Prioritise the Best Variation

Wen Kokke[1] and Ornela Dardha[2]

University of Edinburgh, Edinburgh wen.kokke@ed.ac.uk
University of Glasgow, Glasgow ornela.dardha@glasgow.ac.uk

Abstract. Session types are used to specify and verify communication protocols and have been integrated in the π-calculus [21,37,22] and a linear concurrent λ-calculus—Good Variation (GV) [44,30], among others. Session type systems guarantee communication safety and session fidelity, but cannot guarantee deadlock freedom and avoid bad interleaving of different sessions. In Classical Processes (CP) [43], a process calculus based on classical linear logic, deadlock freedom is guaranteed by combining channel creation and parallel composition under the same logical cut rule. Similarly, deadlock freedom in GV is guaranteed by combining channel creation and thread spawning under the same operation, called fork. In both GV and CP, deadlock freedom is achieved at the expense of expressivity as the only communication structures allowed are trees. Dardha and Gay [12] define Priority CP (PCP), which allows for cyclic structures and restores deadlock freedom by adding priorities to types, in line with Kobayashi and Padovani's work [26,34].
Following PCP, we present Priority GV (PGV), a variant of GV which decouples channel creation from thread spawning, and restores deadlock freedom by adding priorities. PGV has strong ties to linear logic and solves several problems in the original GV. We show our type system is sound by proving subject reduction and progress. We define an encoding from PCP to PGV and prove that the encoding preserves typing and is sound and complete with respect to the operational semantics.

1 Introduction

Session types are a type formalism used to specify and verify communication protocols between two or more communicating agents [21,37,22,9]. They have been studied for many programming paradigms, including concurrent and functional paradigms. In particular, they have been defined for the π-calculus [36]—a process calculus for communication and concurrency, and for Good Variation (GV) [43,30]—a linear concurrent λ-calculus. Session type systems guarantee communication safety and session fidelity, namely that communication proceeds without any type mismatch and it follows the predefined protocol specified as a session type. Another key property in a concurrent setting is deadlock freedom, stating that agents communicate without getting stuck in cyclic dependencies. Deadlock freedom is a more involving property and session types alone are not enough to rule out deadlocks in cyclic structures of multiple interleaved sessions. Several techniques have been developed to address deadlock freedom in the π-calculus and concurrent λ-calculus with session types, as detailed below.

With regards to π-calculus, a growing line of work on deadlock freedom leverages the Curry-Howard correspondences between intuitionistic or classical linear logic and π-calculus with session types [6,43], which guarantee deadlock freedom by design. This is achieved by merging constructs for parallel composition and channel restriction under the logical cut rule, which forces processes to share only one channel for communication and eliminates cyclic dependencies. Recent developments in the Classical Process (CP) [43] line of work have led to various approaches to decoupling these constructs, either by maintaining a strong correspondence to classical linear logic, as in Hypersequent CP [28,29], or by weakening the correspondence to classical linear logic in exchange for a more expressive language, as in Priority CP (PCP) [12]. PCP decouples CP's cut rule into two separate constructs: a construct for parallel composition, by introducing a mix rule; and a construct for restriction, by introducing a cycle rule. Differently from previous work, PCP allows cyclic structures and restores deadlock freedom by adding priorities to types following Kobayashi [26] and Padovani [34]. Priorities establish an order of actions in a π-calculus process and are used to rule out bad interleaving of channels, guaranteeing deadlock freedom.

With regards to GV, the original work on session types in GV [18,19] did not satisfy deadlock freedom. Later on, calculi in the GV family [43,30] have achieved deadlock freedom via a syntactic restriction, *i.e.*, by combining channel creation and thread spawning into a single operation, called fork. The fork construct is the term representation of the cut construct in CP, which as mentioned above, evaluates to a channel restriction combined with a parallel composition. As with CP, this is restrictive because it limits communication structures to only trees.

We aim to further investigate GV and deadlock freedom, as there are several benefits of working with a functional language as opposed to name-passing calculi. Functional languages support higher-order functions, and have a capability for abstraction not usually present in process calculi. Working within a functional language allows to derive extensions of the communication capabilities of the language via well-understood extensions of the functional fragment, *i.e.*, deriving internal/external choice via sum types. Finally, in concurrent functional calculi, there is a clear separation between a program that the user writes, and a configuration, which is the state of a system as it evaluates. In process calculi, these are conflated. Moreover, the benefit of working with GV over other session-typed functional languages is that GV has strong ties to linear logic, via its relation to CP [43], and consequently it has strong formal properties, *e.g.*, deadlock freedom, albeit in a restrictive way. Following the above observations, we thus pose our research question.

RQ: *Can we design a more expressive GV where deadlock freedom is guaranteed by design and communication structures are not limited to only trees?*

We address this research question by following the line of work from CP to Priority CP, and present Priority GV (PGV), a variant of GV which decouples channel creation from thread spawning and restores deadlock freedom by adding priorities. This work closes the circle of the strong connection of CP and GV, together with their priority-based versions, PCP and PGV.

In this paper we make the following contributions:

1. **Priority GV**. We present Priority GV (PGV) in section 2, and prove subject reduction (theorem 1) and progress (theorem 2). PGV is the *first* session-typed functional language with priorities *and* strong ties to linear logic via its correspondence with PCP and its relation to the CP and GV languages.
2. **Solving GV problems**. PGV addresses several problems in the original GV language, most notably: (a) our version does not have the pseudo-type $S^\sharp$; (b) our structural congruence is type preserving.
3. **Updated Priority CP.** We present an updated version of Priority CP [12] in section 3. We remove commuting conversions and move away from reduction as cut elimination, towards reduction based on structural congruence, as it is standard in process calculi.
4. **Connection to linear logic**. The connection of PGV to linear logic is given via a *sound and complete encoding* of PCP to PGV in section 3, where we prove the encoding preserves typing (theorem 4) and satisfies operational correspondence (theorems 5 and 6). We recall Milner's cyclic scheduler [32] presented in PCP [12] and encode it in PGV.

2 Priority GV

We present Priority GV (PGV), a session-typed functional language based on GV [44,30] which uses priorities à la Kobayashi and Padovani [26,35] to enforce deadlock freedom. Priority GV offers a more fine-grained analysis of communication structures, and by separating channel creation form thread spawning it allows cyclic structures. We illustrate this with two programs in PGV, examples 1 and 2. Each program contains two processes—the main process, and the child process created by **spawn**—which communicate using *two* channels. The child process receives a unit over the channel x/x', and then sends a unit over the channel y/y'. The main process does one of two things: (a) in example 1, it sends a unit over the channel x/x', and then waits to receive a unit over the channel y/y'; (b) in example 2, it does these in the opposite order, which results in a deadlock. We will show that only example 1 is typeable in PGV, while it is not typable in GV [43] or its predecessor [18].

Example 1 (Cyclic Structure).

$$\begin{array}{l}\textbf{let } (x,x') = \textbf{new in}\\ \textbf{let } (y,y') = \textbf{new in}\\ \textbf{spawn}\left(\begin{array}{l}\textbf{let } ((),x') = \textbf{recv } x' \textbf{ in}\\ \textbf{let } y = \textbf{send } ((),y) \textbf{ in}\\ \textbf{wait } x';\textbf{close } y\end{array}\right);\\ \underline{\textbf{let } x = \textbf{send } ((),x) \textbf{ in}}\\ \underline{\textbf{let } ((),y') = \textbf{recv } y' \textbf{ in}}\\ \textbf{close } x;\textbf{wait } y'\end{array}$$

Example 2 (Deadlock).

$$\begin{array}{l}\textbf{let } (x,x') = \textbf{new in}\\ \textbf{let } (y,y') = \textbf{new in}\\ \textbf{spawn}\left(\begin{array}{l}\textbf{let } ((),x') = \textbf{recv } x' \textbf{ in}\\ \textbf{let } y = \textbf{send } ((),y) \textbf{ in}\\ \textbf{wait } x';\textbf{close } y\end{array}\right);\\ \underline{\textbf{let } ((),y') = \textbf{recv } y' \textbf{ in}}\\ \underline{\textbf{let } x = \textbf{send } ((),x) \textbf{ in}}\\ \textbf{close } x;\textbf{wait } y'\end{array}$$

Session types. Session types (S) are defined by the following grammar:

$$S ::= !^o T.S \mid ?^o T.S \mid \mathbf{end}_!^o \mid \mathbf{end}_?^o$$

Session types $!^o T.S$ and $?^o T.S$ describe the endpoints of a channel over which we send or receive a value of type T, and then proceed as S. Types $\mathbf{end}_!^o$ and $\mathbf{end}_?^o$ describe endpoints of a channel whose communication has finished, and over which we must synchronise before closing the channel. Each connective in a session type is annotated with a *priority* $o \in \mathbb{N}$.

Types. Types (T, U) are defined by the following grammar:

$$T, U ::= T \times U \mid \mathbf{1} \mid T + U \mid \mathbf{0} \mid T \multimap^{p,q} U \mid S$$

Types $T \times U$, $\mathbf{1}$, $T + U$, and $\mathbf{0}$ are the standard linear λ-calculus product type, unit type, sum type, and empty type. Type $T \multimap^{p,q} U$ is the standard linear function type, annotated with *priority bounds* $p, q \in \mathbb{N} \cup \{\bot, \top\}$. Every session type is also a type. Given a function with type $T \multimap^{p,q} U$, p is a *lower bound* on the priorities of the endpoints captured by the body of the function, and q is an *upper bound* on the priority of the communications that take place as a result of applying the function. The type of *pure functions* $T \multimap U$, *i.e.*, those which perform no communications, is syntactic sugar for $T \multimap^{\top,\bot} U$.

Environments. Typing environments Γ, Δ associate types to names. Environments are linear, so two environments can only be combined as Γ, Δ if their names are distinct, *i.e.*, $\mathrm{fv}(\Gamma) \cap \mathrm{fv}(\Delta) = \varnothing$.

$$\Gamma, \Delta ::= \varnothing \mid \Gamma, x : T$$

Duality. Duality plays a crucial role in session types. The two endpoints of a channel are assigned dual types, ensuring that, for instance, whenever one program *sends* a value on a channel, the program on the other end is waiting to *receive*. Each session type S has a dual, written $\overline{S}$. Duality is an involutive function which *preserves priorities*:

$$\overline{!^o T.S} = ?^o T.\overline{S} \qquad \overline{?^o T.S} = !^o T.\overline{S} \qquad \overline{\mathbf{end}_!^o} = \overline{\mathbf{end}_?^o} \qquad \overline{\mathbf{end}_?^o} = \overline{\mathbf{end}_!^o}$$

Priorities. Function $\mathrm{pr}(\cdot)$ returns the smallest priority of a session type. The type system guarantees that the top-most connective always holds the smallest priority, so we simply return the priority of the top-most connective:

$$\mathrm{pr}(!^o T.S) = o \qquad \mathrm{pr}(?^o T.S) = o \qquad \mathrm{pr}(\mathbf{end}_!^o) = o \qquad \mathrm{pr}(\mathbf{end}_?^o) = o$$

We extend the function $\mathrm{pr}(\cdot)$ to types and typing contexts by returning the smallest priority in the type or context, or $\top$ if there is no priority. We use $\sqcap$ and $\sqcup$ to denote the minimum and maximum:

$$\begin{array}{lll}
\min_{\mathrm{pr}}(T \times U) & = \min_{\mathrm{pr}}(T) \sqcap \min_{\mathrm{pr}}(U) & \min_{\mathrm{pr}}(\mathbf{1}) = \top \\
\min_{\mathrm{pr}}(T + U) & = \min_{\mathrm{pr}}(T) \sqcap \min_{\mathrm{pr}}(U) & \min_{\mathrm{pr}}(\mathbf{0}) = \top \\
\min_{\mathrm{pr}}(T \multimap^{p,q} U) & = p & \min_{\mathrm{pr}}(S) = \mathrm{pr}(S) \\
\min_{\mathrm{pr}}(\Gamma, x : A) & = \min_{\mathrm{pr}}(\Gamma) \sqcap \min_{\mathrm{pr}}(A) & \min_{\mathrm{pr}}(\varnothing) = \top
\end{array}$$

Terms. Terms (L, M, N) are defined by the following grammar:

$$\begin{array}{rcl} L, M, N & ::= & x \mid K \mid \lambda x.M \mid M\ N \\ & & \mid () \mid M; N \mid (M, N) \mid \mathbf{let}\ (x, y) = M\ \mathbf{in}\ N \\ & & \mid \mathbf{inl}\ M \mid \mathbf{inr}\ M \mid \mathbf{case}\ L\ \{\mathbf{inl}\ x \mapsto M;\ \mathbf{inr}\ y \mapsto N\} \mid \mathbf{absurd}\ M \\ K & ::= & \mathbf{link} \mid \mathbf{new} \mid \mathbf{spawn} \mid \mathbf{send} \mid \mathbf{recv} \mid \mathbf{close} \mid \mathbf{wait} \end{array}$$

Let x, y, z, and w range over variable names. Occasionally, we use a, b, c, and d. The term language is the standard linear λ-calculus with products, sums, and their units, extended with constants K for the communication primitives.

Constants are best understood in conjunction with their typing and reduction rules in figs. 1 and 2. Briefly, **link** links two endpoints together, forwarding messages from one to the other, **new** creates a new channel and returns a pair of its endpoints, and **spawn** spawns off its argument as a new thread. The **send** and **recv** functions send and receive values on a channel. However, since the typing rules for PGV ensure the linear usage of endpoints, they also return a new copy of the endpoint to continue the session. The **close** and **wait** functions close a channel. We use syntactic sugar to make terms more readable: we write $\mathbf{let}\ x = M\ \mathbf{in}\ N$ in place of $(\lambda x.N)\ M$, $\lambda().M$ in place of $\lambda z.z; M$, and $\lambda(x, y).M$ in place of $\lambda z.\mathbf{let}\ (x, y) = z\ \mathbf{in}\ M$. We recover **fork** as $\lambda x.\mathbf{let}\ (y, z) = \mathbf{new}\ ()\ \mathbf{in}\ \mathbf{spawn}\ (\lambda().x\ y); z$.

Internal and External Choice. Typically, session-typed languages feature constructs for internal and external choice. In GV, these can be defined in terms of the core language, by sending or receiving a value of a sum type [30]. We use the following syntactic sugar for internal ($S \oplus^o S'$) and external ($S \mathbin{\&}^o S'$) choice and their units:

$$\begin{array}{ll} S \oplus^o S' \triangleq\ !^o(\overline{S} + \overline{S'}).\mathbf{end}_!^{o+1} & \qquad \oplus^o\{\} \triangleq\ !^o\mathbf{0}.\mathbf{end}_!^{o+1} \\ S \mathbin{\&}^o S' \triangleq\ ?^o(S + S').\mathbf{end}_?^{o+1} & \qquad \mathbin{\&}^o\{\} \triangleq\ ?^o\mathbf{0}.\mathbf{end}_?^{o+1} \end{array}$$

As the syntax for units suggests, these are the binary and nullary forms of the more common n-ary choice constructs $\oplus^o\{l_i : S_i\}_{i \in I}$ and $\mathbin{\&}^o\{l_i : S_i\}_{i \in I}$, which one may obtain generalising the sum types to variant types. For simplicity, we present only the binary and nullary forms.

Similarly, we use syntactic sugar for the term forms of choice, which combine sending and receiving with the introduction and elimination forms for the sum and empty types. There are two constructs for binary internal choice, expressed using the meta-variable ℓ which ranges over $\{\mathbf{inl}, \mathbf{inr}\}$. As there is no introduction for the empty type, there is no construct for nullary internal choice:

$$\begin{array}{l} \mathbf{select}\ \ell \triangleq \\ \quad \lambda x.\mathbf{let}\ (y, z) = \mathbf{new}\ \mathbf{in}\ \mathbf{close}\ (\mathbf{send}\ (\ell\ y, x)); z \\ \mathbf{offer}\ L\ \{\mathbf{inl}\ x \mapsto M; \mathbf{inr}\ y \mapsto N\} \triangleq \\ \quad \mathbf{let}\ (z, w) = \mathbf{recv}\ L\ \mathbf{in}\ \mathbf{wait}\ w; \mathbf{case}\ z\ \{\mathbf{inl}\ x \mapsto M;\ \mathbf{inr}\ y \mapsto N\} \\ \mathbf{offer}\ L\ \{\} \triangleq \\ \quad \mathbf{let}\ (z, w) = \mathbf{recv}\ L\ \mathbf{in}\ \mathbf{wait}\ w; \mathbf{absurd}\ z \end{array}$$

Operational Semantics. Priority GV terms are evaluated as part of a configuration of processes. Configurations are defined by the following grammar:

$$\begin{array}{lll} \phi & ::= & \bullet \mid \circ \\ \mathcal{C}, \mathcal{D}, \mathcal{E} & ::= & \phi\ M \mid \mathcal{C} \parallel \mathcal{D} \mid (\nu x x')\mathcal{C} \end{array}$$

Configurations ($\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$) consist of threads $\phi\ M$, parallel compositions $\mathcal{C} \parallel \mathcal{D}$, and name restrictions $(\nu x x')\mathcal{C}$. To preserve the functional nature of PGV, where programs return a single value, we use flags (ϕ) to differentiate between the main thread, marked $\bullet$, and child threads created by **spawn**, marked $\circ$. Only the main thread returns a value. We determine the flag of a configuration by combining the flags of all threads in that configuration:

$$\bullet + \circ = \bullet \quad \circ + \bullet = \bullet \quad \circ + \circ = \circ \quad (\bullet + \bullet \text{ is undefined})$$

Values (V, W), evaluation contexts (E), thread evaluation contexts ($\mathcal{F}$), and configuration contexts ($\mathcal{G}$) are defined by the following grammars:

$$\begin{array}{lll} V, W & ::= & x \mid K \mid \lambda x.M \mid () \mid (V, W) \mid \mathbf{inl}\ V \mid \mathbf{inr}\ V \\ E & ::= & \Box \mid E\ M \mid V\ E \\ & & \mid E; N \mid (E, M) \mid (V, E) \mid \mathbf{let}\ (x, y) = E\ \mathbf{in}\ M \\ & & \mid \mathbf{inl}\ E \mid \mathbf{inr}\ E \mid \mathbf{case}\ E\ \{\mathbf{inl}\ x \mapsto M;\ \mathbf{inr}\ y \mapsto N\} \mid \mathbf{absurd}\ E \\ \mathcal{F} & ::= & \phi\ E \\ \mathcal{G} & ::= & \Box \mid \mathcal{G} \parallel \mathcal{C} \mid (\nu x y)\mathcal{G} \end{array}$$

We factor the reduction relation of PGV into a deterministic reduction on terms ($\longrightarrow_M$) and a non-deterministic reduction on configurations ($\longrightarrow_{\mathcal{C}}$), see fig. 1. We write $\longrightarrow_M^+$ and $\longrightarrow_{\mathcal{C}}^+$ for the transitive closures, and $\longrightarrow_M^\star$ and $\longrightarrow_{\mathcal{C}}^\star$ for the reflexive-transitive closures.

Term reduction is the standard call-by-value, left-to-right evaluation for GV, and only deviates from reduction for the linear λ-calculus in that it reduces terms to values *or* ready terms waiting to perform a communication action.

Configuration reduction resembles evaluation for a process calculus: E-Link, E-Send, and E-Close perform communications, E-LiftC allows reduction under configuration contexts, and E-LiftSC embeds a structural congruence $\equiv$. The remaining rules mediate between the process calculus and the functional language: E-New and E-Spawn evaluate the **new** and **spawn** constructs, creating the equivalent configuration constructs, and E-LiftM embeds term reduction.

Structural congruence satisfies the following axioms: SC-LinkSwap allows swapping channels in the link process. SC-ResLink allows restriction to applied to link which is structurally equivalent to the terminated process, thus allowing elimination of unnecessary restrictions. SC-ResSwap allows swapping channels and SC-ResComm states that restriction is commutative. SC-ResExt is the standard scope extrusion rule. Rules SC-ParNil, SC-ParComm and SC-ParAssoc state that parallel composition uses the terminated process as the neutral element; it is commutative and associative.

