



A New Linear Logic for Deadlock-Free Session-Typed Processes

Ornela Dardha^(✉)  and Simon J. Gay

School of Computing Science, University of Glasgow, Glasgow, UK
{Ornela.Dardha,Simon.Gay}@glasgow.ac.uk

Abstract. The π -calculus, viewed as a core concurrent programming language, has been used as the target of much research on type systems for concurrency. In this paper we propose a new type system for deadlock-free session-typed π -calculus processes, by integrating two separate lines of work. The first is the propositions-as-types approach by Caires and Pfenning, which provides a linear logic foundation for session types and guarantees deadlock-freedom by forbidding cyclic process connections. The second is Kobayashi’s approach in which types are annotated with priorities so that the type system can check whether or not processes contain genuine cyclic dependencies between communication operations. We combine these two techniques for the first time, and define a new and more expressive variant of classical linear logic with a proof assignment that gives a session type system with Kobayashi-style priorities. This can be seen in three ways: (i) as a new linear logic in which cyclic structures can be derived and a CYCLE-elimination theorem generalises CUT-elimination; (ii) as a logically-based session type system, which is more expressive than Caires and Pfenning’s; (iii) as a logical foundation for Kobayashi’s system, bringing it into the sphere of the propositions-as-types paradigm.

1 Introduction

The Curry-Howard correspondence, or propositions-as-types paradigm, provides a canonical logical foundation for functional programming [42]. It identifies types with logical propositions, programs with proofs, and computation with proof normalisation. It was natural to ask for a similar account of concurrent programming, and this question was brought into focus by the discovery of linear logic [24] and Girard’s explicit suggestion that it should have some connection with concurrent computation. Several attempts were made to relate π -calculus processes to the proof nets of classical linear logic [1, 8], and to relate CCS-like processes to the $*$ -autonomous categories that provide semantics for classical linear logic [2]. However, this work did not result in a convincing propositions-as-types framework for concurrency, and did not continue beyond the 1990s.

Supported by the UK EPSRC grant EP/K034413/1, “From Data Types to Session Types: A Basis for Concurrency and Distribution (ABCD)”, and by COST Action IC1201, “Behavioural Types for Reliable Large-Scale Software Systems (BETTY)”.

© The Author(s) 2018

C. Baier and U. Dal Lago (Eds.): FOSSACS 2018, LNCS 10803, pp. 91–109, 2018.

https://doi.org/10.1007/978-3-319-89366-2_5

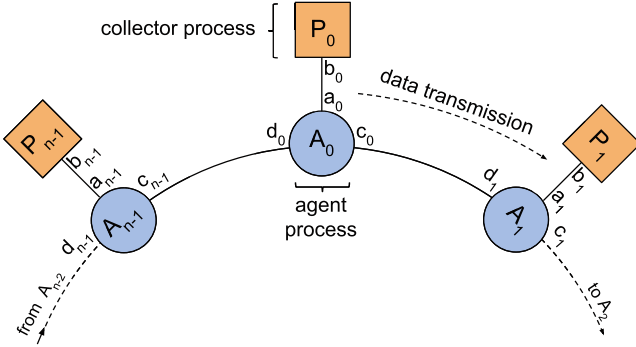


Fig. 1. Cyclic scheduler

Meanwhile, Honda *et al.* [26, 27, 38] developed *session types* as a formalism for statically checking that messages have the correct types and sequence according to a communication protocol. Research on session types developed and matured over several years, eventually inspiring Caires and Pfenning [12] to discover a Curry-Howard correspondence between dual intuitionistic linear logic [7] and a form of π -calculus with session types [38]. Wadler [41] subsequently gave an alternative formulation based on classical linear logic, and related it to existing work on session types for functional languages [23]. The Caires-Pfenning approach has been widely accepted as a propositions-as-types theory of concurrent programming, as well as providing a logical foundation for session types.

Caires and Pfenning’s type system guarantees deadlock-freedom by forbidding cyclic process structures. It provides a logical foundation for deadlock-free session processes, complementing previous approaches to deadlock-freedom in session type systems [9, 15, 21, 22]. The logical approach to session types has been extended in many ways, including features such as dependent types [39], failures and non-determinism [11], sharing and races [6]. All this work relies on the acyclicity condition. However, rejecting cyclic process structures is unnecessarily strict: they are a necessary, but not sufficient, condition for the existence of deadlocked communication operations. As we will show in Example 1 (Fig. 1), there are deadlock-free processes that can naturally be implemented in a cyclic way, but are rejected by Caires and Pfenning’s type system.

Our contribution is to define a new logic, *priority-based linear logic* (PLL), and formulate it as a type system for *priority-based CP* (PCP), which is a more expressive class of processes than Wadler’s CP [41]. This is the first Curry-Howard correspondence that allows cyclic interconnected processes, while still ensuring deadlock-freedom. The key idea is that PLL includes conditions on inter-channel dependencies based on Kobayashi’s type systems [29, 30, 32]. Our work can be viewed in three ways: (i) as a new linear logic in which cyclic proof structures can be derived; (ii) as an extension of Caires-Pfenning type systems so that they accept more processes, while maintaining the strong logical foundation; (iii) as a logical foundation for Kobayashi-style type systems.

An example of a deadlock-free cyclic process is Milner’s well-known scheduler [35], described in the following Example 1.

