[i][j-1], z_6); y!\langle \mathtt{W}[i][j+1], f''(z_6)\rangle; \mathbf{0}\}\}\rangle\\
P_{\text{start}}(n,m) = \bar{a}[\mathtt{W}[0][0]..\mathtt{W}[n][m]](y).\mu\mathbf{t}.y \oplus \langle \mathtt{W}[n][m-1], \mathtt{W}[n-1][m]\{true :\\
& y!\langle \mathtt{W}[n][m-1], f(n,m-1)\rangle; y?(\mathtt{W}[n][m-1], z);\\
& y!\langle \mathtt{W}[n-1][m], f(n-1,m)\rangle; y?(\mathtt{W}[n-1][m], z_2);\\
& y?(\mathtt{W}[n-1][m], z_3); y?(\mathtt{W}[n][m-1], z_4)); \mathbf{t},\\
& \qquad false : y?(\mathtt{W}[n-1][m], z_5); y?(\mathtt{W}[n][m-1], z_6); \mathbf{0}\}\rangle\\
P_{\text{LBC}}(m) \quad = a[\mathtt{W}[0][m]](y).\mu\mathbf{t}.y\&\langle \mathtt{W}[1][m], \{true : y \oplus \langle \mathtt{W}[0][m-1]\{true :\\
& y?(\mathtt{W}[1][m], z_1); y!\langle \mathtt{W}[1][m], f(i+1,j)\rangle;\\
& y!\langle \mathtt{W}[0][m-1], f(0,1)\rangle; y?(\mathtt{W}[0][m-1], z_2);\\
& y?(\mathtt{W}[0][m-1], z_5); y!\langle \mathtt{W}[1][m], f'(z_5)\rangle; \mathbf{t}\},\\
& \qquad false : y \oplus \langle \mathtt{W}[1][m], \{false :\\
& y?(\mathtt{W}[0]m-1], z_6); y!\langle \mathtt{W}[1][m], f''(z_6)\rangle; \mathbf{0}\}\rangle\}\rangle
\end{aligned}$$

$$
\begin{aligned}
P_{\text{TR}}(j) = a[\mathtt{W}[n][j]](y).\mu\mathbf{t}.y\&\langle \mathtt{W}[n][j+1], \{true : y \oplus \langle \mathtt{W}[n][j-1]\{true :\\
& y?(\mathtt{W}[n][j+1], z_2); y!\langle \mathtt{W}[n][j+1], f(i, j+1)\rangle;\\
& y?(\mathtt{W}[n][j-1], z_3); y!\langle \mathtt{W}[n][j-1], f(i, j-1)\rangle;\\
& y?(\mathtt{W}[n-1][j], z_4); y!\langle \mathtt{W}[n-1][j], f(i-1, j)\rangle;\\
& y?(\mathtt{W}[n][j-1], z_5); y!\langle \mathtt{W}[n][j+1], f'(z_5)\rangle; \mathbf{t}\}\rangle,\\
& \qquad\qquad\qquad\qquad false : y \oplus \langle \mathtt{W}[n][j-1], \{false :\\
& y?(\mathtt{W}[n][j-1], z_6); y!\langle \mathtt{W}[n][j+1], f''(z_6)\rangle; \mathbf{0}\}\rangle\}\rangle
\end{aligned}
$$

The implementation of the other roles is similar and can be easily created from the definition of $P_{\text{center}}$.