While our configuration reduction is based on the standard evaluation for GV, the increased expressiveness of PGV allows us to simplify the relation on

Term reduction.

$$\begin{array}{lll}
\text{E-Lam} & (\lambda x.M)\ V & \longrightarrow_M M\{V/x\} \\
\text{E-Unit} & \mathbf{let}\ () = ()\ \mathbf{in}\ M & \longrightarrow_M M \\
\text{E-Pair} & \mathbf{let}\ (x,y) = (V,W)\ \mathbf{in}\ M & \longrightarrow_M M\{V/x\}\{W/y\} \\
\text{E-Inl} & \mathbf{case}\ \mathbf{inl}\ V\ \{\mathbf{inl}\ x \mapsto M;\ \mathbf{inr}\ y \mapsto N\} & \longrightarrow_M M\{V/x\} \\
\text{E-Inr} & \mathbf{case}\ \mathbf{inr}\ V\ \{\mathbf{inl}\ x \mapsto M;\ \mathbf{inr}\ y \mapsto N\} & \longrightarrow_M N\{V/y\}
\end{array}$$

$$\text{E-Lift}\ \frac{M \longrightarrow_M M'}{E[M] \longrightarrow_M E[M']}$$

Structural congruence.

$$\begin{array}{lll}
\text{SC-LinkSwap} & \mathcal{F}[\mathbf{link}\ (x,y)] & \equiv \mathcal{F}[\mathbf{link}\ (y,x)] \\
\text{SC-ResLink} & (\nu xy)(\phi\ \mathbf{link}\ (x,y)) & \equiv \phi\ () \\
\text{SC-ResSwap} & (\nu xy)\mathcal{C} & \equiv (\nu yx)\mathcal{C} \\
\text{SC-ResComm} & (\nu xy)(\nu zw)\mathcal{C} & \equiv (\nu zw)(\nu xy)\mathcal{C},\ \text{if}\ \{x,y\} \cap \{z,w\} = \varnothing \\
\text{SC-ResExt} & (\nu xy)(\mathcal{C} \parallel \mathcal{D}) & \equiv \mathcal{C} \parallel (\nu xy)\mathcal{D},\ \text{if}\ x,y \notin \text{fv}(\mathcal{C}) \\
\text{SC-ParNil} & \mathcal{C} \parallel \circ() & \equiv \mathcal{C} \\
\text{SC-ParComm} & \mathcal{C} \parallel \mathcal{D} & \equiv \mathcal{D} \parallel \mathcal{C} \\
\text{SC-ParAssoc} & \mathcal{C} \parallel (\mathcal{D} \parallel \mathcal{E}) & \equiv (\mathcal{C} \parallel \mathcal{D}) \parallel \mathcal{E}
\end{array}$$

Configuration reduction.

$$\begin{array}{lll}
\text{E-Link} & (\nu xy)(\mathcal{F}[\mathbf{link}\ (w,x)] \parallel \mathcal{C}) & \longrightarrow_{\mathcal{C}} \mathcal{F}[()] \parallel \mathcal{C}\{w/y\} \\
\text{E-New} & \mathcal{F}[\mathbf{new}\ ()] & \longrightarrow_{\mathcal{C}} (\nu xy)(\mathcal{F}[(x,y)]),\ \text{if}\ x,y \notin \text{fv}(\mathcal{F}) \\
\text{E-Spawn} & \mathcal{F}[(\mathbf{spawn}\ V)] & \longrightarrow_{\mathcal{C}} \mathcal{F}[()] \parallel \circ\ V\ () \\
\text{E-Send} & (\nu xy)(\mathcal{F}[\mathbf{send}\ (V,x)] \parallel \mathcal{F}'[\mathbf{recv}\ y]) & \longrightarrow_{\mathcal{C}} (\nu xy)(\mathcal{F}[x] \parallel \mathcal{F}'[(V,y)]) \\
\text{E-Close} & (\nu xy)(\mathcal{F}[\mathbf{wait}\ x] \parallel \mathcal{F}'[\mathbf{close}\ y]) & \longrightarrow_{\mathcal{C}} \mathcal{F}[()] \parallel \mathcal{F}'[()]
\end{array}$$

$$\text{E-LiftC}\ \frac{\mathcal{C} \longrightarrow_{\mathcal{C}} \mathcal{C}'}{\mathcal{G}[\mathcal{C}] \longrightarrow_{\mathcal{C}} \mathcal{G}[\mathcal{C}']} \qquad \text{E-LiftM}\ \frac{M \longrightarrow_M M'}{\mathcal{F}[M] \longrightarrow_M \mathcal{F}[M']} \qquad \text{E-LiftSC}\ \frac{\mathcal{C} \equiv \mathcal{C}' \quad \mathcal{C}' \longrightarrow_{\mathcal{C}} \mathcal{D}' \quad \mathcal{D}' \equiv \mathcal{D}}{\mathcal{C} \longrightarrow_{\mathcal{C}} \mathcal{D}}$$

Fig. 1. Operational Semantics for PGV.

two counts. (a) *We decompose the* **fork** *construct.* In GV, **fork** creates a new channel, spawns a child thread, and, when the child thread finishes, it closes the channel to its parent. In PGV, these are three separate operations: **new**, **spawn**, and **close**. We no longer require that every child thread finishes by returning a terminated channel. Consequently, we also simplify the evaluation of the **link** construct. Intuitively, evaluating **link** causes a substitution: if we have a channel bound as (νxy), then **link** (w,x) replaces all occurrences of y by w. However, in GV, **link** is required to return a terminated channel, which means that the semantics for *link* must *create* a fresh channel of type $\mathbf{end}_!/\mathbf{end}_?$. The endpoint of type $\mathbf{end}_!$ is returned by the *link* construct, and a **wait** on the other

endpoint guards the *actual* substitution. In PGV, evaluating **link** simply causes a substitution. (b) *Our structural congruence is type preserving.* Consequently, we can embed it directly into the reduction relation. In GV, this is not the case, and subject reduction relies on proving that if $\equiv\!\!\longrightarrow_{\mathcal{C}}$ ends up in an ill-typed configuration, we can rewrite it to a well-typed configuration using $\equiv$.

Typing. Figure 2 gives the typing rules for PGV. Typing rules for terms are at the top of fig. 2. Terms are typed by a judgement $\Gamma \vdash^{p} M : T$ stating that "a term M has type T and an upper bound on its priority p under the typing environment Γ". Typing for the linear λ-calculus is standard. Linearity is ensured by splitting environments on branching rules, requiring that the environment in the variable rule consists of just the variable, and the environment in the constant and unit rules are empty. Constants K are typed using type schemas, and embedded using T-Const (mid of fig. 2). The typing rules treat *all variables* as linear resources, even those of non-linear types such as $\mathbf{1}$. However, the rules can easily be extended to allow values with unrestricted usage [43].

The only non-standard feature of the typing rules is the priority annotations. Priorities are based on *obligations/capabilities* used by Kobayashi [26], and simplified to single priorities following Padovani [34]. The integration of priorities into GV is adapted from Padovani and Novara [35]. Paraphrasing Dardha and Gay [12], priorities obey the following two laws: (i) an action with lower priority happens before an action with higher priority; and (ii) communication requires *equal* priorities for dual actions.

In PGV, we keep track of a lower and upper bound on the priorities of a term, *i.e.*, while evaluating the term, when does it start communicating, and when does it finish. The upper bound is written on the sequent, whereas the lower bound is approximated from the typing environment. Typing rules for sequential constructs enforce sequentially, *e.g.*, the typing for $M; N$ has a side condition which requires that the upper bound of M is smaller than the lower bound of N, *i.e.*, M finishes before N starts. The typing rule for **new** ensures that both endpoints of a channel share the same priorities. Together, these two constraints guarantee deadlock freedom.

To illustrate this, let's go back to the deadlocked program in example 2. Crucially, it composes the terms below in parallel. While each of these terms itself is well-typed, they impose opposite conditions on the priorities, so their composition is ill-typed. (We omit the priorities on $\mathbf{end}_!$ and $\mathbf{end}_?$.)

$$\frac{\begin{array}{c} y' : ?^{o'}\mathbf{1}.\mathbf{end}_? \vdash^{o'} \mathbf{recv}\ y' : \mathbf{1} \times \mathbf{end}_? \\ x : !^{o}\mathbf{1}.\mathbf{end}_!, y' : \mathbf{end}_? \vdash^{p} \mathbf{let}\ x = \mathbf{send}\ ((), x)\ \mathbf{in}\ \ldots : \mathbf{1} \qquad o' < o \end{array}}{x : !^{o}\mathbf{1}.\mathbf{end}_!, y' : ?^{o'}\mathbf{1}.\mathbf{end}_? \vdash^{p} \mathbf{let}\ ((), y') = \mathbf{recv}\ y'\ \mathbf{in}\ \mathbf{let}\ x = \mathbf{send}\ ((), x)\ \mathbf{in}\ \ldots : \mathbf{1}}$$

$$\frac{\begin{array}{c} x' : ?^{o}\mathbf{1}.\mathbf{end}_? \vdash^{o} \mathbf{recv}\ x' : \mathbf{1} \times \mathbf{end}_? \\ y : !^{o'}\mathbf{1}.\mathbf{end}_!, x' : \mathbf{end}_? \vdash^{q} \mathbf{let}\ y = \mathbf{send}\ ((), y)\ \mathbf{in}\ \ldots : \mathbf{1} \qquad o < o' \end{array}}{y : !^{o'}\mathbf{1}.\mathbf{end}_!, x' : ?^{o}\mathbf{1}.\mathbf{end}_? \vdash^{q} \mathbf{let}\ ((), x') = \mathbf{recv}\ x'\ \mathbf{in}\ \mathbf{let}\ y = \mathbf{send}\ ((), y)\ \mathbf{in}\ \ldots : \mathbf{1}}$$

Closures suspend communication, so T-Lam stores the priority bounds of the function body on the function type, and T-App restores them. For instance,

Static Typing Rules.

$$\text{T-Var}\ \frac{}{x : T \vdash^{\perp} x : T} \qquad \text{T-Lam}\ \frac{\Gamma, x : T \vdash^{q} M : U}{\Gamma \vdash^{\perp} \lambda x.M : T \multimap^{\min_{\mathrm{pr}}(\Gamma), q} U} \qquad \text{T-Const}\ \frac{}{\varnothing \vdash^{\perp} K : T}$$

$$\text{T-App}\ \frac{\Gamma \vdash^{p} M : T \multimap^{p', q'} U \qquad \Delta \vdash^{q} N : T \qquad p < \min_{\mathrm{pr}}(\Delta) \qquad q < p'}{\Gamma, \Delta \vdash^{p \sqcup q \sqcup q'} M\ N : U}$$

$$\text{T-Unit}\ \frac{}{\varnothing \vdash^{\perp} () : \mathbf{1}} \qquad \text{T-LetUnit}\ \frac{\Gamma \vdash^{p} M : \mathbf{1} \qquad \Delta \vdash^{q} N : T \qquad p < \min_{\mathrm{pr}}(\Delta)}{\Gamma, \Delta \vdash^{p \sqcup q} M; N : T}$$

$$\text{T-Pair}\ \frac{\Gamma \vdash^{p} M : T \qquad \Delta \vdash^{q} N : U \qquad p < \min_{\mathrm{pr}}(\Delta)}{\Gamma, \Delta \vdash^{p \sqcup q} (M, N) : T \times U}$$

$$\text{T-LetPair}\ \frac{\Gamma \vdash^{p} M : T \times T' \qquad \Delta, x : T, y : T' \vdash^{q} N : U \qquad p < \min_{\mathrm{pr}}(\Delta, T, T')}{\Gamma, \Delta \vdash^{p \sqcup q} \mathbf{let}\ (x, y) = M\ \mathbf{in}\ N : U}$$

$$\text{T-Inl}\ \frac{\Gamma \vdash^{p} M : T \qquad \min_{\mathrm{pr}}(T) = \min_{\mathrm{pr}}(U)}{\Gamma \vdash^{p} \mathbf{inl}\ M : T + U}$$

$$\text{T-Inr}\ \frac{\Gamma \vdash^{p} M : U \qquad \min_{\mathrm{pr}}(T) = \min_{\mathrm{pr}}(U)}{\Gamma \vdash^{p} \mathbf{inr}\ M : T + U}$$

$$\text{T-CaseSum}\ \frac{\Gamma \vdash^{p} L : T + T' \qquad \Delta, x : T \vdash^{q} M : U \qquad \Delta, y : T' \vdash^{q} N : U \qquad p < \min_{\mathrm{pr}}(\Delta)}{\Gamma, \Delta \vdash^{p \sqcup q} \mathbf{case}\ L\ \{\mathbf{inl}\ x \mapsto M;\ \mathbf{inr}\ y \mapsto N\} : U}$$

$$\text{T-Absurd}\ \frac{\Gamma \vdash^{p} M : \mathbf{0}}{\Gamma, \Delta \vdash^{p} \mathbf{absurd}\ M : T}$$

We write $K : T$ for $\varnothing \vdash^{\perp} K : T$ in typing derivations.

Type Schemas for Constants.

$$\mathbf{link} : S \times \overline{S} \multimap \mathbf{1} \qquad \mathbf{new} : \mathbf{1} \multimap S \times \overline{S} \qquad \mathbf{spawn} : (\mathbf{1} \multimap^{p,q} \mathbf{1}) \multimap \mathbf{1}$$

$$\mathbf{send} : T \times !^{o}T.S \multimap^{\top, o} S \qquad \mathbf{recv} : ?^{o}T.S \multimap^{\top, o} T \times S$$

$$\mathbf{close} : \mathbf{end}_{!}^{o} \multimap^{\top, o} \mathbf{1} \qquad \mathbf{wait} : \mathbf{end}_{?}^{o} \multimap^{\top, o} \mathbf{1}$$

Runtime Typing Rules.

$$\text{T-Main}\ \frac{\Gamma \vdash^{p} M : T}{\Gamma \vdash^{\bullet} \bullet\ M} \qquad \text{T-Child}\ \frac{\Gamma \vdash^{p} M : \mathbf{1}}{\Gamma \vdash^{\circ} \circ\ M} \qquad \text{T-Res}\ \frac{\Gamma, x : S, y : \overline{S} \vdash^{\phi} \mathcal{C}}{\Gamma \vdash^{\phi} (\nu xy)\mathcal{C}} \qquad \text{T-Par}\ \frac{\Gamma \vdash^{\phi} \mathcal{C} \qquad \Delta \vdash^{\phi'} \mathcal{D}}{\Gamma, \Delta \vdash^{\phi + \phi'} \mathcal{C} \parallel \mathcal{D}}$$

Fig. 2. Typing Rules for PGV.

$\lambda x.\mathbf{send}\ (x, y)$ is assigned the type $A \multimap^{o,o} S$, *i.e.*, a function which, when applied, starts and finishes communicating at priority o.

$$\dfrac{\dfrac{}{\mathbf{send} : A \times !^o A.S \multimap^{\top,o} S} \qquad \dfrac{\dfrac{}{x : A \vdash^{\perp} x : A} \qquad \dfrac{}{x : A, y : !^o A.S \vdash^{\perp} y : !^o A.S}}{x : A, y : !^o A.S \vdash^{\perp} (x, y) : A \times !^o A.S}}{x : A, y : !^o A.S \vdash^{o} \mathbf{send}\ (x, y) : S}}{y : !^o A.S \vdash^{\perp} \lambda x.\mathbf{send}\ (x, y) : A \multimap^{o,o} S}$$

Typing rules for configurations are at the bottom of fig. 2. Configurations are typed by a judgement $\Gamma \vdash^{\phi} \mathcal{C}$ stating that "a configuration $\mathcal{C}$ with flag ϕ is well typed under typing environment Γ". Configuration typing is based on the standard typing for GV. Terms are embedded either as main or as child threads. The priority bound from the term typing is discarded, as configurations contain no further blocking actions. Main threads are allowed to return a value, whereas child threads are required to return the unit value. Sequents are annotated with a flag ϕ, which ensure that there is at most one main thread.

While our configuration typing is based on the standard typing for GV, it differs on two counts: (i) *we require that child threads return the unit value*, as opposed to a terminated channel; and (ii) *we simplify typing for parallel composition*. In order to guarantee deadlock freedom, in GV each parallel composition must split *exactly one* channel of the channel pseudo-type $S^\sharp$ into two endpoints of type S and $\overline{S}$. Consequently, associativity of parallel composition does not preserve typing. In PGV, we guarantee deadlock freedom using priorities, which removes the need for the channel pseudo-type $S^\sharp$, and simplifies typing for parallel composition, while restoring type preservation for the structural congruence.

Subject reduction. Unlike with previous versions of GV, structural congruence, term reduction, and configuration reduction are all type preserving.

We must show that substitution preserves priority constraints. For this, we prove lemma 1, which shows that values have finished all their communication, and that any priorities in the type of the value come from the typing environment. The proofs are by structural induction and can be found in appendix A.

Lemma 1. *If $\Gamma \vdash^{p} V : T$, then $p = \perp$, and $min_{\mathrm{pr}}(\Gamma) = min_{\mathrm{pr}}(T)$.*

Lemma 2 (Substitution).
If $\Gamma, x : U' \vdash^{p} M : T$ and $\Theta \vdash^{q} V : U'$, then $\Gamma, \Theta \vdash^{p} M\{V/x\} : T$.

Lemma 3 (Subject Reduction, $\longrightarrow_M$).
If $\Gamma \vdash^{p} M : T$ and $M \longrightarrow_M M'$, then $\Gamma \vdash^{p} M' : T$.

Lemma 4 (Subject Congruence, $\equiv$).
If $\Gamma \vdash^{\phi} \mathcal{C}$ and $\mathcal{C} \equiv \mathcal{C}'$, then $\Gamma \vdash^{\phi} \mathcal{C}'$.

Theorem 1 (Subject Reduction, $\longrightarrow_{\mathcal{C}}$).
If $\Gamma \vdash^{\phi} \mathcal{C}$ and $\mathcal{C} \longrightarrow_{\mathcal{C}} \mathcal{C}'$, then $\Gamma \vdash^{\phi} \mathcal{C}'$.

Progress and deadlock freedom. PGV satisfies progress, as PGV configurations either reduce or are in normal form. However, the normal forms may

seem surprising at first, as evaluating a well-typed PGV term does not necessarily produce *just* a value. If a term returns an endpoint, then its normal form contains a thread which is ready to communicate on the dual of that endpoint. This behaviour is not new to PGV. Let us consider an example, adapted from Lindley and Morris [30], in which a term returns an endpoint linked to an echo server. The echo server receives a value and sends it back unchanged:

$$\mathbf{echo}_x \triangleq \mathbf{let}\ (y, x) = \mathbf{recv}\ x\ \mathbf{in}\ \mathbf{let}\ x = \mathbf{send}\ (y, x)\ \mathbf{in}\ \mathbf{close}\ x$$

Consider the program which creates a new channel, with endpoints x and x', spawns off an echo server listening on x', and then returns x:

$$\bullet\ \mathbf{let}\ (x, x') = \mathbf{new}\ \mathbf{in}\ \mathbf{spawn}\ (\lambda().\mathbf{echo}_{x'}); x$$

If we reduce the above program, we get $(\nu x x')(\bullet\ x \parallel \circ\ \mathbf{echo}_{x'})$. Clearly, no more evaluation is possible, even though the configuration contains the thread $\circ\ \mathbf{echo}_{x'}$, which is blocked on x'. In corollary 1 we show that if a term does not return an endpoint, it must produce *only* a value.

Actions are terms which perform communication actions and which synchronise between two threads.

Definition 1 (Actions). *A term* acts on *an endpoint x if it is* $\mathbf{send}\ (V, x)$, $\mathbf{recv}\ x$, $\mathbf{close}\ x$, *or* $\mathbf{wait}\ x$. *A term is an* action *if it acts on some endpoint x.*

Ready terms are terms which are ready to perform a communication action, either by themselves, *e.g.*, creating a new channel or thread, or with another thread, *e.g.*, sending or receiving.

Definition 2 (Ready Terms). *A term L is* ready *if it is of the form $E[M]$, where M is of the form* $\mathbf{new}$, $\mathbf{spawn}\ N$, $\mathbf{link}\ (x, y)$ *or* $\mathbf{link}\ (y, x)$, *or M acts on x. In the latter case, we say that L is* ready to act on x.

Progress for the term language is standard for GV, and deviates from progress for linear λ-calculus only in that terms may reduce to values or *ready terms*:

Lemma 5 (Progress, $\longrightarrow_M$). *If $\Gamma \vdash^p M : T$ and Γ contains only session types, then:* (a) *M is a value;* (b) *$M \longrightarrow_M N$ for some N; or* (c) *M is ready.*

Canonical forms deviate from those for GV, in that we opt to move all ν-binders to the top. The standard GV canonical form, alternating ν-binders and their corresponding parallel compositions, does not work for PGV, since multiple channels may be split across a single parallel composition.

A configuration either reduces, or it is equivalent to configuration in normal form. Crucial to the normal form is that each term M_i is blocked on the corresponding channel x_i, and hence no two terms act on dual endpoints. Furthermore, no term M_i can perform a communication action by itself, since those are excluded by the definition of actions. Finally, as a corollary, we get that well-typed terms which do not return endpoints return *just* a value:

Definition 3 (Canonical Forms). *A configuration $\mathcal{C}$ is in canonical form if it is of the following form, where no term M_i is a value:*

$$(\nu x_1 x_1') \ldots (\nu x_n x_n')(\circ\ M_1 \parallel \cdots \parallel \circ\ M_m \parallel \bullet\ N).$$

Definition 4 (Normal Forms). *A configuration $\mathcal{C}$ is in normal form if it is of the following form, where each M_i is ready to act on x_i:*

$$(\nu x_1 x_1') \dots (\nu x_n x_n')(\circ\, M_1 \parallel \dots \parallel \circ\, M_m \parallel \bullet\, V).$$

Lemma 6 (Canonical Forms). *If $\Gamma \vdash^\bullet \mathcal{C}$, there exists some $\mathcal{D}$ such that $\mathcal{C} \equiv \mathcal{D}$ and $\mathcal{D}$ is in canonical form.*

Theorem 2 (Progress, $\longrightarrow_{\mathcal{C}}$). *If $\varnothing \vdash^\bullet \mathcal{C}$ and $\mathcal{C}$ is in canonical form, then either $\mathcal{C} \longrightarrow_{\mathcal{C}} \mathcal{D}$ for some $\mathcal{D}$; or $\mathcal{C} \equiv \mathcal{D}$ for some $\mathcal{D}$ in normal form.*

Proof. Our proof follows the reasoning by Kobayashi [26] used in the proof of deadlock freedom for closed processes (Theorem 2). Details are in Appendix A.

Corollary 1. *If $\varnothing \vdash^\bullet \mathcal{C}$, $\mathcal{C} \not\longrightarrow_{\mathcal{C}}$, and $\mathcal{C}$ contains no endpoints, then $\mathcal{C} \equiv \phi\, V$ for some value V.*

It follows immediately from theorem 2 and corollary 1 that a term which does not return an endpoint will complete all its communication actions, thus satisfying deadlock freedom.

3 Relation to Priority CP

We present a correspondence between Priority GV and an updated version of Priority CP [12, PCP], which is Wadler's CP [43] with priorities. This correspondence connects PGV to (a relaxed variant of) classical linear logic.

3.1 Revisiting Priority CP

Types (A, B) in PCP correspond to linear logic connectives annotated with priorities $o \in \mathbb{N}$. Typing environments, duality, and the priority function $\mathrm{pr}(\cdot)$ are defined as expected (see appendix B.1).

$$A, B ::= A \otimes^o B \mid A \parr^o B \mid \mathbf{1}^o \mid \bot^o \mid A \oplus^o B \mid A \,\&^o\, B \mid \mathbf{0}^o \mid \top^o$$

Processes (P, Q) in PCP are defined by the following grammar.

$$\begin{aligned} P, Q ::= {} & x \leftrightarrow y \mid (\nu x y) P \mid (P \parallel Q) \mid \mathbf{0} \\ \mid {} & x[y].P \mid x[].P \mid x(y).P \mid x().P \\ \mid {} & x \triangleleft \mathrm{inl}.P \mid x \triangleleft \mathrm{inr}.P \mid x \triangleright \{\mathrm{inl} : P; \mathrm{inr} : Q\} \mid x \triangleright \{\} \end{aligned}$$

Processes are typed by sequents $P \vdash \Gamma$, which correspond to the one-sided sequents in classical linear logic. Differently from PGV, in PCP we do not need to store the greatest priority on the sequent, as, due to the absence of higher-order functions, we cannot compose processes *sequentially*.

PCP decomposes cut into T-Res and T-Par rules—logically corresponding to cycle and mix rules, respectively—and guarantees deadlock freedom by using priority constraints, *e.g.*, as in T-Send. The full typing rules and operational semantics are available in figs. 7 and 8 in appendix B.1:

$$\text{T-Res}\ \frac{P \vdash \Gamma, x : A, y : A^\perp}{(\nu x y) P \vdash \Gamma} \qquad \text{T-Par}\ \frac{P \vdash \Gamma \qquad Q \vdash \Delta}{P \parallel Q \vdash \Gamma, \Delta} \qquad \text{T-Send}\ \frac{P \vdash \Gamma, y : A, x : B \qquad o < \min_{\mathrm{pr}}(\Gamma, A, B)}{x[y].P \vdash \Gamma, x : A \otimes^o B}$$

The main change we make to PCP is *removing commuting conversions* and defining an operational semantics based on structural congruence. Commuting conversions are necessary if we want our reduction strategy to correspond *exactly* to cut (or cycle in [12]) elimination. However, from the perspective of process calculi, commuting conversions behave strangely: they allow an input/output action to be moved to the top of a process, thus potentially blocking actions which were previously possible (an example is given in appendix B.1). This makes CP, and PCP in [12], non-confluent. As Lindley and Morris [30] show, all communications that can be performed *with* the use of commuting conversions, can also be performed *without* them, if using structural congruence.

In particular for PCP, commuting conversions break our intuition that an action with lower priority *occurs before* an action with higher priority. To cite Dardha and Gay [12] "*if a prefix on a channel endpoint* x *with priority* o *is pulled out at top level, then to preserve priority constraints in the typing rules [..], it is necessary to increase priorities of all actions after the prefix on* x" by $o+1$. One benefit of removing commuting conversions is that we no longer need to dynamically change the priorities during reduction, which means that the intuition for priorities holds true in our updated version of PCP. Furthermore, we can safely define reduction on untyped processes, which means that type and priority information is erasable!

We prove closed progress for our updated PCP. The proof is in appendix B.1.

Theorem 3 (Progress, $\Longrightarrow$).
If $P \vdash \varnothing$*, then either* $P = \mathbf{0}$ *or there exists a* Q *such that* $P \Longrightarrow Q$*.*

3.2 Correspondence between PGV and PCP

We illustrate the relation between PCP and PGV by defining a translation from PCP to PGV. The translation on types is defined as follows:

$$(\!|A \otimes^o B|\!) = !^o\overline{(\!|A|\!)}.(\!|B|\!) \quad (\!|\mathbf{1}^o|\!) = \mathbf{end}_!^o \quad (\!|A \parr^o B|\!) = ?^o(\!|A|\!).(\!|B|\!) \quad (\!|\bot^o|\!) = \mathbf{end}_?^o$$
$$(\!|A \oplus^o B|\!) = (\!|A|\!) \oplus^o (\!|B|\!) \quad (\!|\mathbf{0}^o|\!) = \oplus^o\{\} \quad (\!|A \,\&^o\, B|\!) = (\!|A|\!) \,\&^o\, (\!|B|\!) \quad (\!|\top^o|\!) = \&^o\{\}$$

There are two separate translations on processes. The main translation, $(\!|\cdot|\!)_M$, translates processes to *terms*:

$$\begin{aligned}
(\!|x{\leftrightarrow}y|\!)_M &= \mathbf{link}\ (x, y)\\
(\!|(\nu xy)P|\!)_M &= \mathbf{let}\ (x, y) = \mathbf{new}\ \mathbf{in}\ (\!|P|\!)_M\\
(\!|P \parallel Q|\!)_M &= \mathbf{spawn}\ (\lambda().(\!|P|\!)_M); (\!|Q|\!)_M\\
(\!|\mathbf{0}|\!)_M &= ()\\
(\!|x[].P|\!)_M &= \mathbf{close}\ x; (\!|P|\!)_M\\
(\!|x().P|\!)_M &= \mathbf{wait}\ x; (\!|P|\!)_M\\
(\!|x[y].P|\!)_M &= \mathbf{let}\ (y, z) = \mathbf{new}\ \mathbf{in}\ \mathbf{let}\ x = \mathbf{send}\ (z, x)\ \mathbf{in}\ (\!|P|\!)_M\\
(\!|x(y).P|\!)_M &= \mathbf{let}\ (y, x) = \mathbf{recv}\ x\ \mathbf{in}\ (\!|P|\!)_M\\
(\!|x \triangleleft \mathrm{inl}.P|\!)_M &= \mathbf{let}\ x = \mathbf{select}\ \mathbf{inl}\ x\ \mathbf{in}\ (\!|P|\!)_M\\
(\!|x \triangleleft \mathrm{inr}.P|\!)_M &= \mathbf{let}\ x = \mathbf{select}\ \mathbf{inr}\ x\ \mathbf{in}\ (\!|P|\!)_M
\end{aligned}$$

$$
\begin{aligned}
(\!|x \triangleright \{\mathrm{inl} : P; \mathrm{inr} : Q\}|\!)_M &= \mathbf{offer}\ x\ \{\mathbf{inl}\ x \mapsto (\!|P|\!)_M; \mathbf{inr}\ x \mapsto (\!|Q|\!)_M\} \\
(\!|x \triangleright \{\}|\!)_M &= \mathbf{offer}\ x\ \{\}
\end{aligned}
$$

Unfortunately, the operational correspondence along $(\!|\cdot|\!)_M$ is unsound, as it translates ν-binders and parallel compositions to **new** and **spawn**, which can reduce to their equivalent configuration constructs using E-New and E-Spawn. The same goes for ν-binders which are inserted when translating bound send to unbound send. For instance, the process $x[y].P$ is blocked, but its translation uses **new** and can reduce. To address this issue, we use a second translation, $(\!|\cdot|\!)_c$, which is equivalent to $(\!|\cdot|\!)_M$ followed by reductions using E-New and E-Spawn:

$$
\begin{aligned}
(\!|(\nu xy)P|\!)_c &= (\nu xy)(\!|P|\!)_c \\
(\!|P \parallel Q|\!)_c &= (\!|P|\!)_c \parallel (\!|Q|\!)_c \\
(\!|x[y].P|\!)_c &= (\nu yz)(\circ\ \mathbf{let}\ x = \mathbf{send}\ (z, x)\ \mathbf{in}\ (\!|P|\!)_M) \\
(\!|x \triangleleft \mathrm{inl}.P|\!)_c &= (\nu yz)(\circ\ \mathbf{let}\ x = \mathbf{close}\ (\mathbf{send}\ (\mathbf{inl}\ y, x)); z\ \mathbf{in}\ (\!|P|\!)_M) \\
(\!|x \triangleleft \mathrm{inr}.P|\!)_c &= (\nu yz)(\circ\ \mathbf{let}\ x = \mathbf{close}\ (\mathbf{send}\ (\mathbf{inr}\ y, x)); z\ \mathbf{in}\ (\!|P|\!)_M) \\
(\!|P|\!)_c &= \circ(\!|P|\!)_M, \quad \text{if none of the above apply}
\end{aligned}
$$

Typing environments are translated pointwise, and sequents $P \vdash \Gamma$ are translated as $(\!|\Gamma|\!) \vdash^\circ (\!|P|\!)_c$. The translations $(\!|\cdot|\!)_M$ and $(\!|\cdot|\!)_c$ preserve typing, and the latter induces a sound and complete operational correspondence. The proofs are by structural induction and can be found in appendix B.2.

Lemma 7 (Preservation, $(\!|\cdot|\!)_M$). *If $P \vdash \Gamma$, then $(\!|\Gamma|\!) \vdash^p (\!|P|\!)_M : \mathbf{1}$.*

Theorem 4 (Preservation, $(\!|\cdot|\!)_c$). *If $P \vdash \Gamma$, then $(\!|\Gamma|\!) \vdash^\circ (\!|P|\!)_c$.*

Lemma 8. *For any P, either:*

- $\circ\,(\!|P|\!)_M = (\!|P|\!)_c$*; or*
- $\circ\,(\!|P|\!)_M \longrightarrow_{\mathcal{C}}^+ (\!|P|\!)_c$*, and for any $\mathcal{C}$, if $\circ\,(\!|P|\!)_M \longrightarrow_{\mathcal{C}} \mathcal{C}$, then $\mathcal{C} \longrightarrow_{\mathcal{C}}^\star (\!|P|\!)_c$.*

Theorem 5 (Operational Correspondence, Soundness, $(\!|\cdot|\!)_c$).
If $P \vdash \Gamma$ and $(\!|P|\!)_c \longrightarrow_{\mathcal{C}} \mathcal{C}$, there exists a Q such that $P \Longrightarrow^+ Q$ and $\mathcal{C} \longrightarrow_{\mathcal{C}}^\star (\!|Q|\!)_c$

Theorem 6 (Operational Correspondence, Completeness, $(\!|\cdot|\!)_c$).
If $P \vdash \Gamma$ and $P \Longrightarrow Q$, then $(\!|P|\!)_c \longrightarrow_{\mathcal{C}}^+ (\!|Q|\!)_c$.

Milner's Cyclic Scheduler. As an example of a deadlock-free cyclic process, Dardha and Gay [12] introduce an implementation of Milner's cyclic scheduler [32] in Priority CP. We have reproduced the scheduler in our updated PCP, and showed its translation to PGV. Full details are given in appendix C.

4 Related Work and Conclusion

Deadlock freedom and progress. For the standard typed π-calculus, one line of work is Kobayashi's approach to deadlock freedom [24], where priorities are abstract tags defined over a partially ordered set. These tags were later simplified to pairs of natural numbers, called obligations and capabilities [26], which allowed more π-calculus processes to be typed. Padovani [33] adapted the obligation and capability pairs to session types, and later simplified them to a single priority for the linear π-calculus [34]. Furthermore, by exploiting the

encoding of session types into linear types [27,13] and the priority-based linear π-calculus, we can obtain deadlock freedom for the π-calculus with session types.

For the session-typed π-calculus, Dezani *et al.* [16] guarantee progress by allowing only one active session at a time. Dezani later [15] introduces a partial order on channels similar to Kobayashi [24]. Carbone and Debois [8] define progress for session typed π-calculus in terms of a *catalyser*, which provides a missing counterpart to a process, thus guaranteeing deadlock freedom. Carbone *et al.* [7] use such catalysers to show that progress is a compositional form of lock-freedom and that it can be lifted to session types via the encoding of session types to linear types. Vieira and Vasconcelos [41] use single priorities and an abstract partial order to guarantee deadlock freedom in a session-typed π-calculus. Multiparty Session Types (MPST) [23] guarantee deadlock freedom *within a single* session, but not for session interleaving. Consequently, several techniques for deadlock freedom in MPST have been defined [5,11].

Gay *et al.* [17] and Vasconcelos *et al.* [39,40] were the first to introduce a functional language with session types. However, such works, including early GV [18,19] did not guarantee deadlock freedom, until it was addressed via syntactic restrictions [30,43]. Toninho *et al.* [38] present a translation of simply-typed λ-calculus into session-typed π-calculus. However, their focus is not on deadlock freedom.

Ties with logic. The correspondence between logic and types lays the foundation for functional programming [44]. Since its inception by Girard [20], linear logic has been a candidate for a foundational correspondence for concurrent programs. A correspondence with linear π-calculus was established early on by Abramsky [1,4]. A correspondence between session-typed π-calculus and dual intuitionistic linear logic was developed by Caires and Pfenning [6, πDILL], and with classical linear logic by Wadler [44, CP], both guaranteeing deadlock freedom as a result of their connection to logic. Dardha and Gay [12, PCP] integrate Kobayashi and Padovani's work on priorities [26,34] with CP, creating the first calculus which combines priorities and strong ties with logic. Dardha and Pérez [14] compare Kobayashi-style typing and CLL typing, and show that CLL corresponds to a subsystem of Kobayashi's where restriction is applied once. Balzer *et al.* [2, SILL_S] introduce shared state, which breaks deadlock freedom. They later restore deadlock freedom using priorities [3, SILL_{S+}]. Carbone *et al.* [10] give a logical view of MPST with a generalised duality, called coherence.

Conclusion and future work. We answered our research question by presenting Priority GV, a session-typed functional language which allows cyclic communication structures and uses priorities to ensure deadlock freedom. We showed its relation to Priority CP [12] via an operational correspondence.

Our formalism so far only captures the core of GV. In future work, we plan to explore: recursion in PGV, by integrating the works of Lindley and Morris [31] and Padovani and Novara [35]; and sharing, following Balzer and Pfenning [2].

Acknowledgement. The authors would like to thank Simon Fowler and Philip Wadler for they useful and insightful feedback.

References

1. Abramsky, S.: Proofs as processes. Theoretical Computer Science **135**(1), 5–9 (1994)
2. Balzer, S., Pfenning, F.: Manifest sharing with session types. Proc. of the ACM on Programming Languages **1**(ICFP), 37:1–37:29 (2017)
3. Balzer, S., Toninho, B., Pfenning, F.: Manifest deadlock-freedom for shared session types. In: Proc. of ESOP. Lecture Notes in Computer Science, vol. 11423, pp. 611–639. Springer (2019)
4. Bellin, G., Scott, P.J.: On the pi-calculus and linear logic. Theoretical Computer Science **135**(1), 11–65 (1994)
5. Bettini, L., Coppo, M., D'Antoni, L., Luca, M.D., Dezani-Ciancaglini, M., Yoshida, N.: Global progress in dynamically interleaved multiparty sessions. In: Proc. of CONCUR. LNCS, vol. 5201, pp. 418–433. Springer (2008)
6. Caires, L., Pfenning, F.: Session types as intuitionistic linear propositions. In: Proc. of CONCUR. LNCS, vol. 6269, pp. 222–236. Springer (2010)
7. Carbone, M., Dardha, O., Montesi, F.: Progress as compositional lock-freedom. In: Proc. of COORDINATION. LNCS, vol. 8459, pp. 49–64. Springer (2014)
8. Carbone, M., Debois, S.: A graphical approach to progress for structured communication in web services. In: Proc. of ICE. EPTCS, vol. 38, pp. 13–27 (2010)
9. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centred programming for web services. In: Proc. of ESOP. LNCS, vol. 4421, pp. 2–17. Springer (2007)
10. Carbone, M., Lindley, S., Montesi, F., Schürmann, C., Wadler, P.: Coherence generalises duality: A logical explanation of multiparty session types. In: Proc. of CONCUR. LIPIcs, vol. 59, pp. 33:1–33:15. Schloss Dagstuhl—Leibniz-Zentrum für Informatik (2016)
11. Carbone, M., Montesi, F.: Deadlock-freedom-by-design: multiparty asynchronous global programming. In: Proc. of POPL. pp. 263–274 (2013)
12. Dardha, O., Gay, S.J.: A new linear logic for deadlock-free session-typed processes. In: Proc. of FoSSaCS. LNCS, vol. 10803, pp. 91–109. Springer (2018)
13. Dardha, O., Giachino, E., Sangiorgi, D.: Session types revisited. In: Proc. of PPDP. pp. 139–150. ACM (2012)
14. Dardha, O., Pérez, J.A.: Comparing deadlock-free session typed processes. In: Proc. of EXPRESS/SOS. EPTCS, vol. 190, pp. 1–15 (2015)
15. Dezani-Ciancaglini, M., de'Liguoro, U., Yoshida, N.: On progress for structured communications. In: Proc. of TGC. LNCS, vol. 4912, pp. 257–275. Springer (2009)
16. Dezani-Ciancaglini, M., Mostrous, D., Yoshida, N., Drossopoulou, S.: Session types for object-oriented languages. In: Proc. of ECOOP. LNCS, vol. 4067, pp. 328–352. Springer (2006)
17. Gay, S.J., Nagarajan, R.: Intensional and extensional semantics of dataflow programs. Formal Aspects of Computing **15**(4), 299–318 (2003)
18. Gay, S.J., Vasconcelos, V.T.: Linear type theory for asynchronous session types. Journal of Functional Programming **20**(1), 19–50 (2010)
19. Gay, S.J., Vasconcelos, V.T.: Linear type theory for asynchronous session types. JFP **20**(1), 19–50 (2012), extended version of [18]
20. Girard, J.Y.: Linear logic. Theoretical Computer Science **50**, 1–102 (1987)
21. Honda, K.: Types for dyadic interaction. In: Proc. of CONCUR. LNCS, vol. 715, pp. 509–523. Springer (1993)
22. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Proc. of ESOP. LNCS, vol. 1381, pp. 122–138. Springer (1998)

23. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: Proc. of POPL. vol. 43(1), pp. 273–284. ACM (2008)
24. Kobayashi, N.: A partially deadlock-free typed process calculus. ACM Transactions on Programming Languages and Systems **20**(2), 436–482 (1998)
25. Kobayashi, N.: Type systems for concurrent programs. In: Proc. of UNU/IIST. pp. 439–453 (2002)
26. Kobayashi, N.: A new type system for deadlock-free processes. In: Proc. of CONCUR. LNCS, vol. 4137, pp. 233–247. Springer (2006)
27. Kobayashi, N.: Type systems for concurrent programs (2007), extended version of [25]
28. Kokke, W., Montesi, F., Peressotti, M.: Better late than never: A fully-abstract semantics for classical processes. Proc. of POPL **3**(POPL) (Jan 2019)
29. Kokke, W., Montesi, F., Peressotti, M.: Taking linear logic apart. In: Ehrhard, T., Fernández, M., Paiva, V.d., Tortora de Falco, L. (eds.) Proc. of Linearity & TLLA. EPTCS, vol. 292, pp. 90–103. Open Publishing Association (Apr 2019)
30. Lindley, S., Morris, J.G.: A semantics for propositions as sessions. In: Proc. of ESOP. pp. 560–584 (2015)
31. Lindley, S., Morris, J.G.: Talking bananas: structural recursion for session types. In: Proc. of ICFP. Association for Computing Machinery (ACM) (2016)
32. Milner, R.: Communication and Concurrency. Prentice Hall (1989)
33. Padovani, L.: From lock freedom to progress using session types. In: Proc. of PLACES. vol. 137, pp. 3–19. EPTCS (2013)
34. Padovani, L.: Deadlock and Lock Freedom in the Linear π-Calculus. In: Proc. of CSL-LICS. pp. 72:1–72:10. ACM (2014)
35. Padovani, L., Novara, L.: Types for Deadlock-Free Higher-Order Programs. In: Proc. of FORTE. LNCS, vol. 9039, pp. 3–18. Springer (2015)
36. Sangiorgi, D., Walker, D.: The π-calculus: a Theory of Mobile Processes. Cambridge University Press (2001)
37. Takeuchi, K., Honda, K., Kubo, M.: An interaction-based language and its typing system. In: Proc. of PARLE. LNCS, vol. 817, pp. 398–413. Springer (1994)
38. Toninho, B., Caires, L., Pfenning, F.: Functions as session-typed processes. In: Proc. of FoSSaCS. LNCS, vol. 7213, pp. 346–360. Springer (2012)
39. Vasconcelos, V., Ravara, A., Gay, S.J.: Session types for functional multithreading. In: Proc. of CONCUR. LNCS, vol. 3170, pp. 497–511. springer (2004)
40. Vasconcelos, V.T., Gay, S.J., Ravara, A.: Type checking a multithreaded functional language with session types. Theor. Comput. Sci. **368**(1-2), 64–87 (2006)
41. Vieira, H.T., Vasconcelos, V.T.: Typing progress in communication-centred systems. In: Proc. of COORDINATION. LNCS, vol. 7890, pp. 236–250. Springer (2013)
42. Wadler, P.: Propositions as sessions. In: Proc. of ICFP. pp. 273–286 (2012)
43. Wadler, P.: Propositions as sessions. Journal of Functional Programming **24**(2-3), 384–418 (Jan 2014), extended version of [42]
44. Wadler, P.: Propositions as types. Communications of the ACM **58**(12), 75–84 (2015)

A Priority GV

Lemma 1. *If* $\Gamma \vdash^{p} V : T$*, then* $p = \bot$*, and* $min_{\mathrm{pr}}(\Gamma) = min_{\mathrm{pr}}(T)$*.*

Proof. By induction on the derivation of $\Gamma \vdash^{o} V : T$.

Case (T-Lam). Immediately.

$$\frac{\Gamma, x : T \vdash^{q} M : U}{\Gamma \vdash^{\bot} \lambda x.M : T \multimap^{\mathrm{pr}(\Gamma),q} U}$$

Case (T-Unit). Immediately.

$$\frac{}{\varnothing \vdash^{\bot} () : \mathbf{1}}$$

Case (T-Pair). The induction hypotheses give us $p = q = \bot$, hence $p \sqcup q = \bot$, and $\mathrm{pr}(\Gamma) = \mathrm{pr}(T)$ and $\mathrm{pr}(\Delta) = \mathrm{pr}(U)$, hence $\mathrm{pr}(\Gamma, \Delta) = \mathrm{pr}(\Gamma) \sqcap \mathrm{pr}(\Delta) = \mathrm{pr}(T) \sqcap \mathrm{pr}(U) = \mathrm{pr}(T \times U)$.

$$\frac{\Gamma \vdash^{p} V : T \qquad \Delta \vdash^{q} W : U \qquad p < \mathrm{pr}(\Delta)}{\Gamma, \Delta \vdash^{p \sqcup q} (V, W) : T \times U}$$

Case (T-Inl). The induction hypothesis gives us $p = \bot$, and $\mathrm{pr}(\Gamma) = \mathrm{pr}\,(T)$. We know $\mathrm{pr}(T) = \mathrm{pr}(U)$, hence $\mathrm{pr}(\Gamma) = \mathrm{pr}(T + U)$.

$$\frac{\Gamma \vdash^{p} V : T \qquad \mathrm{pr}(T) = \mathrm{pr}(U)}{\Gamma \vdash^{p} \mathbf{inl}\ V : T + U}$$

Case (T-Inr). The induction hypothesis gives us $p = \bot$, and $\mathrm{pr}(\Gamma) = \mathrm{pr}\,(U)$. We know $\mathrm{pr}(T) = \mathrm{pr}(U)$, hence $\mathrm{pr}(\Gamma) = \mathrm{pr}(T + U)$.

$$\frac{\Gamma \vdash^{p} V : U \qquad \mathrm{pr}(T) = \mathrm{pr}(U)}{\Gamma \vdash^{p} \mathbf{inr}\ V : T + U}$$

Lemma 2 (Substitution).
If $\Gamma, x : U' \vdash^{p} M : T$ *and* $\Theta \vdash^{q} V : U'$*, then* $\Gamma, \Theta \vdash^{p} M\{V/x\} : T$*.*

Proof. By induction on the derivation of $\Gamma, x : U' \vdash^{p} M : T$.

Case (T-Var). By lemma 1, $q = \bot$.

$$\frac{}{x : U' \vdash^{\bot} x : U'} \xRightarrow{\{V/x\}} \Theta \vdash^{\bot} V : U'$$

Case (T-Lam). By lemma 1, $\mathrm{pr}(\Theta) = \mathrm{pr}(U')$, hence $\mathrm{pr}(\Gamma, \Theta) = \mathrm{pr}(\Gamma, U')$.

$$\frac{\Gamma, x : U', y : T \vdash^{q} M : U}{\Gamma, x : U' \vdash^{\bot} \lambda y.M : T \multimap^{\mathrm{pr}(\Gamma, U'),q} U} \xRightarrow{\{V/x\}} \frac{\Gamma, \Theta, y : T \vdash^{q} M\{V/x\} : U}{\Gamma, \Theta \vdash^{\bot} \lambda y.M\{V/x\} : T \multimap^{\mathrm{pr}(\Gamma, \Theta),q} U}$$

Case (T-App). There are two subcases:

Subcase $(x \in M)$. Immediately, from the induction hypothesis.

$$\frac{\Gamma, x : U' \vdash^{p} M : T \multimap^{p',q'} U \qquad \Delta \vdash^{q} N : T \qquad p < \mathrm{pr}(\Delta) \qquad q < p'}{\Gamma, \Delta, x : U' \vdash^{p \sqcup q \sqcup q'} M\ N : U} \xRightarrow{\{V/x\}}$$

$$\frac{\Gamma, \Theta \vdash^{p} M\{V/x\} : T \multimap^{p',q'} U \qquad \Delta \vdash^{q} N : T \qquad p < \mathrm{pr}(\Delta) \qquad q < p'}{\Gamma, \Delta, \Theta \vdash^{p \sqcup q \sqcup q'} (M\{V/x\})\ N : U}$$

Subcase $(x \in N)$. By lemma 1, $\mathrm{pr}(\Theta) = \mathrm{pr}(U')$, hence $\mathrm{pr}(\Delta, \Theta) = \mathrm{pr}(\Delta, U')$.

$$\frac{\Gamma \vdash^{p} M : T \multimap^{p',q'} U \qquad \Delta, x : U' \vdash^{q} N : T \qquad p < \mathrm{pr}(\Delta, U') \qquad q < p'}{\Gamma, \Delta, x : U' \vdash^{p \sqcup q \sqcup q'} M\ N : U} \xRightarrow{\{V/x\}}$$

$$\frac{\Gamma \vdash^{p} M : T \multimap^{p',q'} U \qquad \Delta, \Theta \vdash^{q} N\{V/x\} : T \qquad p < \mathrm{pr}(\Delta, \Theta) \qquad q < p'}{\Gamma, \Delta, \Theta \vdash^{p \sqcup q \sqcup q'} M\ (N\{V/x\}) : U}$$

Case (T-LetUnit). There are two subcases:

Subcase $(x \in M)$. Immediately, from the induction hypothesis.

$$\frac{\Gamma, x : U' \vdash^{p} M : \mathbf{1} \qquad \Delta \vdash^{q} N : T \qquad p < \mathrm{pr}(\Delta)}{\Gamma, \Delta, x : U' \vdash^{p \sqcup q} \mathbf{let}\ () = M\ \mathbf{in}\ N : T} \xRightarrow{\{V/x\}}$$

$$\frac{\Gamma, \Theta \vdash^{p} M\{V/x\} : \mathbf{1} \qquad \Delta \vdash^{q} N : T \qquad p < \mathrm{pr}(\Delta)}{\Gamma, \Delta, \Theta \vdash^{p \sqcup q} \mathbf{let}\ () = M\{V/x\}\ \mathbf{in}\ N : T}$$

Subcase $(x \in N)$. By lemma 1, $\mathrm{pr}(\Theta) = \mathrm{pr}(U')$, hence $\mathrm{pr}(\Delta, \Theta) = \mathrm{pr}(\Delta, U')$.

$$\frac{\Gamma \vdash^{p} M : \mathbf{1} \qquad \Delta, x : U' \vdash^{q} N : T \qquad p < \mathrm{pr}(\Delta, U')}{\Gamma, \Delta, x : U' \vdash^{p \sqcup q} \mathbf{let}\ () = M\ \mathbf{in}\ N : T} \xRightarrow{\{V/x\}}$$

$$\frac{\Gamma \vdash^{p} M : \mathbf{1} \qquad \Delta, \Theta \vdash^{q} N\{V/x\} : T \qquad p < \mathrm{pr}(\Delta, \Theta)}{\Gamma, \Delta, \Theta \vdash^{p \sqcup q} \mathbf{let}\ () = M\ \mathbf{in}\ N\{V/x\} : T}$$

Case (T-Pair). There are two subcases:

Subcase $(x \in M)$. Immediately, from the induction hypothesis.

$$\frac{\Gamma, x : U' \vdash^{p} M : T \qquad \Delta \vdash^{q} N : U \qquad p < \mathrm{pr}(\Delta, U')}{\Gamma, \Delta, x : U' \vdash^{p \sqcup q} (M, N) : T \times U} \xRightarrow{\{V/x\}}$$

$$\frac{\Gamma, \Theta \vdash^{p} M\{V/x\} : T \qquad \Delta \vdash^{q} N : U \qquad p < \mathrm{pr}(\Delta, \Theta)}{\Gamma, \Delta, \Theta \vdash^{p \sqcup q} (M\{V/x\}, N) : T \times U}$$

Subcase ($x \in N$). By lemma 1, $\mathrm{pr}(\Theta) = \mathrm{pr}(U')$, hence $\mathrm{pr}(\Delta, \Theta) = \mathrm{pr}(\Delta, U')$.

$$\frac{\Gamma \vdash^p M : T \qquad \Delta, x : U' \vdash^q N : U \qquad p < \mathrm{pr}(\Delta, U')}{\Gamma, \Delta, x : U' \vdash^{p \sqcup q} (M, N) : T \times U} \xrightarrow{\{V/x\}}$$

$$\frac{\Gamma \vdash^p M : T \qquad \Delta, \Theta \vdash^q N\{V/x\} : U \qquad p < \mathrm{pr}(\Delta, \Theta)}{\Gamma, \Delta, \Theta \vdash^{p \sqcup q} (M, N\{V/x\}) : T \times U}$$

Case (T-LetPair). There are two subcases:

Subcase ($x \in M$). Immediately, from the induction hypothesis.

$$\frac{\Gamma, x : U' \vdash^p M : T \times T' \qquad \Delta, y : T, z : T' \vdash^q N : U \qquad p < \mathrm{pr}(\Delta, T, T')}{\Gamma, \Delta, x : U' \vdash^{p \sqcup q} \mathbf{let}\ (y, z) = M\ \mathbf{in}\ N : U} \xrightarrow{\{V/x\}}$$

$$\frac{\Gamma, \Theta \vdash^p M\{V/x\} : T \times T' \qquad \Delta, y : T, z : T' \vdash^q N : U \qquad p < \mathrm{pr}(\Delta, T, T')}{\Gamma, \Delta, \Theta \vdash^{p \sqcup q} \mathbf{let}\ (y, z) = M\{V/x\}\ \mathbf{in}\ N : U}$$

Subcase ($x \in N$). By lemma 1, $\mathrm{pr}(\Theta) = \mathrm{pr}(U')$, hence $\mathrm{pr}(\Delta, \Theta, T, T') = \mathrm{pr}(\Delta, U', T, T')$.

$$\frac{\Gamma \vdash^p M : T \times T' \qquad \Delta, x : U', y : T, z : T' \vdash^q N : U \qquad p < \mathrm{pr}(\Delta, U', T, T')}{\Gamma, \Delta, x : U' \vdash^{p \sqcup q} \mathbf{let}\ (y, z) = M\ \mathbf{in}\ N : U} \xrightarrow{\{V/x\}}$$

$$\frac{\Gamma \vdash^p M : T \times T' \qquad \Delta, \Theta, y : T, z : T' \vdash^q N\{V/x\} : U \qquad p < \mathrm{pr}(\Delta, \Theta, T, T')}{\Gamma, \Delta, \Theta \vdash^{p \sqcup q} \mathbf{let}\ (y, z) = M\ \mathbf{in}\ N\{V/x\} : U}$$

Case (T-Absurd).

$$\frac{\Gamma, x : U' \vdash^p M : \mathbf{0}}{\Gamma, \Delta, x : U' \vdash^p \mathbf{absurd}\ M : T} \xrightarrow{\{V/x\}} \frac{\Gamma, \Theta \vdash^p M\{V/x\} : \mathbf{0}}{\Gamma, \Delta, \Theta \vdash^p \mathbf{absurd}\ M\{V/x\} : T}$$

Case (T-Inl).

$$\frac{\Gamma, x : U' \vdash^p M : T \qquad \mathrm{pr}(T) = \mathrm{pr}(U)}{\Gamma, x : U' \vdash^p \mathbf{inl}\ M : T + U} \xrightarrow{\{V/x\}} \frac{\Gamma, \Theta \vdash^p M\{V/x\} : T \qquad \mathrm{pr}(T) = \mathrm{pr}(U)}{\Gamma, \Theta \vdash^p \mathbf{inl}\ M\{V/x\} : T + U}$$

Case (T-Inr).

$$\frac{\Gamma, x : U' \vdash^p M : U \qquad \mathrm{pr}(T) = \mathrm{pr}(U)}{\Gamma, x : U' \vdash^p \mathbf{inr}\ M : T + U} \xrightarrow{\{V/x\}} \frac{\Gamma, \Theta \vdash^p M\{V/x\} : U \qquad \mathrm{pr}(T) = \mathrm{pr}(U)}{\Gamma, \Theta \vdash^p \mathbf{inr}\ M\{V/x\} : T + U}$$

Case (T-CaseSum). There are two subcases:

Subcase $(x \in L)$. Immediately, from the induction hypothesis.

$$\frac{\Gamma, x : U' \vdash^{p} L : T + T' \qquad \Delta, y : T \vdash^{q} M : U \qquad \Delta, z : T' \vdash^{q} N : U \qquad p < \mathrm{pr}(\Delta)}{\Gamma, \Delta, x : U' \vdash^{p \sqcup q} \mathbf{case}\ L\ \{\mathbf{inl}\ y \mapsto M;\ \mathbf{inr}\ z \mapsto N\} : U} \xRightarrow{\{V/x\}}$$

$$\frac{\Gamma, \Theta \vdash^{p} L\{V/x\} : T + T' \qquad \Delta, y : T \vdash^{q} M : U \qquad \Delta, z : T' \vdash^{q} N : U \qquad p < \mathrm{pr}(\Delta)}{\Gamma, \Delta, \Theta \vdash^{p \sqcup q} \mathbf{case}\ L\{V/x\}\ \{\mathbf{inl}\ y \mapsto M;\ \mathbf{inr}\ z \mapsto N\} : U}$$

Subcase $(x \in M$ *and* $x \in N)$. By lemma 1, $\mathrm{pr}(\Theta) = \mathrm{pr}(U')$, hence $\mathrm{pr}(\Delta, \Theta, T) = \mathrm{pr}(\Delta, U', T)$ and $\mathrm{pr}(\Delta, \Theta, T') = \mathrm{pr}(\Delta, U', T')$.

$$\frac{\Gamma \vdash^{p} L : T + T' \qquad \Delta, x : U', y : T \vdash^{q} M : U \qquad \Delta, x : U', z : T' \vdash^{q} N : U \qquad p < \mathrm{pr}(\Delta, U')}{\Gamma, \Delta, x : U' \vdash^{p \sqcup q} \mathbf{case}\ L\ \{\mathbf{inl}\ y \mapsto M;\ \mathbf{inr}\ z \mapsto N\} : U} \xRightarrow{\{V/x\}}$$

$$\frac{\Gamma \vdash^{p} L : T + T' \qquad \Delta, \Theta, y : T \vdash^{q} M\{V/x\} : U \qquad \Delta, \Theta, z : T' \vdash^{q} N\{V/x\} : U \qquad p < \mathrm{pr}(\Delta, \Theta)}{\Gamma, \Delta, \Theta \vdash^{p \sqcup q} \mathbf{case}\ L\ \{\mathbf{inl}\ y \mapsto M\{V/x\};\ \mathbf{inr}\ z \mapsto N\{V/x\}\} : U}$$

We omit the cases where $x \notin M$.

Lemma 3 (Subject Reduction, $\longrightarrow_M$).
If $\Gamma \vdash^{p} M : T$ *and* $M \longrightarrow_M M'$, *then* $\Gamma \vdash^{p} M' : T$.

Proof. By induction on the derivation of $M \longrightarrow_M M'$.

Case (E-Lam). By lemma 2.

$$\frac{\dfrac{\Gamma, x : T \vdash^{p} M : U}{\Gamma \vdash^{\perp} \lambda x.M : T \multimap^{\mathrm{pr}(\Gamma), p} U} \qquad \Delta \vdash^{\perp} V : T}{\Gamma, \Delta \vdash^{p} (\lambda x.M)\ V : U} \longrightarrow_M \Gamma, \Delta \vdash^{p} M\{V/x\} : U$$

Case (E-Unit). By lemma 2.

$$\frac{\dfrac{}{\varnothing \vdash^{\perp} () : \mathbf{1}} \qquad \Gamma \vdash^{p} M : T}{\Gamma \vdash^{p} \mathbf{let}\ () = ()\ \mathbf{in}\ M : T} \longrightarrow_M \Gamma \vdash^{p} M : T$$

Case (E-Pair). By lemma 2.

$$\frac{\dfrac{\dfrac{\Gamma \vdash^{\perp} V : T \qquad \Delta \vdash^{\perp} W : T'}{\Gamma, \Delta \vdash^{\perp} (V, W) : T \times T'} \qquad \Theta, x : T, y : T' \vdash^{p} M : U}{\Gamma, \Delta, \Theta \vdash \mathbf{let}\ (x, y) = (V, W)\ \mathbf{in}\ M : U}$$

$$\downarrow_M$$

$$\Gamma, \Delta, \Theta \vdash^{p} M\{V/x\}\{W/y\} : U$$

Case (E-Inl). By lemma 2.

$$\dfrac{\dfrac{\Gamma \vdash^{\perp} V : T}{\Gamma \vdash^{\perp} \mathbf{inl}\ V : T + T'} \qquad \Delta, x : T \vdash^{p} M : U \qquad \Delta, y : T' \vdash^{p} N : U}{\Gamma, \Delta \vdash^{p} \mathbf{case}\ \mathbf{inl}\ V\ \{\mathbf{inl}\ x \mapsto M;\ \mathbf{inr}\ y \mapsto N\} : U}$$

$$\Downarrow M$$

$$\Gamma, \Delta \vdash^{p} M\{V/x\} : U$$

Case (E-Inr). By lemma 2.

$$\dfrac{\dfrac{\Gamma \vdash^{\perp} V : T'}{\Gamma \vdash^{\perp} \mathbf{inr}\ V : T + T'} \qquad \Delta, x : T \vdash^{p} M : U \qquad \Delta, y : T' \vdash^{p} N : U}{\Gamma, \Delta \vdash^{p} \mathbf{case}\ \mathbf{inr}\ V\ \{\mathbf{inl}\ x \mapsto M;\ \mathbf{inr}\ y \mapsto N\} : U}$$

$$\Downarrow M$$

$$\Gamma, \Delta \vdash^{p} N\{V/y\} : U$$

Case (E-Lift). By induction on the evaluation context E.

Lemma 4 (Subject Congruence, $\equiv$).
If $\Gamma \vdash^{\phi} \mathcal{C}$ and $\mathcal{C} \equiv \mathcal{C}'$, then $\Gamma \vdash^{\phi} \mathcal{C}'$.

Proof. By induction on the derivation of $\mathcal{C} \equiv \mathcal{C}'$.

Case (SC-LinkSwap).

$$\dfrac{\dfrac{}{\mathbf{link} : S \times \overline{S} \multimap \mathbf{1}} \qquad \dfrac{\dfrac{}{x : S \vdash^{\perp} x : S} \qquad \dfrac{}{y : \overline{S} \vdash^{\perp} y : \overline{S}}}{x : S, y : \overline{S} \vdash^{\perp} (x, y) : S \times \overline{S}}}{x : S, y : \overline{S} \vdash^{\perp} \mathbf{link}\ (x, y) : \mathbf{1}}$$
$$\vdots$$
$$\Gamma, x : S, y : \overline{S} \vdash^{\phi} \mathcal{F}[\mathbf{link}\ (x, y)]$$

$$\Vert$$

$$\dfrac{\dfrac{}{\mathbf{link} : \overline{S} \times S \multimap \mathbf{1}} \qquad \dfrac{\dfrac{}{y : \overline{S} \vdash^{\perp} y : \overline{S}} \qquad \dfrac{}{x : S \vdash^{\perp} x : S}}{x : S, y : \overline{S} \vdash^{\perp} (y, x) : S \times \overline{S}}}{x : S, y : \overline{S} \vdash^{\perp} \mathbf{link}\ (y, x) : \mathbf{1}}$$
$$\vdots$$
$$\Gamma, x : S, y : \overline{S} \vdash^{\phi} \mathcal{F}[\mathbf{link}\ (y, x)]$$

Case (SC-ResLink).

$$\dfrac{\dfrac{\mathbf{link} : S \times \overline{S} \multimap \mathbf{1} \qquad \dfrac{\overline{x : S \vdash^{\perp} x : S} \qquad \overline{y : \overline{S} \vdash^{\perp} y : \overline{S}}}{x : S, y : \overline{S} \vdash^{\perp} (x, y) : S \times \overline{S}}}{x : S, y : \overline{S} \vdash^{\perp} \mathbf{link}\ (x, y) : \mathbf{1}}}{\dfrac{x : S, y : \overline{S} \vdash^{\phi} \phi\ \mathbf{link}\ (x, y)}{\varnothing \vdash^{\phi} (\nu xy)(\phi\ \mathbf{link}\ (x, y))}} \equiv \dfrac{\overline{\varnothing \vdash^{\phi} () : \mathbf{1}}}{\varnothing \vdash^{\phi} \phi\ ()}$$

Case (SC-ResSwap).

$$\dfrac{\Gamma, x : S, y : \overline{S} \vdash^{\phi} \mathcal{C}}{\Gamma \vdash^{\phi} (\nu xy)\mathcal{C}} \equiv \dfrac{\Gamma, x : S, y : \overline{S} \vdash^{\phi} \mathcal{C}}{\Gamma \vdash^{\phi} (\nu yx)\mathcal{C}}$$

Case (SC-ResComm).

$$\dfrac{\dfrac{\Gamma, x : S, y : \overline{S}, z : S', w : \overline{S'} \vdash^{\phi} \mathcal{C}}{\Gamma, x : S, y : \overline{S} \vdash^{\phi} (\nu zw)\mathcal{C}}}{\Gamma \vdash^{\phi} (\nu xy)(\nu zw)\mathcal{C}} \equiv \dfrac{\dfrac{\Gamma, x : S, y : \overline{S}, z : S', w : \overline{S'} \vdash^{\phi} \mathcal{C}}{\Gamma, z : S', w : \overline{S'} \vdash^{\phi} (\nu xy)\mathcal{C}}}{\Gamma \vdash^{\phi} \vdash (\nu zw)(\nu xy)\mathcal{C}}$$

Case (SC-ResExt).

$$\dfrac{\dfrac{\Gamma \vdash^{\phi} \mathcal{C} \qquad \Delta, x : S, y : \overline{S} \vdash^{\phi} \mathcal{D}}{\Gamma, \Delta, x : S, y : \overline{S} \vdash^{\phi} (\mathcal{C} \parallel \mathcal{D})}}{\Gamma, \Delta \vdash^{\phi} (\nu xy)(\mathcal{C} \parallel \mathcal{D})} \equiv \dfrac{\Gamma \vdash^{\phi} \mathcal{C} \qquad \dfrac{\Delta, x : S, y : \overline{S} \vdash^{\phi} \mathcal{D}}{\Delta \vdash^{\phi} (\nu xy)\mathcal{D}}}{\Gamma, \Delta \vdash^{\phi} \mathcal{C} \parallel (\nu xy)\mathcal{D}}$$

Case (SC-ParNil).

$$\dfrac{\Gamma \vdash^{\phi} \mathcal{C} \qquad \dfrac{\overline{\varnothing \vdash^{\perp} () : \mathbf{1}}}{\varnothing \vdash^{\circ} \circ()}}{\Gamma \vdash^{\phi} \mathcal{C} \parallel \circ()} \equiv \Gamma \vdash^{\phi} \mathcal{C}$$

Case (SC-ParComm).

$$\dfrac{\Gamma \vdash^{\phi} \mathcal{C} \qquad \Delta \vdash^{\phi'} \mathcal{D}}{\Gamma, \Delta \vdash^{\phi+\phi'} (\mathcal{C} \parallel \mathcal{D})} \equiv \dfrac{\Delta \vdash^{\phi'} \mathcal{D} \qquad \Gamma \vdash^{\phi} \mathcal{C}}{\Gamma, \Delta \vdash^{\phi'+\phi} (\mathcal{D} \parallel \mathcal{C})}$$

Case (SC-ParAssoc).

$$\dfrac{\Gamma \vdash^{\phi} \mathcal{C} \qquad \dfrac{\Delta \vdash^{\phi'} \mathcal{D} \qquad \Theta \vdash^{\phi''} \mathcal{E}}{\Delta, \Theta \vdash^{\phi'+\phi''} (\mathcal{D} \parallel \mathcal{E})}}{\Gamma, \Delta, \Theta \vdash^{\phi+\phi'+\phi''} \mathcal{C} \parallel (\mathcal{D} \parallel \mathcal{E})} \equiv \dfrac{\dfrac{\Gamma \vdash^{\phi} \mathcal{C} \qquad \Delta \vdash^{\phi'} \mathcal{D}}{\Gamma, \Delta \vdash^{\phi+\phi'} (\mathcal{C} \parallel \mathcal{D})} \qquad \Theta \vdash^{\phi''} \mathcal{E}}{\Gamma, \Delta, \Theta \vdash^{\phi+\phi'+\phi''} (\mathcal{C} \parallel \mathcal{D}) \parallel \mathcal{E}}$$

Theorem 1 (Subject Reduction, $\longrightarrow_{\mathcal{C}}$).
If $\Gamma \vdash^{\phi} \mathcal{C}$ and $\mathcal{C} \longrightarrow_{\mathcal{C}} \mathcal{C}'$, then $\Gamma \vdash^{\phi} \mathcal{C}'$.

Proof. By induction on the derivation of $\mathcal{C} \longrightarrow_{\mathcal{C}} \mathcal{C}'$.

Case (E-New).

$$\dfrac{\dfrac{\overline{\mathbf{new} : \mathbf{1} \multimap S \times \overline{S}} \qquad \overline{\varnothing \vdash^{\perp} () : \mathbf{1}}}{\varnothing \vdash^{\perp} \mathbf{new}\ () : S \times \overline{S}} \\ \vdots \\ \Gamma \vdash^{\phi} \mathcal{F}[\mathbf{new}\ ()]}{\longrightarrow_{\mathcal{C}}} \qquad \dfrac{\dfrac{\overline{x : S \vdash^{\perp} x : S} \qquad \overline{y : \overline{S} \vdash^{\perp} y : \overline{S}}}{x : S, y : \overline{S} \vdash^{\perp} (x, y) : S \times \overline{S}} \\ \vdots \\ \Gamma, x : S, y : \overline{S} \vdash^{\phi} \mathcal{F}[(x, y)]}{\Gamma \vdash^{\phi} (\nu xy)\mathcal{F}[(x, y)]}$$

Case (E-Spawn).

$$\dfrac{\dfrac{\overline{\mathbf{spawn} : (\mathbf{1} \multimap^{p,q} \mathbf{1}) \multimap \mathbf{1}} \quad \Delta \vdash^{\perp} V : \mathbf{1} \multimap^{p,q} \mathbf{1}}{\Delta \vdash^{\perp} \mathbf{spawn}\ V : \mathbf{1}} \\ \vdots \\ \Gamma, \Delta \vdash^{\phi} \mathcal{F}[\mathbf{spawn}\ V]}{\searrow}$$

$$\dfrac{\dfrac{\overline{\varnothing \vdash^{\perp} () : \mathbf{1}}}{\vdots} \qquad \dfrac{\Delta \vdash^{\perp} V : \mathbf{1} \multimap^{p,q} \mathbf{1} \qquad \overline{\varnothing \vdash^{\perp} () : \mathbf{1}}}{\Delta \vdash^{q} V\ () : \mathbf{1}}}{\dfrac{\Gamma \vdash^{\phi} \mathcal{F}[()]}{\Gamma, \Delta \vdash^{\phi} \mathcal{F}[()] \parallel \circ\ (V\ ())} \qquad \Delta \vdash^{\circ} \circ\ (V\ ())}$$

Case (E-Send). See fig. 3.

Case (E-Close).

$$\dfrac{\dfrac{\dfrac{\overline{\mathbf{close} : \mathbf{end}^{o}_{!} \multimap^{\top,o} \mathbf{1}} \qquad \overline{x : \mathbf{end}^{o}_{!} \vdash^{\perp} x : \mathbf{end}^{o}_{!}}}{x : \mathbf{end}^{o}_{!} \vdash^{o} \mathbf{close}\ x : \mathbf{1}} \\ \vdots \\ \Gamma, x : \mathbf{end}^{o}_{!} \vdash^{\phi} \mathcal{F}[\mathbf{close}\ x]}{\dfrac{\Gamma, \Delta, x : \mathbf{end}^{o}_{!}, y : \mathbf{end}^{o}_{?} \vdash^{\phi+\phi'} \mathcal{F}[\mathbf{close}\ x] \parallel \mathcal{F}'[\mathbf{wait}\ y]}{\Gamma, \Delta \vdash^{\phi+\phi'} (\nu xy)(\mathcal{F}[\mathbf{close}\ x] \parallel \mathcal{F}'[\mathbf{wait}\ y])}} \qquad \dfrac{\dfrac{\dfrac{\overline{\mathbf{wait} : \mathbf{end}^{o}_{?} \multimap^{\top,o} \mathbf{1}} \qquad \overline{y : \mathbf{end}^{o}_{?} \vdash^{\perp} y : \mathbf{end}^{o}_{?}}}{y : \mathbf{end}^{o}_{?} \vdash^{o} \mathbf{wait}\ y : \mathbf{1}} \\ \vdots \\ \Delta, y : \mathbf{end}^{o}_{?} \vdash^{\phi'} \mathcal{F}'[\mathbf{wait}\ y]}$$

$$\searrow$$

$$\dfrac{\dfrac{\overline{\varnothing \vdash^{\perp} () : \mathbf{1}}}{\vdots} \qquad \dfrac{\overline{\varnothing \vdash^{\perp} () : \mathbf{1}}}{\vdots}}{\dfrac{\Gamma \vdash^{\phi} \mathcal{F}[()]}{\Gamma, \Delta \vdash^{\phi+\phi'} \mathcal{F}[()] \parallel \mathcal{F}'[()]} \qquad \Delta \vdash^{\phi'} \mathcal{F}'[()]}$$

Case (E-LiftC). By induction on the evaluation context $\mathcal{G}$.

Case (E-LiftM). By lemma 3.

Case (E-LiftSC). By lemma 4.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\textbf{send} : T \times {!^o}T.S \multimap^{\top,o} S} \quad \dfrac{\Delta \vdash^p V : T \qquad \dfrac{}{x : {!^o}T.S \vdash^{\perp} x : {!^o}T.S}}{\Delta, x : {!^o}T.S \vdash^p (V, x) : T \times {!^o}T.S}}{\Delta, x : {!^o}T.S \vdash^{p \sqcup o} \textbf{send}\ (V, x) : S}}{\vdots}}{\text{(A)} \triangleq \quad \Gamma, \Delta, x : {!^o}T.S \vdash^{\phi} \mathcal{F}[\textbf{send}\ (V, x)]}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\textbf{recv} : {?^o}T.\overline{S} \multimap^{\top,o} T \times \overline{S}} \quad \dfrac{}{y : {?^o}T.\overline{S} \vdash^{\perp} y : {?^o}T.\overline{S}}}{y : {?^o}T.\overline{S} \vdash^o \textbf{recv}\ y : T \times \overline{S}}}{\vdots}}{\text{(B)} \triangleq \quad \Theta, y : {?^o}T.\overline{S} \vdash^{\phi'} \mathcal{F}'[\textbf{recv}\ y]}$$

$$\Big\downarrow_{\mathcal{C}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : S \vdash^{\perp} x : S}}{\vdots} \quad \dfrac{\Delta \vdash^p V : T \qquad \dfrac{}{\Delta, y : \overline{S} \vdash^{\perp} y : \overline{S}}}{\Delta, y : \overline{S} \vdash^p (V, y) : T \times \overline{S}}}{\vdots}}{\Gamma, x : S \vdash^{\phi} \mathcal{F}[x] \qquad \Delta, \Theta, y : \overline{S} \vdash^{\phi'} \mathcal{F}'[(V, y)]}$$

$$\dfrac{\Gamma, \Delta, \Theta, x : S, y : \overline{S} \vdash^{\phi+\phi'} \mathcal{F}[x] \parallel \mathcal{F}'[(V, y)]}{\Gamma, \Delta, \Theta \vdash^{\phi+\phi'} (\nu xy)(\mathcal{F}[x] \parallel \mathcal{F}'[(V, y)])}$$

Fig. 3. Subject Reduction (E-Send)

Lemma 13. *If $\Gamma \vdash^{p} L : T$ is ready to act on $x : S \in \Gamma$, then the priority bound p is some priority o,* i.e.*, not $\bot$ or $\top$.*

Proof. Let $L = E[M]$. By induction on the structure of E. M has priority $\mathrm{pr}(S)$, and each constructor of the evaluation context E passes on the *maximum* of the priorities of its premises. No rule introduces the priority bound $\top$ on the sequent.

Lemma 6 (Canonical Forms). *If $\Gamma \vdash^{\bullet} \mathcal{C}$, there exists some $\mathcal{D}$ such that $\mathcal{C} \equiv \mathcal{D}$ and $\mathcal{D}$ is in canonical form.*

Proof. We move any ν-binders to the top using SC-ResExt, discard any superfluous occurrences of $\circ\ ()$ using SC-ParNil, and move the main thread to the rightmost position using SC-ParComm and SC-ParAssoc.

Theorem 2 (Progress, $\longrightarrow_{\mathcal{C}}$). *If $\varnothing \vdash^{\bullet} \mathcal{C}$ and $\mathcal{C}$ is in canonical form, then either $\mathcal{C} \longrightarrow_{\mathcal{C}} \mathcal{D}$ for some $\mathcal{D}$; or $\mathcal{C} \equiv \mathcal{D}$ for some $\mathcal{D}$ in normal form.*

Proof. Let $\mathcal{C} = (\nu x_1 x_1') \ldots (\nu x_n x_n')(\circ\ M_1 \parallel \cdots \parallel \circ\ M_m \parallel \bullet\ N)$. We apply lemma 5 to each M_i and N. If for any M_i or N we obtain a reduction $M_i \longrightarrow_M M_i'$ or $N \longrightarrow_M N'$, we apply E-LiftM and E-LiftC to obtain a reduction on $\mathcal{C}$. Otherwise, each term M_i is ready, and N is either ready or a value. Pick the *ready* term $L \in \{M_1, \ldots, M_m, N\}$ with the smallest priority bound. There are four cases:

1. If L is a new $E[\mathbf{new}]$, we apply E-New.
2. If L is a spawn $E[\mathbf{spawn}\ M]$, we apply E-Spawn.
3. If L is a link $E[\mathbf{link}\ (y, z)]$ or $E[\mathbf{link}\ (z, y)]$, we apply E-Link.
4. Otherwise, L is ready to act on some endpoint $y : S$. Let $y' : \overline{S}$ be the dual endpoint of y. The typing rules enforce the linear use of endpoints, so there must be a term $L' \in \{M_1, \ldots, M_m, N\}$ which uses y'. There are two cases:
   (a) L' is ready. By lemma 13, the priority of L is $\mathrm{pr}(S)$. By duality, $\mathrm{pr}(\overline{S}) = \mathrm{pr}(S)$.
   We cannot have $L = L'$, otherwise the action on y' would be guarded by the action on y, requiring $\mathrm{pr}(\overline{S}) < \mathrm{pr}(S)$. The term L' must be ready to act on y', otherwise the action y' would be guarded by another action with priority smaller than $\mathrm{pr}\,(S)$, which contradicts our choice of L as having the smallest priority. Therefore, we have two terms ready to act on dual endpoints. We apply the appropriate reduction rule, *i.e.*, E-Send or E-Close.
   (b) $L' = N$ and is a value. We rewrite $\mathcal{C}$ to put L in the position corresponding to the endpoint it is blocked on, using SC-ParComm, SC-ParAssoc, and optionally SC-ResSwap. We then repeat the steps above with the term with the next smallest priority, until either we find a reduction, or the configuration has reached the desired normal form. (The argument based on the priority being the smallest continues to hold, since we know that neither L nor L' will be picked, and no other term uses y or y'.)

$$\textsc{T-LamUnit}\ \frac{\Gamma \vdash^{q} M : T}{\Gamma \vdash^{\bot} \lambda().M : \mathbf{1} \multimap^{\min_{\mathrm{pr}}(\Gamma),q} T} \triangleq \frac{\dfrac{}{z : \mathbf{1} \vdash^{\bot} z : \mathbf{1}} \qquad \Gamma \vdash^{q} M : T}{\Gamma, z : \mathbf{1} \vdash^{q} \mathbf{let}\ () = z\ \mathbf{in}\ M : T}}{\Gamma \vdash^{\bot} \lambda z.\mathbf{let}\ () = z\ \mathbf{in}\ M : \mathbf{1} \multimap^{\min_{\mathrm{pr}}(\Gamma),q} T}$$

$$\textsc{T-LamPair}\ \frac{\Gamma, x : T, y : T' \vdash^{q} M : U}{\Gamma \vdash^{\bot} \lambda(x,y).M : T \times T' \multimap^{\min_{\mathrm{pr}}(\Gamma),q} U} \triangleq \frac{\dfrac{}{z : T \times T' \vdash^{\bot} z : T \times T'} \qquad \Gamma, x : T, y : T' \vdash^{q} M : U}{\Gamma, z : T \times T' \vdash^{q} \mathbf{let}\ (x,y) = z\ \mathbf{in}\ M : T}}{\Gamma \vdash^{\bot} \lambda z.\mathbf{let}\ (x,y) = z\ \mathbf{in}\ M : T \times T' \multimap^{\min_{\mathrm{pr}}(\Gamma),q} U}$$

$$\textsc{T-Let}\ \frac{\Gamma \vdash^{p} M : T \qquad \Delta, x : T \vdash^{q} N : U \qquad p < \min_{\mathrm{pr}}(\Delta)}{\Gamma, \Delta \vdash^{p \sqcup q} \mathbf{let}\ x = M\ \mathbf{in}\ N : U} \triangleq$$

$$\frac{\dfrac{\Delta, x : T \vdash^{q} N : U}{\Delta \vdash^{\bot} \lambda x.N : T \multimap^{\min_{\mathrm{pr}}(\Delta),q} U} \qquad \Gamma \vdash^{p} M : T \qquad p < \min_{\mathrm{pr}}(\Delta)}{\Gamma, \Delta \vdash^{q \sqcup p} (\lambda x.N)\ M : U}$$

$$\textsc{T-Fork}\ \frac{}{\varnothing \vdash^{\bot} \mathbf{fork} : (S \multimap^{p,q} \mathbf{1}) \multimap \overline{S}} \triangleq$$

$$\frac{\dfrac{\dfrac{}{\varnothing \vdash^{\bot} () : \mathbf{1}}(\textsc{A})}{\varnothing \vdash^{\bot} \mathbf{new}\ () : S \times \overline{S}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{x : S \multimap^{p,q} \mathbf{1} \vdash^{\bot} x : S \multimap^{p,q} \mathbf{1}} \qquad \dfrac{}{y : S \vdash^{\bot} y : S}}{x : S \multimap^{p,q} \mathbf{1}, y : S \vdash^{q} x\ y : \mathbf{1}}}{x : S \multimap^{p,q} \mathbf{1}, y : S \vdash^{\bot} \lambda().x\ y : \mathbf{1} \multimap^{p,q} \mathbf{1}}(\textsc{B})}{x : S \multimap^{p,q} \mathbf{1}, y : S \vdash^{\bot} \mathbf{spawn}\ (\lambda().x\ y) : \mathbf{1}} \qquad \dfrac{}{z : \overline{S} \vdash^{\bot} z : \overline{S}}}{x : S \multimap^{p,q} \mathbf{1}, y : S, z : \overline{S} \vdash^{\bot} \mathbf{spawn}\ (\lambda().x\ y); z : \overline{S}}}{x : S \multimap^{p,q} \mathbf{1} \vdash^{\bot} \mathbf{let}\ (y,z) = \mathbf{new}\ ()\ \mathbf{in}\ \mathbf{spawn}\ (\lambda().x\ y); z : \overline{S}}}{\varnothing \vdash^{\bot} \lambda x.\mathbf{let}\ (y,z) = \mathbf{new}\ ()\ \mathbf{in}\ \mathbf{spawn}\ (\lambda().x\ y); z : (S \multimap^{p,q} \mathbf{1}) \multimap \overline{S}}$$

$$(\textsc{A}) = \mathbf{new} : \mathbf{1} \multimap S \times \overline{S} \qquad (\textsc{B}) = \mathbf{spawn} : (\mathbf{1} \multimap^{p,q} \mathbf{1}) \multimap \mathbf{1}$$

Fig. 4. Typing Rules for Syntactic Sugar for PGV (T-LamUnit, T-LamPair, T-Let, and T-Fork).

$$\textsc{T-Select-Inl}\quad \frac{\min_{\mathrm{pr}}(S) = \min_{\mathrm{pr}}(S')}{\varnothing \vdash^{\perp} \mathbf{select\ inl} : S \oplus^{o} S' \multimap^{\top,o} S} \triangleq$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y : \overline{S} \vdash y : \overline{S}}}{y : \overline{S} \vdash \mathbf{inl}\ y : \overline{S} + \overline{S'}} \qquad \dfrac{}{x : S \oplus^{o} S' \vdash x : S \oplus^{o} S'}}{x : S \oplus^{o} S', y : \overline{S} \vdash (\mathbf{inl}\ y, x) : (\overline{S} + \overline{S'}) \times (S \oplus^{o} S')}\ (\textsc{c})}{x : S \oplus^{o} S', y : \overline{S} \vdash \mathbf{send}\ (\mathbf{inl}\ y, x) : \mathbf{end}_{!}^{o+1}}\ (\textsc{b}) \qquad \dfrac{}{z : S \vdash z : S}}{\dfrac{\dfrac{\dfrac{}{\varnothing \vdash () : \mathbf{1}}\ (\textsc{a})}{\varnothing \vdash \mathbf{new}\ () : \overline{S} \times S} \qquad x : S \oplus^{o} S', y : \overline{S} \vdash \mathbf{close}\ (\mathbf{send}\ (\mathbf{inl}\ y, x)) : \mathbf{1}}{x : S \oplus^{o} S', y : \overline{S}, z : S \vdash \mathbf{close}\ (\mathbf{send}\ (\mathbf{inl}\ y, x)); z : S}}{x : S \oplus^{o} S' \vdash \mathbf{let}\ (y, z) = \mathbf{new}\ ()\ \mathbf{in\ close}\ (\mathbf{send}\ (\mathbf{inl}\ y, x)); z : S}}{\varnothing \vdash \lambda x.\mathbf{let}\ (y, z) = \mathbf{new}\ ()\ \mathbf{in\ close}\ (\mathbf{send}\ (\mathbf{inl}\ y, x)); z : S \oplus^{o} S' \multimap^{\top,o} S}$$

$$(\textsc{a}) = \mathbf{new} : \mathbf{1} \multimap^{\top,o} \overline{S} \times S \qquad (\textsc{b}) = \mathbf{close} : \mathbf{end}_{!}^{o+1} \multimap^{\top,o+1} \mathbf{1} \qquad (\textsc{c}) = \mathbf{send} : (\overline{S} + \overline{S'}) \times (S \oplus^{o} S') \multimap^{\top,o} \mathbf{end}_{!}^{o+1}$$

$$\textsc{T-Select-Inr}\quad \frac{\min_{\mathrm{pr}}(S) = \min_{\mathrm{pr}}(S')}{\varnothing \vdash^{\perp} \mathbf{select\ inr} : S \oplus^{o} S' \multimap^{\top,o} S'} \triangleq$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y : \overline{S'} \vdash y : \overline{S'}}}{y : \overline{S'} \vdash \mathbf{inr}\ y : \overline{S} + \overline{S'}} \qquad \dfrac{}{x : S \oplus^{o} S' \vdash x : S \oplus^{o} S'}}{x : S \oplus^{o} S', y : \overline{S'} \vdash (\mathbf{inr}\ y, x) : (\overline{S} + \overline{S'}) \times (S \oplus^{o} S')}\ (\textsc{c})}{x : S \oplus^{o} S', y : \overline{S'} \vdash \mathbf{send}\ (\mathbf{inr}\ y, x) : \mathbf{end}_{!}^{o+1}}\ (\textsc{b}) \qquad \dfrac{}{z : S' \vdash z : S'}}{\dfrac{\dfrac{\dfrac{}{\varnothing \vdash () : \mathbf{1}}\ (\textsc{a})}{\varnothing \vdash \mathbf{new}\ () : \overline{S'} \times S'} \qquad x : S \oplus^{o} S', y : \overline{S'} \vdash \mathbf{close}\ (\mathbf{send}\ (\mathbf{inr}\ y, x)) : \mathbf{1}}{x : S \oplus^{o} S', y : \overline{S'}, z : S' \vdash \mathbf{close}\ (\mathbf{send}\ (\mathbf{inr}\ y, x)); z : S'}}{x : S \oplus^{o} S' \vdash \mathbf{let}\ (y, z) = \mathbf{new}\ ()\ \mathbf{in\ close}\ (\mathbf{send}\ (\mathbf{inr}\ y, x)); z : S'}}{\varnothing \vdash \lambda x.\mathbf{let}\ (y, z) = \mathbf{new}\ ()\ \mathbf{in\ close}\ (\mathbf{send}\ (\mathbf{inr}\ y, x)); z : S \oplus^{o} S' \multimap^{\top,o} S'}$$

$$(\textsc{a}) = \mathbf{new} : \mathbf{1} \multimap^{\top,o} \overline{S'} \times S' \qquad (\textsc{b}) = \mathbf{close} : \mathbf{end}_{!}^{o+1} \multimap^{\top,o+1} \mathbf{1} \qquad (\textsc{c}) = \mathbf{send} : (\overline{S} + \overline{S'}) \times (S \oplus^{o} S') \multimap^{\top,o} \mathbf{end}_{!}^{o+1}$$

Fig. 5. Typing Rules for Syntactic Sugar for PGV (T-Select-Inl and T-Select-Inr).

$$\textsc{T-Offer}\quad \frac{\Gamma \vdash^{p} L : S \,\&^{o}\, S' \quad \Delta, x : S \vdash^{q} M : T \quad \Delta, y : S' \vdash^{q} N : T \quad o \sqcup p < \min_{\mathrm{pr}}(\Delta, S, S')}{\Gamma, \Delta \vdash^{o \sqcup p \sqcup q} \mathbf{offer}\ L\ \{\mathbf{inl}\ x \mapsto M; \mathbf{inr}\ y \mapsto N\} : T} \quad \triangleq$$

$$\dfrac{\dfrac{(\textsc{b})\quad \overline{w : \mathbf{end}_{?}^{o+1} \vdash^{\perp} w : \mathbf{end}_{?}^{o+1}}}{w : \mathbf{end}_{?}^{o+1} \vdash^{o} \mathbf{wait}\ w : \mathbf{1}} \quad \dfrac{\overline{z : S + S' \vdash^{\perp} z : S + S'} \quad \Delta, x : S \vdash^{q} M : T \quad \Delta, y : S' \vdash^{q} N : T}{\Delta, z : S + S' \vdash^{q} \mathbf{case}\ z\ \{\mathbf{inl}\ x \mapsto M;\ \mathbf{inr}\ y \mapsto N\} : T} \quad o < \min_{\mathrm{pr}}(\Delta, S + S')}{\Delta, z : S + S', w : \mathbf{end}_{?}^{o+1} \vdash^{o \sqcup q} \mathbf{wait}\ w; \mathbf{case}\ z\ \{\mathbf{inl}\ x \mapsto M;\ \mathbf{inr}\ y \mapsto N\} : T}$$

(c)

$$\dfrac{\dfrac{(\textsc{a})\quad \Gamma \vdash^{p} L : ?^{o}(S + S').\mathbf{end}_{?}^{o+1}}{\Gamma \vdash^{o \sqcup p} \mathbf{recv}\ L : (S + S') \times \mathbf{end}_{?}^{o+1}} \quad (\textsc{c})\ \ o \sqcup p < \min_{\mathrm{pr}}(\Delta)}{\Gamma, \Delta \vdash^{o \sqcup p \sqcup q} \mathbf{let}\ (z, w) = \mathbf{recv}\ L\ \mathbf{in}\ \mathbf{wait}\ w; \mathbf{case}\ z\ \{\mathbf{inl}\ x \mapsto M;\ \mathbf{inr}\ y \mapsto N\} : T}$$

$$(\textsc{a}) = \mathbf{recv} : ?^{o}(S + S').\mathbf{end}_{?}^{o+1} \multimap^{\top, o} (S + S') \times \mathbf{end}_{?}^{o+1} \qquad (\textsc{b}) = \mathbf{wait} : \mathbf{end}_{?}^{o+1} \multimap^{\top, o} \mathbf{1}$$

$$\textsc{T-Offer-Absurd}\quad \frac{\Gamma \vdash^{p} L : \&^{o}\{\}\quad o \sqcup p < \min_{\mathrm{pr}}(\Delta)}{\Gamma, \Delta \vdash^{o \sqcup p} \mathbf{offer}\ L\ \{\} : T} \quad \triangleq$$

$$\dfrac{(\textsc{a})\quad \Gamma \vdash^{p} L : ?^{o}\mathbf{0}.\mathbf{end}_{?}^{o+1}}{\Gamma \vdash^{o \sqcup p} \mathbf{recv}\ L : \mathbf{0} \times \mathbf{end}_{?}^{o+1}} \quad \dfrac{\dfrac{(\textsc{b})\quad \overline{w : \mathbf{end}_{?}^{o+1} \vdash^{\perp} w : \mathbf{end}_{?}^{o+1}}}{w : \mathbf{end}_{?}^{o+1} \vdash^{o} \mathbf{wait}\ w : \mathbf{1}} \quad \dfrac{\overline{z : \mathbf{0} \vdash^{\perp} z : \mathbf{0}}}{\Delta, z : \mathbf{0} \vdash^{\perp} \mathbf{absurd}\ z : T} \quad o < \min_{\mathrm{pr}}(\Delta)}{\Delta, z : \mathbf{0}, w : \mathbf{end}_{?}^{o+1} \vdash^{o} \mathbf{wait}\ w; \mathbf{absurd}\ z : T} \quad o \sqcup p < \min_{\mathrm{pr}}(\Delta)}{\Gamma, \Delta \vdash^{o \sqcup p} \mathbf{let}\ (z, w) = \mathbf{recv}\ L\ \mathbf{in}\ \mathbf{wait}\ w; \mathbf{absurd}\ z : T}$$

$$(\textsc{a}) = \mathbf{recv} : ?^{o}\mathbf{0}.\mathbf{end}_{?}^{o+1} \multimap^{\top, o} \mathbf{0} \times \mathbf{end}_{?}^{o+1} \qquad (\textsc{b}) = \mathbf{wait} : \mathbf{end}_{?}^{o+1} \multimap^{\top, o} \mathbf{1}$$

Fig. 6. Typing Rules for Syntactic Sugar for PGV (T-Offer and T-Offer-Absurd).

B Relation to Priority CP

B.1 Revisiting Priority CP

Types. Types (A, B) in PCP are based on classical linear logic propositions, and are defined by the following grammar:

$$A, B ::= A \otimes^o B \mid A \parr^o B \mid \mathbf{1}^o \mid \bot^o \mid A \oplus^o B \mid A \,\&^o\, B \mid \mathbf{0}^o \mid \top^o$$

Each connective is annotated with a priority $o \in \mathbb{N}$. Types $A \otimes^o B$ and $A \parr^o B$ type the endpoints of a channel over which we send or receive a channel of type A, and then proceed as type B. Types $\mathbf{1}^o$ and $\bot[o]$ type the endpoints of a channel whose session has terminated, and over which we send or receive a *ping* before closing the channel. These two types act as units for $A \otimes^o B$ and $A \parr^o B$, respectively. Types $A \oplus^o B$ and $A \,\&^o\, B$ type the endpoints of a channel over which we can receive or send a choice between two branches A or B. We have opted for a simplified version of choice and followed the original Wadler's CP [43], however types $\oplus$ and $\&$ can be trivially generalised to $\oplus^o\{l_i : A_i\}_{i\in I}$ and $\&^o\{l_i : A_i\}_{i\in I}$, respectively, as in the original PCP [12]. Types $\mathbf{0}^o$ and $\top^o$ type the endpoints of a channel over which we can send or receive a choice between *no options*. These two types act as units for $A \oplus^o B$ and $A \,\&^o\, B$, respectively.

Environments. Typing environments Γ, Δ associate names to types. Environments are linear, so two environments can only be combined as Γ, Δ if their names are distinct, *i.e.*, $\mathrm{fv}(\Gamma) \cap \mathrm{fv}(\Delta) = \varnothing$.

$$\Gamma, \Delta ::= \varnothing \mid \Gamma, x : A$$

Duality. Duality is an involutive function on types which preserves priorities:

$$\begin{array}{llll} (\mathbf{1}^o)^\bot = \bot^o & (A \otimes^o B)^\bot = A^\bot \parr^o B^\bot & (\mathbf{0}^o)^\bot = \top^o & (A \oplus^o B)^\bot = A^\bot \,\&^o\, B^\bot \\ (\bot^o)^\bot = \mathbf{1}^o & (A \parr^o B)^\bot = A^\bot \otimes^o B^\bot & (\top^o)^\bot = \mathbf{0}^o & (A \,\&^o\, B)^\bot = A^\bot \oplus^o B^\bot \end{array}$$

Priorities. The function $\mathrm{pr}(\cdot)$ returns smallest priority of a type. As with PGV, the type system guarantees that the top-most connective always holds the smallest priority. The function $\min_{\mathrm{pr}}(\cdot)$ returns the *minimum* priority of all types a typing context, or $\top$ if the context is empty:

$$\begin{array}{llll} \mathrm{pr}(\mathbf{1}^o) = o & \mathrm{pr}(A \otimes^o B) = o & \mathrm{pr}(\mathbf{0}^o) = o & \mathrm{pr}(A \oplus^o B) = o \\ \mathrm{pr}(\bot^o) = o & \mathrm{pr}(A \parr^o B) = o & \mathrm{pr}(\top^o) = o & \mathrm{pr}(A \,\&^o\, B) = o \end{array}$$

$$\min_{\mathrm{pr}}(\varnothing) = \top \quad \min_{\mathrm{pr}}(\Gamma, x : A) = \min_{\mathrm{pr}}(\Gamma) \sqcap \min_{\mathrm{pr}}(A)$$

Terms. Processes (P, Q) in PCP are defined by the following grammar.

$$\begin{array}{rl} P, Q ::= & x \leftrightarrow y \mid (\nu xy)P \mid (P \parallel Q) \mid \mathbf{0} \\ & \mid x[y].P \mid x[].P \mid x(y).P \mid x().P \\ & \mid x \lhd \mathrm{inl}.P \mid x \lhd \mathrm{inr}.P \mid x \rhd \{\mathrm{inl} : P; \mathrm{inr} : Q\} \mid x \rhd \{\} \end{array}$$

Process $x \leftrightarrow y$ links endpoints x and y and forwards communication from one to the other. $(\nu xy)P$, $(P \parallel Q)$ and $\mathbf{0}$ denote respectively the restriction processes where channel endpoints x and y are bound together and with scope P, the parallel composition of processes P and Q and the terminated process. Processes $x[y].P$ and $x(y).P$ send or receive over channel x a value y and proceed as process P. Processes $x[].P$ and $x().P$ send and receive an empty value—denoting the closure of channel x, and continue as

P. Processes $x \triangleleft \mathsf{inl}.P$ and $x \triangleleft \mathsf{inr}.P$ make a left and right choice, respectively and proceed as process P. Dually, $x \triangleright \{\mathsf{inl} : P; \mathsf{inr} : Q\}$ offers both left and right branches, with continuations P and Q, and $x \triangleright \{\}$ is the empty offer. We write *unbound* send as $x\langle y\rangle.P$, which is syntactic sugar for $x[z].(y \leftrightarrow z \parallel P)$. Alternatively, we could take $x\langle y\rangle.P$ as primitive, and let $x[y].P$ be syntactic sugar for $(\nu yz)(x\langle z\rangle.P)$. CP takes *bound* sending as primitive, as it is impossible to eliminate the top-level cut in terms such as $(\nu yz)(x\langle z\rangle.P)$, even with commuting conversions. In our setting without commuting conversions and with more permissive normal forms, this is no longer an issue, but, for simplicity, we keep bound sending as primitive.

On Commuting Conversions. The main change we make to PCP is *removing commuting conversions*. Commuting conversions are necessary if we want our reduction strategy to correspond *exactly* to cut (or cycle in [12]) elimination. However, as Lindley and Morris [30] show, all communications that can be performed *with* the use of commuting conversions, can also be performed *without* them, but using structural congruence.

From the perspective of process calculi, commuting conversions behave strangely. Consider the commuting conversion $(\kappa_{\mathcal{Y}})$ for $x(y).P$:

$$(\kappa_{\mathcal{Y}}) \quad (\nu zz')(x(y).P \parallel Q) \Longrightarrow x(y).(\nu zz')(P \parallel Q)$$

As a result of $(\kappa_{\mathcal{Y}})$, Q becomes blocked on $x(y)$, and any actions Q was able to perform become unavailable. Consequently, CP is non-confluent:

$$\begin{array}{ccc} & (\nu xx')(a(y).P \parallel (\nu zz')(z[].\mathbf{0} \parallel z'().Q)) & \\ & \Downarrow \qquad\qquad\qquad\qquad \Downarrow^{+} & \\ a(y).(\nu xx')(P \parallel (\nu zz')(z[].\mathbf{0} \parallel z'().Q)) & & a(y).(\nu xx')(P \parallel Q) \end{array}$$

In PCP, commuting conversions break our intuition that an action with lower priority occurs before an action with higher priority. To cite Dardha and Gay [12] "*if a prefix on a channel endpoint x with priority o is pulled out at top level, then to preserve priority constraints in the typing rules [..], it is necessary to increase priorities of all actions after the prefix on x*" by $o + 1$.

Operational Semantics. The operational semantics for PCP, given in fig. 7, is defined as a reduction relation $\Longrightarrow$ on processes (bottom) and uses structural congruence (top). Each of the axioms of structural congruence corresponds to the axiom of the same name for PGV. We write $\Longrightarrow^{+}$ for the transitive closures, and $\Longrightarrow^{\star}$ for the reflexive-transitive closures.

The reduction relation is given by a set of axioms and inference rules for context closure. Reduction occurs under restriction. E-Link reduces a parallel composition with a link into a substitution. E-Send is the main communication rule, where send and receive processes sychronise and reduce to the corresponding continuations. E-Close follows the previous rule and it closes the channel identified by endpoints x and y. E-Select-Inl and E-Select-Inr are generalised versions of E-Send. They state respectively that a left and right selection synchronises with a choice offering and reduces to the corresponding continuations. The last three rules state that reduction is closed under restriction, parallel composition and structural congruence, respectively.

Typing. Figure 8 gives the typing rules for our version of PCP. A typing judgement $P \vdash \Gamma$ states that "process P is well typed under the typing context Γ".

T-Link states that the link process $x \leftrightarrow y$ is well typed under channels x and y having dual types, respectively A and $A^{\perp}$. T-Res sates that the restriction process $(\nu xy)P$ is well typed under typing

Structural congruence.

$$
\begin{array}{lll}
\text{SC-LinkSwap} & x{\leftrightarrow}y & \equiv y{\leftrightarrow}x \\
\text{SC-ResLink} & (\nu xy)x{\leftrightarrow}y & \equiv \mathbf{0} \\
\text{SC-ResSwap} & (\nu xy)P & \equiv (\nu yx)P \\
\text{SC-ResComm} & (\nu xy)(\nu zw)P & \equiv (\nu zw)(\nu xy)P \\
\text{SC-ResExt} & (\nu xy)(P \parallel Q) & \equiv P \parallel (\nu xy)Q, \text{ if } x, y \notin \mathrm{fv}(P) \\
\text{SC-ParNil} & P \parallel \mathbf{0} & \equiv P \\
\text{SC-ParComm} & P \parallel Q & \equiv Q \parallel P \\
\text{SC-ParAssoc} & P \parallel (Q \parallel R) & \equiv (P \parallel Q) \parallel R
\end{array}
$$

Reduction.

$$
\begin{array}{lll}
\text{E-Link} & (\nu xy)(w{\leftrightarrow}x \parallel P) & \Longrightarrow P\{w/x\} \\
\text{E-Send} & (\nu xy)(x[z].P \parallel x(w).Q) & \Longrightarrow (\nu xy)(\nu zw)(P \parallel Q) \\
\text{E-Close} & (\nu xy)(x[].P \parallel y().Q) & \Longrightarrow P \parallel Q \\
\text{E-Select-Inl} & (\nu xy)(x \triangleleft \mathrm{inl}.P \parallel x \triangleright \{\mathrm{inl} : Q; \mathrm{inr} : R\}) & \Longrightarrow (\nu xy)(P \parallel Q) \\
\text{E-Select-Inr} & (\nu xy)(x \triangleleft \mathrm{inr}.P \parallel x \triangleright \{\mathrm{inl} : Q; \mathrm{inr} : R\}) & \Longrightarrow (\nu xy)(P \parallel R)
\end{array}
$$

$$
\text{E-LiftRes}\ \frac{P \Longrightarrow P'}{(\nu xy)P \Longrightarrow (\nu xy)P'} \qquad \text{E-LiftPar}\ \frac{P \Longrightarrow P'}{P \parallel Q \Longrightarrow P' \parallel Q} \qquad \text{E-LiftSC}\ \frac{P \equiv P' \quad P' \Longrightarrow Q' \quad Q' \equiv Q}{P \Longrightarrow Q}
$$

Fig. 7. Operational Semantic for PCP.

context Γ if process P is well typed in Γ augmented with channel endpoints x and y having dual types, respectively A and $A^{\perp}$. T-Par states that the parallel composition of processes P and Q is well typed under the disjoint union of their respective typing contexts. T-Halt states that the terminated process $\mathbf{0}$ is well typed in the empty context. T-Send and T-Recv state that the sending and receiving of a bound name y over a channel x is well typed under Γ and x of type $A \otimes^{o} B$, respectively $A \parr^{o} B$. Priority o is the smallest among all priorities of the types used by the output or input process, captured by the side condition $o < \min_{\mathrm{pr}}(\Gamma, A, B)$. Rules T-Close and T-Wait type the closure of channel x and are in the same lines as the previous two rules, requiring that the priority of channel x is the smallest among all priorities in Γ. T-Select-Inl and T-Select-Inr type respectively the left $x \triangleleft \mathrm{inl}.P$ and right $x \triangleleft \mathrm{inr}.P$ choice performed on channel x. T-Offer and T-Offer-Absurd type the offering of a choice, or empty choice, on channel x. In all the above rules the priority o of channel x is the smallest with respect to the typing context $o < \min_{\mathrm{pr}}(\Gamma)$ and types involved in the choice $o < \min_{\mathrm{pr}}(\Gamma, A, B)$.

Finally, since our reduction relation is a strict subset of the reduction relation in the original [12], we defer to their proofs. We prove progress for our version of PCP, see appendix B.1.

Definition 5 (Actions). *A process acts on an endpoint x if it is $x{\leftrightarrow}y$, $y{\leftrightarrow}x$, $x[y].P$, $x(y).P$, $x[].P$, $x().P$, $x \triangleleft \mathit{inl}.P$, $x \triangleleft \mathit{inr}.P$, $x \triangleright \{\mathit{inl} : P; \mathit{inr} : Q\}$, or $x \triangleright \{\}$. A process is an action if it acts on some endpoint x.*

Definition 6 (Canonical Forms). *A process P is in canonical form if it is either $\mathbf{0}$ or of the form $(\nu x_1 x_1') \ldots (\nu x_n x_n')(P_1 \parallel \cdots \parallel P_m)$ where $m > 0$ and each P_j is an action.*

Lemma 15 (Canonical Forms). *If $P \vdash \Gamma$, there exists some Q such that $P \equiv Q$ and Q is in canonical form.*

$$\textsc{T-Link}\ \frac{}{x{\leftrightarrow}^{A}y \vdash x : A, y : A^{\perp}} \qquad \textsc{T-Res}\ \frac{P \vdash \Gamma, x : A, y : A^{\perp}}{(\nu xy)P \vdash \Gamma} \qquad \textsc{T-Par}\ \frac{P \vdash \Gamma \qquad Q \vdash \Delta}{P \parallel Q \vdash \Gamma, \Delta} \qquad \textsc{T-Halt}\ \frac{}{\mathbf{0} \vdash \varnothing}$$

$$\textsc{T-Send}\ \frac{P \vdash \Gamma, y : A, x : B \qquad o < \min_{\mathrm{pr}}(\Gamma, A, B)}{x[y].P \vdash \Gamma, x : A \otimes^{o} B} \qquad \textsc{T-Close}\ \frac{P \vdash \Gamma \qquad o < \min_{\mathrm{pr}}(\Gamma)}{x[].P \vdash \Gamma, x : \mathbf{1}^{o}}$$

$$\textsc{T-Recv}\ \frac{P \vdash \Gamma, y : A, x : B \qquad o < \min_{\mathrm{pr}}(\Gamma, A, B)}{x(y).P \vdash \Gamma, x : A \parr^{o} B} \qquad \textsc{T-Wait}\ \frac{P \vdash \Gamma \qquad o < \min_{\mathrm{pr}}(\Gamma)}{x().P \vdash \Gamma, x : \perp^{o}}$$

$$\textsc{T-Select-Inl}\ \frac{P \vdash \Gamma, x : A \qquad o < \min_{\mathrm{pr}}(\Gamma, A, B) \qquad \mathrm{pr}(A) = \mathrm{pr}(B)}{x \triangleleft \mathrm{inl}.P \vdash \Gamma, x : A \oplus^{o} B}$$

$$\textsc{T-Select-Inr}\ \frac{P \vdash \Gamma, x : B \qquad o < \min_{\mathrm{pr}}(\Gamma, A, B) \qquad \mathrm{pr}(A) = \mathrm{pr}(B)}{x \triangleleft \mathrm{inr}.P \vdash \Gamma, x : A \oplus^{o} B}$$

$$\textsc{T-Offer}\ \frac{P \vdash \Gamma, x : A \qquad Q \vdash \Gamma, x : B \qquad o < \min_{\mathrm{pr}}(\Gamma, A, B)}{x \triangleright \{\mathrm{inl} : P; \mathrm{inr} : Q\} \vdash \Gamma, x : A \mathbin{\&}^{o} B} \qquad \textsc{T-Offer-Absurd}\ \frac{o < \mathrm{pr}(\Gamma)}{x \triangleright \{\} \vdash \Gamma, x : \top^{o}}$$

Fig. 8. Typing Rules for PCP.

Proof. If $P = \mathbf{0}$, we are done. Otherwise, we move any ν-binders to the top using SC-ResExt, and discard any superfluous occurrences of $\mathbf{0}$ using SC-ParNil.

Theorem 3 (Progress, $\Longrightarrow$).
If $P \vdash \varnothing$, then either $P = \mathbf{0}$ or there exists a Q such that $P \Longrightarrow Q$.

Proof. By lemma 15, we rewrite P to canonical form. If the resulting process is $\mathbf{0}$, we are done. Otherwise, it is of the form

$$(\nu x_1 x_1') \ldots (\nu x_n x_n')(P_1 \parallel \cdots \parallel P_m) \vdash \varnothing$$

where $m > 0$ and each $P_i \vdash \Gamma_i$ is an action.

Our proof follows the same reasoning by Kobayashi [26] used in the proof of deadlock freedom for closed processes (Theorem 2).

Consider processes $P_1 \parallel \cdots \parallel P_m$. Among them, we pick the process with the smallest priority $\min_{\mathrm{pr}}(\Gamma_i)$ for all i. Let this process be let this be P_i and the priority of the top prefix be o. P_i acts on some endpoint $y : A \in \Gamma_i$. We must have $\min_{\mathrm{pr}}(\Gamma_i) = \mathrm{pr}\,(A) = o$, since the other actions in P_i are guarded by the action on $y : A$, thus satisfying law (i) of priorities.

If P_i is a link $y{\leftrightarrow}z$ or $z{\leftrightarrow}y$, we apply E-Link.

Otherwise, P_i is an input/branching or output/selection action on endpoint y of type A with priority o. Since process P is closed and consequently it respects law (ii) of priorities, there must be a co-action

$$\text{T-UnboundSend}\ \frac{P \vdash \Gamma, x : B \qquad o < \min_{\mathrm{pr}}(\Gamma, A, B)}{x\langle y\rangle.P \vdash \Gamma, x : A \otimes B, y : A^{\perp}} \triangleq \frac{\dfrac{\overline{z{\leftrightarrow}^{A} y \vdash y : A^{\perp}, z : A} \qquad \overline{P \vdash \Gamma, x : B}}{z{\leftrightarrow}^{A} y \parallel P \vdash \Gamma, x : B, y : A^{\perp}, z : A} \qquad o < \min_{\mathrm{pr}}(\Gamma, A, B)}{x[z].(z{\leftrightarrow}^{A} y \parallel P) \vdash \Gamma, x : A \otimes B, y : A^{\perp}}$$

Fig. 9. Typing Rules for Syntactic Sugar for PCP.

y' of type $A^{\perp}$ where y and y' are dual endpoints of the same channel (by application of rule T-Res). By duality, $\mathrm{pr}(A) = \mathrm{pr}(A^{\perp}) = o$. In the following we show that: y' is the subject of a top level action of a process P_j with $i \neq j$. This allows for the communication among P_i and P_j to happen immediately over channel endpoints y and y'.

Suppose that y' is an action not in a different parallel process P_j but rather of P_i itself. That means that the action on y' must be prefixed by the action on y, which is top level in P_i. To respect law (i) of priorities we must have $o < o$, which is absurd. This means that y' is in another parallel process P_j for $i \neq j$.

Suppose that y' in P_j is not at top level. In order to respect law (i) of priorities, it means that y' is prefixed by actions that are smaller than its priority o. This leads to a contradiction because stated that o is the smallest priority. Hence, y' must be the subject of a top level action.

We have two processes, acting on dual endpoints. We apply the appropriate reduction rule, *i.e.*, E-Send, E-Close, E-Select-Inl, or E-Select-Inr.

B.2 Correspondence between PGV and PCP

Lemma 7 (Preservation, $(\!|\cdot|\!)_M$). *If $P \vdash \Gamma$, then $(\!|\Gamma|\!) \vdash^{p} (\!|P|\!)_M : \mathbf{1}$.*

Proof. By induction on the derivation of $P \vdash \Gamma$.

*Case (*T-Link, T-Res, T-Par, *and* T-Halt*).* See fig. 10.

*Case (*T-Close, *and* T-Wait*).* See fig. 11.

*Case (*T-Send*).* See fig. 12.

*Case (*T-Recv*).* See fig. 13.

*Case (*T-Select-Inl, T-Select-Inr, *and* T-Offer*).* See fig. 14.

Theorem 4 (Preservation, $(\!|\cdot|\!)_c$). *If $P \vdash \Gamma$, then $(\!|\Gamma|\!) \vdash^{\circ} (\!|P|\!)_c$.*

Proof. By induction on the derivation of $P \vdash \Gamma$.

*Case (*T-Res*).* Immediately, from the induction hypothesis.

$$\text{T-Res}\ \frac{\Gamma, x : A, y : A^{\perp} \vdash P}{\Gamma \vdash (\nu xy)P} \xRightarrow{(\!|\cdot|\!)_c} \frac{(\!|\Gamma|\!), x : (\!|A|\!), y : (\!|B|\!) \vdash^{\circ} (\!|P|\!)_c}{(\!|\Gamma|\!) \vdash^{\circ} (\nu xy)(\!|P|\!)_c}$$

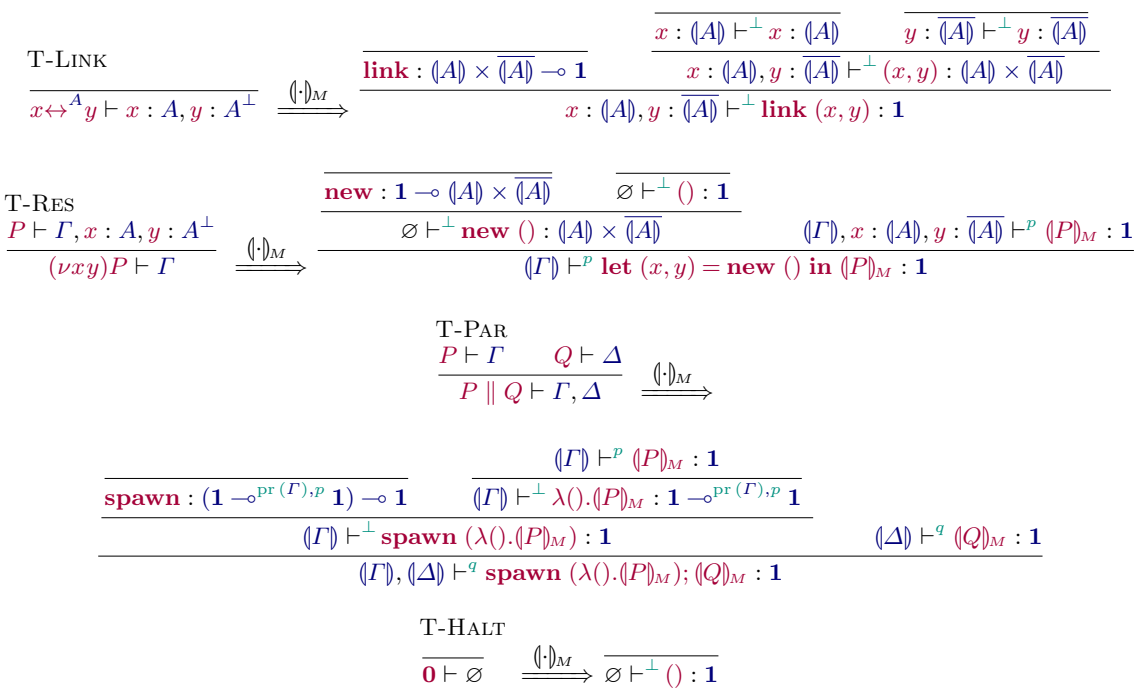


Fig. 10. Translation $(\!|\cdot|\!)_M$ preserves typing (T-Link, T-Res, T-Par, and T-Halt).

Case (T-Par). Immediately, from the induction hypotheses.

$$\dfrac{\Gamma \vdash P \qquad \Delta \vdash Q}{\Gamma, \Delta \vdash P \parallel Q} \xRightarrow{(\!|\cdot|\!)_C} \dfrac{(\!|\Gamma|\!) \vdash^{\circ} (\!|P|\!)_C \qquad (\!|\Delta|\!) \vdash^{\circ} (\!|Q|\!)_C}{(\!|\Gamma|\!), (\!|\Delta|\!) \vdash^{\circ} (\!|P|\!)_C \parallel (\!|Q|\!)_C}$$

Case (*). By lemma 7

$$\Gamma \vdash P \xRightarrow{(\!|\cdot|\!)_C} \dfrac{(\!|\Gamma|\!) \vdash^{p} (\!|P|\!)_M : \mathbf{1}}{(\!|\Gamma|\!) \vdash^{\circ} \circ\, (\!|P|\!)_M}$$

Theorem 5 (Operational Correspondence, Soundness, $(\!|\cdot|\!)_C$).
If $P \vdash \Gamma$ and $(\!|P|\!)_C \longrightarrow_{\mathcal{C}} \mathcal{C}$, there exists a Q such that $P \Longrightarrow^{+} Q$ and $\mathcal{C} \longrightarrow_{\mathcal{C}}^{\star} (\!|Q|\!)_C$

Proof. By induction on the derivation of $(\!|P|\!)_C \longrightarrow_{\mathcal{C}} \mathcal{C}$. We omit the cases which cannot occur as their left-hand side term forms are not in the image of the translation function, *i.e.*, E-New, E-Spawn, and E-LiftM.

Case (E-Link).

$$(\nu x x')(\mathcal{F}[\mathbf{link}\ (w, x)] \parallel \mathcal{C}) \longrightarrow_{\mathcal{C}} \mathcal{F}[()] \parallel \mathcal{C}\{w/x'\}$$

$$\textsc{T-Close}\quad \frac{P \vdash \Gamma \qquad o < \mathrm{pr}(\Gamma)}{x[].P \vdash \Gamma, x : \mathbf{1}^o} \xRightarrow{(\!|\cdot|\!)_M}$$

$$\frac{\dfrac{\overline{\mathbf{close} : \mathbf{end}_!^o \multimap^{\top,o} \mathbf{1}} \qquad \overline{x : \mathbf{end}_!^o \vdash^{\bot} x : \mathbf{end}_!^o}}{x : \mathbf{end}_!^o \vdash^o \mathbf{close}\ x : \mathbf{1}} \qquad (\!|\Gamma|\!) \vdash^p (\!|P|\!)_M : \mathbf{1} \qquad o < \mathrm{pr}((\!|\Gamma|\!))}{(\!|\Gamma|\!), x : \mathbf{end}_!^o \vdash^{o \sqcup p} \mathbf{close}\ x; (\!|P|\!)_M : \mathbf{1}}$$

$$\textsc{T-Wait}\quad \frac{P \vdash \Gamma \qquad o < \mathrm{pr}(\Gamma)}{x().P \vdash \Gamma, x : \mathbf{1}^o} \xRightarrow{(\!|\cdot|\!)_M}$$

$$\frac{\dfrac{\overline{\mathbf{wait} : \mathbf{end}_?^o \multimap^{\top,o} \mathbf{1}} \qquad \overline{x : \mathbf{end}_?^o \vdash^{\bot} x : \mathbf{end}_?^o}}{x : \mathbf{end}_?^o \vdash^o \mathbf{wait}\ x : \mathbf{1}} \qquad (\!|\Gamma|\!) \vdash^p (\!|P|\!)_M : \mathbf{1} \qquad o < \mathrm{pr}((\!|\Gamma|\!))}{(\!|\Gamma|\!), x : \mathbf{end}_?^o \vdash^{o \sqcup p} \mathbf{wait}\ x; (\!|P|\!)_M : \mathbf{1}}$$

Fig. 11. Translation $(\!|\cdot|\!)_M$ preserves typing (T-Close and T-Wait).

The source for $\mathbf{link}\ (w, x)$ *must* be $w \leftrightarrow x$. None of the translation rules introduce an evaluation context around the recursive call, hence $\mathcal{F}$ must be the empty context. Let P be the source term for $\mathcal{C}$, *i.e.*, $(\!|P|\!)_{\mathcal{C}} = \mathcal{C}$. Hence, we have:

$$\begin{array}{ccc}
(\nu x x')(w \leftrightarrow x \parallel P) & \xrightarrow{\Longrightarrow} & P\{w/x'\} \\
\downarrow{\scriptstyle (\!|\cdot|\!)_{\mathcal{C}}} & & \\
(\nu x x')(\circ\ \mathbf{link}\ (w, x) \parallel (\!|P|\!)_{\mathcal{C}}) & & \downarrow{\scriptstyle (\!|\cdot|\!)_{\mathcal{C}}} \\
\downarrow{\scriptstyle \longrightarrow_{\mathcal{C}}^+} & & \\
(\!|P|\!)_{\mathcal{C}}\{w/x'\} & \xrightarrow{=} & (\!|P\{w/x'\}|\!)_{\mathcal{C}}
\end{array}$$

Case (E-Send).

$$(\nu x x')(\mathcal{F}[\mathbf{send}\ (V, x)] \parallel \mathcal{F}'[\mathbf{recv}\ x']) \longrightarrow_{\mathcal{C}} (\nu x x')(\mathcal{F}[x] \parallel \mathcal{F}'[(V, x')])$$

There are three possible sources for **send** and **recv**: $x[y].P$ and $x'(y').Q$; $x \triangleleft \mathrm{inl}.P$ and $x' \triangleright \{\mathrm{inl} : Q; \mathrm{inr} : R\}$; or $x \triangleleft \mathrm{inr}.P$ and $x' \triangleright \{\mathrm{inl} : Q; \mathrm{inr} : R\}$.

Subcase ($x[y].P$ *and* $x'(y').Q$). None of the translation rules introduce an evaluation context around the recursive call, hence $\mathcal{F}$ must be $\circ\ \mathbf{let}\ x = \Box\ \mathbf{in}\ (\!|P|\!)_M$. Similarly, $\mathcal{F}'$ must be $\circ\ \mathbf{let}\ (y', x') = \Box\ \mathbf{in}\ (\!|Q|\!)_M$. The value V must be an endpoint y, bound by the name restriction $(\nu y y')$ introduced by the translation.

$$\dfrac{P \vdash \Gamma, y : A, x : B \qquad o < \mathrm{pr}(\Gamma, A, B)}{x[y].P \vdash \Gamma, x : A \otimes^{o} B}\ \text{T-Send} \quad \xRightarrow{(\!|\cdot|\!)_M}$$

$$\dfrac{\dfrac{}{\mathbf{new} : \mathbf{1} \multimap (\!|A|\!) \times \overline{(\!|A|\!)}}\ (\text{A}) \qquad \dfrac{}{\varnothing \vdash^{\perp} () : \mathbf{1}}}{\varnothing \vdash^{\perp} \mathbf{new}\ () : (\!|A|\!) \times \overline{(\!|A|\!)}}$$

$$\dfrac{\dfrac{}{\mathbf{send} : \overline{(\!|A|\!)} \times\, !^{o}\overline{(\!|A|\!)}.(\!|B|\!) \multimap^{\top, o} (\!|B|\!)}\ (\text{B}) \qquad \dfrac{\dfrac{}{z : \overline{(\!|A|\!)} \vdash^{\perp} x : \overline{(\!|A|\!)}} \qquad \dfrac{}{x : !^{o}\overline{(\!|A|\!)}.(\!|B|\!) \vdash^{\perp} x : !^{o}\overline{(\!|A|\!)}.(\!|B|\!)}}{x : !^{o}\overline{(\!|A|\!)}.(\!|B|\!), z : \overline{(\!|A|\!)} \vdash^{\perp} (z, x) : \overline{(\!|A|\!)} \times\, !^{o}\overline{(\!|A|\!)}.(\!|B|\!)}}{x : !^{o}\overline{(\!|A|\!)}.(\!|B|\!), z : \overline{(\!|A|\!)} \vdash^{o} \mathbf{send}\ (z, x) : (\!|B|\!)}$$

$$\dfrac{(\text{A}) \qquad \dfrac{(\text{B}) \qquad (\!|\Gamma|\!), y : (\!|A|\!), x : (\!|B|\!) \vdash^{p} (\!|P|\!)_M : \mathbf{1} \qquad o < \mathrm{pr}((\!|\Gamma|\!), (\!|A|\!), (\!|B|\!))}{(\!|\Gamma|\!), x : !^{o}\overline{(\!|A|\!)}.(\!|B|\!), y : (\!|A|\!), z : \overline{(\!|A|\!)} \vdash^{o \sqcup p} \mathbf{let}\ x = \mathbf{send}\ (z, x)\ \mathbf{in}\ (\!|P|\!)_M : \mathbf{1}}}{(\!|\Gamma|\!), x : !^{o}\overline{(\!|A|\!)}.(\!|B|\!) \vdash^{o \sqcup p} \mathbf{let}\ (y, z) = \mathbf{new}\ ()\ \mathbf{in\ let}\ x = \mathbf{send}\ (z, x)\ \mathbf{in}\ (\!|P|\!)_M : \mathbf{1}}$$

Fig. 12. Translation $(\!|\cdot|\!)_M$ preserves typing (T-Send).

Hence, we have:

$$\begin{array}{ccc}
(\nu x x')(x[y].P \parallel x'(y').Q) & \Longrightarrow & (\nu x x')(\nu y y')(P \parallel Q) \\
\Big\downarrow (\!|\cdot|\!)_C & & \\
(\nu x x')(\nu y y')\begin{pmatrix} \circ\ \mathbf{let}\ x = \mathbf{send}\ (y, x)\ \mathbf{in}\ (\!|P|\!)_M \parallel \\ \circ\ \mathbf{let}\ (y', x') = \mathbf{recv}\ x'\ \mathbf{in}\ (\!|Q|\!)_M \end{pmatrix} & & \Big\downarrow (\!|\cdot|\!)_C \\
\Big\downarrow \equiv \longrightarrow_C^{+} & & \\
(\nu x x')(\nu y y')(\circ\ (\!|P|\!)_M \parallel \circ\ (\!|Q|\!)_M) & \xrightarrow[\text{(by lemma 8)}]{\longrightarrow_C^{\star}} & (\nu x x')(\nu y y')((\!|P|\!)_C \parallel (\!|Q|\!)_C)
\end{array}$$

Subcase ($x \triangleleft \mathit{inl}.P$ *and* $x' \triangleright \{\mathit{inl} : Q; \mathit{inr} : R\}$). None of the translation rules introduce an evaluation context around the recursive call, hence $\mathcal{F}$ must be

$$\circ\ \mathbf{let}\ x = \mathbf{close}\ \Box; y\ \mathbf{in}\ (\!|P|\!)_M.$$

Similarly, $\mathcal{F}'$ must be

$$\circ\ \mathbf{let}\ (y', x') = \Box\ \mathbf{in\ wait}\ x'; \mathbf{case}\ y'\ \{\mathbf{inl}\ y' \mapsto (\!|Q|\!)_M;\ \mathbf{inr}\ y' \mapsto (\!|R|\!)_M\}.$$

$$\frac{P \vdash \Gamma, y : A, x : B \qquad o < \mathrm{pr}(\Gamma, A, B)}{x(y).P \vdash \Gamma, x : A \parr^{o} B}\ \text{T-Recv} \quad \xRightarrow{(\!|\cdot|\!)_M}$$

$$\frac{\dfrac{}{\mathbf{recv} : ?^{o}(\!|A|\!).(\!|B|\!) \multimap^{\top, o} (\!|A|\!) \times (\!|B|\!)}\ (\text{A}) \qquad \dfrac{}{x : ?^{o}(\!|A|\!).(\!|B|\!) \vdash^{\bot} x : ?^{o}(\!|A|\!).(\!|B|\!)}}{x : ?^{o}(\!|A|\!).(\!|B|\!) \vdash^{o} \mathbf{recv}\ x : (\!|A|\!) \times (\!|B|\!)}$$

$$\frac{(\text{A}) \qquad (\!|\Gamma|\!), y : (\!|A|\!), x : (\!|B|\!) \vdash^{p} (\!|P|\!)_M : \mathbf{1} \qquad o < \mathrm{pr}((\!|\Gamma|\!), (\!|A|\!), (\!|B|\!))}{(\!|\Gamma|\!), x : ?^{o}(\!|A|\!).(\!|B|\!), y : (\!|A|\!), z : (\!|A|\!) \vdash^{o \sqcup p} \mathbf{let}\ x = \mathbf{recv} x\ \mathbf{in}\ (\!|P|\!)_M : \mathbf{1}}$$

Fig. 13. Translation $(\!|\cdot|\!)_M$ preserves typing (T-Recv).

Hence, we have:

$$\begin{array}{ccc}
(\nu x x')(x \triangleleft \mathrm{inl}.P \parallel x \triangleright \{\mathrm{inl} : Q; \mathrm{inr} : R\}) & \Longrightarrow & (\nu x x')(P \parallel Q) \\
\downarrow (\!|\cdot|\!)_M & & \\
(\nu x x')\begin{pmatrix} \circ\ \mathbf{let}\ x = \mathbf{select\ inl}\ x\ \mathbf{in}\ (\!|P|\!)_M \parallel \\ \circ\ \mathbf{offer}\ x'\ \{\mathbf{inl}\ x' \mapsto (\!|Q|\!)_M; \mathbf{inr}\ x' \mapsto (\!|R|\!)_M\} \end{pmatrix} & & \downarrow (\!|\cdot|\!)_c \\
\downarrow \longrightarrow_c^{+} & & \\
(\nu x x')(\circ\ (\!|P|\!)_M \parallel \circ\ (\!|Q|\!)_M) & \xrightarrow[\text{(by lemma 8)}]{\longrightarrow_c^{\star}} & (\nu x x')((\!|P|\!)_c \parallel (\!|Q|\!)_c)
\end{array}$$

Subcase $(x \triangleleft inr.P$ *and* $x' \triangleright \{inl : Q; inr : R\})$. None of the translation rules introduce an evaluation context around the recursive call, hence $\mathcal{F}$ must be

$$\circ\ \mathbf{let}\ x = \mathbf{close}\ \Box; y\ \mathbf{in}\ (\!|P|\!)_M.$$

Similarly, $\mathcal{F}'$ must be

$$\circ\ \mathbf{let}\ (y', x') = \Box\ \mathbf{in\ wait}\ x'; \mathbf{case}\ y'\ \{\mathbf{inl}\ y' \mapsto (\!|Q|\!)_M;\ \mathbf{inr}\ y' \mapsto (\!|R|\!)_M\}.$$

Hence, we have:

$$\begin{array}{ccc}
(\nu x x')(x \triangleleft \mathrm{inr}.P \parallel x \triangleright \{\mathrm{inl} : Q; \mathrm{inr} : R\}) & \Longrightarrow & (\nu x x')(P \parallel Q) \\
\downarrow (\!|\cdot|\!)_M & & \\
(\nu x x')\begin{pmatrix} \circ\ \mathbf{let}\ x = \mathbf{select\ inr}\ x\ \mathbf{in}\ (\!|P|\!)_M \parallel \\ \circ\ \mathbf{offer}\ x'\ \{\mathbf{inl}\ x' \mapsto (\!|Q|\!)_M; \mathbf{inr}\ x' \mapsto (\!|R|\!)_M\} \end{pmatrix} & & \downarrow (\!|\cdot|\!)_c \\
\downarrow \longrightarrow_c^{+} & & \\
(\nu x x')(\circ\ (\!|P|\!)_M \parallel \circ\ (\!|R|\!)_M) & \xrightarrow[\text{(by lemma 8)}]{\longrightarrow_c^{\star}} & (\nu x x')((\!|P|\!)_c \parallel (\!|R|\!)_c)
\end{array}$$

$$\text{T-Select-Inl}\quad \frac{P \vdash \Gamma, x : A \qquad o < \mathrm{pr}(\Gamma)}{x \triangleleft \mathrm{inl}.P \vdash \Gamma, x : A \oplus^o B} \xrightarrow{(\!|\cdot|\!)_M}$$

$$\frac{\dfrac{}{\mathbf{select\ inl} : (\!|A|\!) \oplus^o (\!|B|\!) \multimap^{\top, o} (\!|A|\!)} \qquad \dfrac{}{x : (\!|A|\!) \oplus^o (\!|B|\!) \vdash^{\bot} x : (\!|A|\!) \oplus^o (\!|B|\!)}}{\begin{array}{c} x : (\!|A|\!) \oplus^o (\!|B|\!) \vdash^o \mathbf{select\ inl}\ x : (\!|A|\!) \\ \Gamma, x : (\!|A|\!) \vdash^p (\!|P|\!)_M : \mathbf{1} \qquad o < \mathrm{pr}(\Gamma) \end{array}}{\Gamma, x : (\!|A|\!) \oplus^o (\!|B|\!) \vdash^{o \sqcup p} \mathbf{let}\ x = \mathbf{select\ inl}\ x\ \mathbf{in}\ (\!|P|\!)_M : \mathbf{1}}$$

$$\text{T-Select-Inr}\quad \frac{P \vdash \Gamma, x : A \qquad o < \mathrm{pr}(\Gamma)}{x \triangleleft \mathrm{inr}.P \vdash \Gamma, x : A \oplus^o B} \xrightarrow{(\!|\cdot|\!)_M}$$

$$\frac{\dfrac{}{\mathbf{select\ inr} : (\!|A|\!) \oplus^o (\!|B|\!) \multimap^{\top, o} (\!|B|\!)} \qquad \dfrac{}{x : (\!|A|\!) \oplus^o (\!|B|\!) \vdash^{\bot} x : (\!|A|\!) \oplus^o (\!|B|\!)}}{\begin{array}{c} x : (\!|A|\!) \oplus^o (\!|B|\!) \vdash^o \mathbf{select\ inr}\ x : (\!|B|\!) \\ \Gamma, x : (\!|B|\!) \vdash^p (\!|P|\!)_M : \mathbf{1} \qquad o < \mathrm{pr}(\Gamma) \end{array}}{\Gamma, x : (\!|A|\!) \oplus^o (\!|B|\!) \vdash^{o \sqcup p} \mathbf{let}\ x = \mathbf{select\ inr}\ x\ \mathbf{in}\ (\!|P|\!)_M : \mathbf{1}}$$

$$\text{T-Offer}\quad \frac{P \vdash \Gamma, x : A \qquad Q \vdash \Gamma, x : B \qquad o < \mathrm{pr}(\Gamma, A, B)}{x \triangleright \{\mathrm{inl} : P; \mathrm{inr} : Q\} \vdash \Gamma, x : A \mathbin{\&^o} B} \xrightarrow{(\!|\cdot|\!)_M}$$

$$\frac{\dfrac{}{x : (\!|A|\!) \mathbin{\&^o} (\!|B|\!) \vdash^{\bot} x : (\!|A|\!) \mathbin{\&^o} (\!|B|\!)}}{\begin{array}{c} (\!|\Gamma|\!), x : (\!|A|\!) \vdash^p (\!|P|\!)_M : \mathbf{1} \qquad (\!|\Gamma|\!), x : (\!|B|\!) \vdash^p (\!|Q|\!)_M : \mathbf{1} \qquad o < \mathrm{pr}((\!|\Gamma|\!), (\!|A|\!), (\!|B|\!)) \end{array}}{(\!|\Gamma|\!), x : (\!|A|\!) \mathbin{\&^o} (\!|B|\!) \vdash^{o \sqcup p} \mathbf{offer}\ x\ \{\mathbf{inl}\ x \mapsto (\!|P|\!)_M; \mathbf{inr}\ x \mapsto (\!|Q|\!)_M\} : \mathbf{1}}$$

Fig. 14. Translation $(\!|\cdot|\!)_M$ preserves typing (T-Select-Inl, T-Select-Inr, and T-Offer).

Case (E-Close).

$$(\nu x x')(\mathcal{F}[\mathbf{wait}\ x] \parallel \mathcal{F}'[\mathbf{close}\ x']) \longrightarrow_{\mathcal{C}} \mathcal{F}[()] \parallel \mathcal{F}'[()]$$

The source for **wait** and **close** *must* be $x().P$ and $x'[].Q$.

(The translation for $x \triangleright \{\mathrm{inl} : P; \mathrm{inr} : Q\}$ also introduces a **wait**, but it is blocked on another communication, and hence cannot be the first communication on a translated term. The translations for $x \triangleleft \mathrm{inl}.P$ and $x \triangleleft \mathrm{inr}.P$ also introduce a **close**, but these are similarly blocked.)

None of the translation rules introduce an evaluation context around the recursive call, hence $\mathcal{F}$ must be $\Box; (\!|P|\!)_M$. Similarly, $\mathcal{F}'$ must be $\Box; (\!|Q|\!)_M$. Hence, we have:

$$\begin{array}{ccc}
(\nu x x')(x[].P \parallel x'().Q) & \xrightarrow{\quad\Longrightarrow\quad} & P \parallel Q \\
\Big\downarrow (\!|\cdot|\!)_M & & \\
(\nu x x')(\circ\, \mathbf{close}\ x; (\!|P|\!)_M \parallel \circ\, \mathbf{wait}\ x'; (\!|Q|\!)_M) & & \Big\downarrow (\!|\cdot|\!)_C \\
\Big\downarrow \longrightarrow_C^+ & & \\
\circ\, (\!|P|\!)_M \parallel \circ\, (\!|Q|\!)_M & \xrightarrow[\text{(by lemma 8)}]{\longrightarrow_C^\star} & (\!|P|\!)_C \parallel (\!|Q|\!)_C
\end{array}$$

Case (E-LiftC). By the induction hypothesis and E-LiftC.

Case (E-LiftSC). By the induction hypothesis, E-LiftSC, and lemma 18.

Lemma 8. *For any P, either:*
- $\circ\, (\!|P|\!)_M = (\!|P|\!)_C$*; or*
- $\circ\, (\!|P|\!)_M \longrightarrow_C^+ (\!|P|\!)_C$*, and for any $\mathcal{C}$, if $\circ\, (\!|P|\!)_M \longrightarrow_C \mathcal{C}$, then $\mathcal{C} \longrightarrow_C^\star (\!|P|\!)_C$.*

Proof. By induction on the structure of P.

Case ($(\nu x y)P$). We have:

$$\begin{aligned}
(\!|(\nu x y)P|\!)_M &= \circ\, \mathbf{let}\ (x, y) = \mathbf{new}\ \mathbf{in}\ (\!|P|\!)_M \\
&\longrightarrow_C^+ (\nu x y)(\circ\, (\!|P|\!)_M) \\
&\longrightarrow_C^\star (\nu x y)(\!|P|\!)_C \\
&= (\!|(\nu x y)P|\!)_C
\end{aligned}$$

Case ($P \parallel Q$).

$$\begin{aligned}
(\!|P \parallel Q|\!)_M &= \circ\, \mathbf{spawn}\ (\lambda().(\!|P|\!)_M); (\!|Q|\!)_M \\
&\longrightarrow_C^+ \circ\, (\!|P|\!)_M \parallel \circ\, (\!|Q|\!)_M \\
&\longrightarrow_C^\star (\!|P|\!)_C \parallel (\!|Q|\!)_C \\
&= (\!|P \parallel Q|\!)_C
\end{aligned}$$

Case ($x[y].P$).

$$\begin{aligned}
(\!|x[y].P|\!)_M &= \mathbf{let}\ (y, z) = \mathbf{new}\ \mathbf{in}\ \mathbf{let}\ x = \mathbf{send}\ (z, x)\ \mathbf{in}\ (\!|P|\!)_M \\
&\longrightarrow_C^+ (\nu y z)(\circ\, \mathbf{let}\ x = \mathbf{send}\ (z, x)\ \mathbf{in}\ (\!|P|\!)_M) \\
&= (\!|x[y].P|\!)_C
\end{aligned}$$

Case ($x \triangleleft \mathit{inl}.P$).

$$\begin{aligned}
(\!|x[y].P|\!)_M &= \mathbf{let}\ x = \mathbf{select}\ \mathbf{inl}\ x\ \mathbf{in}\ (\!|P|\!)_M \\
&\triangleq \mathbf{let}\ x = \mathbf{let}\ (y, z) = \mathbf{new}\ \mathbf{in}\ \mathbf{close}\ (\mathbf{send}\ (\mathbf{inl}\ y, x)); z\ \mathbf{in}\ (\!|P|\!)_M \\
&\longrightarrow_C^+ (\nu y z)(\circ\, \mathbf{let}\ x = \mathbf{close}\ (\mathbf{send}\ (\mathbf{inl}\ y, x)); z\ \mathbf{in}\ (\!|P|\!)_M) \\
&= (\!|x[y].P|\!)_C
\end{aligned}$$

Case $(x \triangleleft \mathit{inr}.P)$.

$$\begin{aligned}(\!|x[y].P|\!)_M &= \mathbf{let}\ x = \mathbf{select\ inr}\ x\ \mathbf{in}\ (\!|P|\!)_M \\ &\triangleq \mathbf{let}\ x = \mathbf{let}\ (y,z) = \mathbf{new\ in\ close}\ (\mathbf{send}\ (\mathbf{inr}\ y, x)); z\ \mathbf{in}\ (\!|P|\!)_M \\ &\longrightarrow_C^+ (\nu yz)(\circ\ \mathbf{let}\ x = \mathbf{close}\ (\mathbf{send}\ (\mathbf{inr}\ y, x)); z\ \mathbf{in}\ (\!|P|\!)_M) \\ &= (\!|x[y].P|\!)_C\end{aligned}$$

Case $(*)$. In all other cases, $\circ\ (\!|P|\!)_M = (\!|P|\!)_C$.

Lemma 18. *If* $P \vdash \Gamma$ *and* $P \equiv Q$, *then* $(\!|P|\!)_C \equiv (\!|Q|\!)_C$.

Proof. Every axiom of the structural congruence in PCP maps directly to the axiom of the same name in PGV.

Theorem 6 (Operational Correspondence, Completeness, $(\!|\cdot|\!)_C$).
If $P \vdash \Gamma$ *and* $P \Longrightarrow Q$, *then* $(\!|P|\!)_C \longrightarrow_C^+ (\!|Q|\!)_C$.

Proof. By induction on the derivation of $P \Longrightarrow Q$.

Case (E-Link).

$$\begin{array}{ccc}(\nu xx')(w{\leftrightarrow}x \parallel P) & \xrightarrow{\Longrightarrow} & P\{w/x'\} \\ \downarrow{\scriptstyle (\!|\cdot|\!)_C} & & \\ (\nu xx')(\circ\ \mathbf{link}\ (w,x) \parallel (\!|P|\!)_C) & & \downarrow{\scriptstyle (\!|\cdot|\!)_C} \\ \downarrow{\scriptstyle \longrightarrow_C^+} & & \\ (\!|P|\!)_C\{w/x'\} & \xrightarrow{=} & (\!|P\{w/x'\}|\!)_C\end{array}$$

Case (E-Send).

$$\begin{array}{ccc}(\nu xx')(x[y].P \parallel x'(y').Q) & \xrightarrow{\Longrightarrow} & (\nu xx')(\nu yy')(P \parallel Q) \\ \downarrow{\scriptstyle (\!|\cdot|\!)_M} & & \\ (\nu xx')\left(\circ\begin{pmatrix}\mathbf{let}\ (y,y') = \mathbf{new\ in} \\ \mathbf{let}\ x = \mathbf{send}\ (y,x)\ \mathbf{in}\ (\!|P|\!)_M\end{pmatrix} \parallel \circ\ \mathbf{let}\ (y',x') = \mathbf{recv}\ x'\ \mathbf{in}\ (\!|Q|\!)_M\right) & & \downarrow{\scriptstyle (\!|\cdot|\!)_C} \\ \downarrow{\scriptstyle \longrightarrow_C^+} & & \\ (\nu xx')(\nu yy')(\circ\ (\!|P|\!)_M \parallel \circ\ (\!|Q|\!)_M) & \xrightarrow[\text{(by lemma 8)}]{\longrightarrow_C^\star} & (\nu xx')(\nu yy')((\!|P|\!)_C \parallel (\!|Q|\!)_C)\end{array}$$

Case (E-Close).

$$\begin{array}{ccc}(\nu xx')(x[].P \parallel x'().Q) & \xrightarrow{\Longrightarrow} & P \parallel Q \\ \downarrow{\scriptstyle (\!|\cdot|\!)_M} & & \\ (\nu xx')(\circ\ \mathbf{close}\ x; (\!|P|\!)_M \parallel \circ\ \mathbf{wait}\ x'; (\!|Q|\!)_M) & & \downarrow{\scriptstyle (\!|\cdot|\!)_C} \\ \downarrow{\scriptstyle \longrightarrow_C^+} & & \\ \circ\ (\!|P|\!)_M \parallel \circ\ (\!|Q|\!)_M & \xrightarrow[\text{(by lemma 8)}]{\longrightarrow_C^\star} & (\!|P|\!)_C \parallel (\!|Q|\!)_C\end{array}$$

Case (E-Select-Inl).

$$\begin{array}{ccc}
(\nu x x')(x \triangleleft \mathrm{inl}.P \parallel x \triangleright \{\mathrm{inl} : Q; \mathrm{inr} : R\}) & \xrightarrow{\quad\Longrightarrow\quad} & (\nu x x')(P \parallel Q) \\
\downarrow (\!|\cdot|\!)_M & & \\
(\nu x x') \left(\begin{array}{l} \circ\ \mathbf{let}\ x = \mathbf{select}\ \mathbf{inl}\ x\ \mathbf{in}\ (\!|P|\!)_M \parallel \\ \circ\ \mathbf{offer}\ x'\ \{\mathbf{inl}\ x' \mapsto (\!|Q|\!)_M; \mathbf{inr}\ x' \mapsto (\!|R|\!)_M\} \end{array} \right) & & \downarrow (\!|\cdot|\!)_C \\
\downarrow \longrightarrow_C^+ & & \\
(\nu x x')(\circ\ (\!|P|\!)_M \parallel \circ\ (\!|Q|\!)_M) & \xrightarrow[\text{(by lemma 8)}]{\longrightarrow_C^\star} & (\nu x x')((\!|P|\!)_C \parallel (\!|Q|\!)_C)
\end{array}$$

Case (E-Select-Inr).

$$\begin{array}{ccc}
(\nu x x')(x \triangleleft \mathrm{inr}.P \parallel x \triangleright \{\mathrm{inl} : Q; \mathrm{inr} : R\}) & \xrightarrow{\quad\Longrightarrow\quad} & (\nu x x')(P \parallel R) \\
\downarrow (\!|\cdot|\!)_M & & \\
(\nu x x') \left(\begin{array}{l} \circ\ \mathbf{let}\ x = \mathbf{select}\ \mathbf{inr}\ x\ \mathbf{in}\ (\!|P|\!)_M \parallel \\ \circ\ \mathbf{offer}\ x'\ \{\mathbf{inl}\ x' \mapsto (\!|Q|\!)_M; \mathbf{inr}\ x' \mapsto (\!|R|\!)_M\} \end{array} \right) & & \downarrow (\!|\cdot|\!)_C \\
\downarrow \longrightarrow_C^+ & & \\
(\nu x x')(\circ\ (\!|P|\!)_M \parallel \circ\ (\!|R|\!)_M) & \xrightarrow[\text{(by lemma 8)}]{\longrightarrow_C^\star} & (\nu x x')((\!|P|\!)_C \parallel (\!|R|\!)_C)
\end{array}$$

Case (E-LiftRes). By the induction hypothesis and E-LiftC.

Case (E-LiftPar). By the induction hypotheses and E-LiftC.

Case (E-LiftSC). By the induction hypothesis, E-LiftSC, and lemma 18.

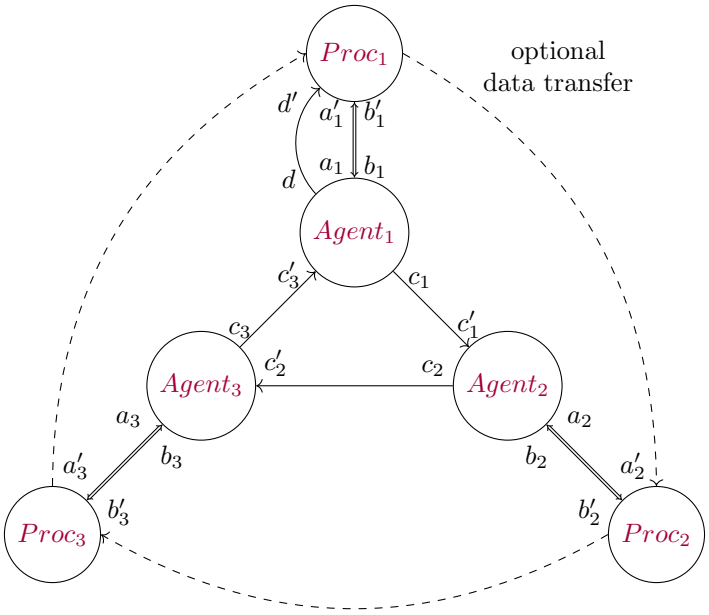
C Milner's Cyclic Scheduler

As an example of a deadlock-free cyclic process, Dardha and Gay [12] introduce an implementation of Milner's cyclic scheduler [32] in Priority CP. We reproduce that scheduler here, and show its translation to Priority GV.

Example 3 (Milner's Cyclic Scheduler, PCP). A set of processes $Proc_i$, $1 \leq i \leq n$, is scheduled to perform some tasks in cyclic order, starting with $Proc_1$, ending with $Proc_n$, and notifying $Proc_1$ when all processes have finished.

Our scheduler *Sched* consists of set of agents $Agent_i$, $1 \leq i \leq n$, each representing their respective process. Each process $Proc_i$ waits for the signal to start their task on a'_i, and signals completion on b'_i. Each agent signals their process to start on a_i, waits for their process to finish on b_i, and then signals for the next agent to continue on c_i. The agent $Agent_1$ initiates, then waits for every other process to finish, and signals $Proc_1$ on d. Every other agent $Agent_i$, $2 \leq i \leq n$ waits on c'_{i-1} for the signal to start. Each of the channels in the scheduler is of a terminated type, and is merely used to synchronise.

Below is a diagram of our scheduler instantiated with three processes:



We implement the scheduler as follows, using $\prod_I P_i$ to denote the parallel composition of the processes P_i, $i \in I$, and $P[Q]$ to denote the plugging of Q in the one-hole process-context P. The process-contexts P_i represent the computations performed by each process $Proc_i$. The process-contexts Q_i represent any post-processing, and any possible data transfer from $Proc_i$ to $Proc_{i+1}$. Finally, Q_1 should contain $d'()$.

$$
\begin{aligned}
Sched &\triangleq (\nu a_1 a'_1)\dots(\nu a_n a'_n)(\nu b_1 b'_1)\dots(\nu b_n b'_n)(\nu c_1 c'_1)\dots(\nu c_n c'_n)(\nu d d') \\
&\quad (Proc_1 \parallel Agent_1 \parallel \textstyle\prod_{2\le i\le n}(Proc_i \parallel c'_{i-1}().Agent_i)) \\
Agent_1 &\triangleq a_i[].b_i().c_i[].c'_n().d[].\mathbf{0} \\
Agent_i &\triangleq a_i[].b_i().c_i[].\mathbf{0} \\
Proc_i &\triangleq a'_i().P_i[b'_i[].Q_i]
\end{aligned}
$$

Example 4 (Milner's Cyclic Scheduler, PGV). The PGV scheduler has exactly the same behaviour as the PCP version in example 3. It is implemented as follows, using $\prod_I \mathcal{C}_i$ to denote the parallel composition of the processes $\mathcal{C}_i$, $i \in I$, and $M[N]$ to denote the plugging of N in the one-hole term-context M. For simplicity, we let **sched** be a configuration. The terms M_i represent the computations performed by each process $\mathbf{proc}_i$. The terms N_i represent any post-processing, and any possible data transfer from $\mathbf{proc}_i$ to $\mathbf{proc}_{i+1}$. Finally, N_1 should contain $\mathbf{wait}\ d'$.

$$
\begin{aligned}
\mathbf{sched} &\triangleq (\nu a_1 a'_1)\dots(\nu a_n a'_n)(\nu b_1 b'_1)\dots(\nu b_n b'_n)(\nu c_1 c'_1)\dots(\nu c_n c'_n)(\nu d d') \\
&\quad (\ \phi\ \mathbf{proc}_1 \parallel \circ\, \mathbf{agent}_1; \mathbf{wait}\ c'_n; \mathbf{close}\ d \\
&\quad \parallel \textstyle\prod_{2\le i\le n}(\circ\, \mathbf{proc}_i \parallel \circ\, \mathbf{wait}\ c'_{i-1}; \mathbf{agent}_i)\ ) \\
\mathbf{agent}_i &\triangleq \mathbf{close}\ a_i; \mathbf{wait}\ b_i; \mathbf{close}\ c_i \\
\mathbf{proc}_i &\triangleq \mathbf{wait}\ a'_i; M_i[\mathbf{close}\ b'_i; N_i]
\end{aligned}
$$

If $(\!|P_i|\!)_M = M_i$ and $(\!|Q_i|\!)_M = N_i$, then the translation of $Sched$ (example 3), $(\!|Sched|\!)_c$, is exactly **sched** (example 4).