Example 1 (Cyclic Scheduler, Fig. 1). A set of agents A_0, \dots, A_{n-1} , for $n > 1$, is scheduled to perform a certain task in cyclic order, starting with agent A_0 . For all $i \in \{1, \dots, n-1\}$, agent A_i sends the result of computation to a collector process P_i , before transmitting further data to agent $A_{(i+1) \bmod n}$. At the end of the round, A_0 sends the final result to P_0 . Here we define a finite version of Milner’s scheduler, which executes one round of communication.

$$\begin{aligned} \text{Sched} &\triangleq \dots(\nu a_i b_i) \dots (\nu c_i d_{(i+1) \bmod n}) (A_0 \mid A_1 \mid \dots \mid A_{n-1} \mid P_0 \mid P_1 \mid \dots \mid P_{n-1}) \\ A_0 &\triangleq c_0[\mathbf{n}_0].d_0(x_0).a_0[\mathbf{m}_0].\text{close}_0 \\ A_i &\triangleq d_i(x_i).a_i[\mathbf{m}_i].c_i[\mathbf{n}_i].\text{close}_i \quad i \in \{1, \dots, n-1\} \\ P_i &\triangleq b_i(y_i).Q_i \quad i \in \{0, \dots, n-1\} \end{aligned}$$

Prefix $c_0[\mathbf{n}_0]$ denotes an output on c_0 , and $d_0(x_0)$ an input on d_0 . For now, let \mathbf{m} and \mathbf{n} denote data. Process close_i closes the channels used by A_i : the details of this closure are irrelevant here (however, they are as in processes Q and R in Example 2). Process Q_i uses the message received from A_i , in internal computation. The construct (νab) creates two channel endpoints a and b and binds them together. The system Sched is deadlock-free because A_1, \dots, A_{n-1} each wait for a message from the previous A_i before sending, and A_0 sends the initial message.

Sched is not typable in the original type systems by Caires-Pfenning and Wadler. To do that, it would be necessary to break A_0 into two parallel agents $A'_0 \triangleq c_0[\mathbf{n}_0].\text{close}_{c_0}$ and $A''_0 \triangleq d_0(x_0).a_0[\mathbf{m}_0].\text{close}_{d_0, a_0}$. This changes the design of the system, yielding a different one. Moreover, if the scheduler continues into a second round of communication, this redesign is not possible because of the potential dependency from the input on d_0 to the next output on c_0 . However, Sched is typable in PCP; we will show the type assignment at the end of Sect. 2.

There is a natural question at this point: *given that the cyclic scheduler is deadlock-free, is it possible to encode its semantics in CP, thus eliminating the need for PCP?* It is possible to define a centralised agent A that communicates with all the collectors P_i , resulting in a system that is semantically equivalent to our Sched . However, such an encoding has a global character, and changes the structure of the overall system from distributed to centralised. In programming terms, it corresponds to changing the software design, as we pointed out in Example 1, and ultimately the software architecture, which is not always desirable or even feasible. The aim of PCP is to generalise CP so that deadlock-free processes can be constructed with their natural structure. We would want any encoding of PCP into CP to be structure-preserving, which would mean translating the CYCLE rule (given in Fig. 2) homomorphically; this is clearly impossible.

Contributions and Structure of the Paper. In Sect. 2 we define priority-based linear logic (PLL), which extends classical linear logic (CLL) with priori-

ties attached to propositions. These priorities are based on Kobayashi’s annotations for deadlock freedom [32]. By following the propositions-as-types paradigm, we define a term assignment for PLL proofs, resulting in priority-based classical processes (PCP), which extends Wadler’s CP [41] with MIX and CYCLE rules (Fig. 2). In Sect. 3 we define an operational semantics for PCP. In Sect. 4 we prove CYCLE-elimination (Theorem 1) for PLL, analogous to the standard CUT-elimination theorem for CLL. Consequently, the results for PCP are subject reduction (Theorem 2), top-level deadlock-freedom (Theorem 3), and full deadlock-freedom for closed processes (Theorem 4). In Sect. 5 we discuss related work and conclude the paper.

2 PCP: Classical Processes with MIX and CYCLE

Priority-based CP (PCP) follows the style of Wadler’s Classical Processes (CP) [41], with details inspired by Carbone *et al.* [14] and Caires and Pérez [11].

Types. We start with types, which are based on CLL propositions. Let A, B range over types, given in Definition 1. Let $\circ, \kappa \in \mathbb{N} \cup \{\omega\}$ range over *priorities*, which are used to annotate types. Let ω be a special element such that $\circ < \omega$ for all $\circ \in \mathbb{N}$. Often, we will omit ω . We will explain priorities later in this section.

Definition 1 (Types). *Types (A, B) are given by:*

$$A, B ::= \perp^\circ \mid \mathbf{1}^\circ \mid A \otimes^\circ B \mid A \wp^\circ B \mid \oplus^\circ \{l_i : A_i\}_{i \in I} \mid \&^\circ \{l_i : A_i\}_{i \in I} \mid ?^\circ A \mid !^\circ A$$

\perp° and $\mathbf{1}^\circ$ are associated with channel endpoints that are ready to be closed. $A \otimes^\circ B$ (respectively, $A \wp^\circ B$) is associated with a channel endpoint that first outputs (respectively, inputs) a channel of type A and then proceeds as B . $\oplus^\circ \{l_i : A_i\}_{i \in I}$ is associated with a channel endpoint over which we can select a label from $\{l_i\}_{i \in I}$, and proceed as A_i . Dually, $\&^\circ \{l_i : A_i\}_{i \in I}$ is associated with a channel endpoint that can offer a set of labelled types. $?^\circ A$ types a collection of clients requesting A . Dually, $!^\circ A$ types a server repeatedly accepting A .

Duality on types is total and is given in Definition 2. It preserves priorities of types.

Definition 2 (Duality). *The duality function $(\cdot)^\perp$ on types is given by:*

$$\begin{aligned} (A \wp^\circ B)^\perp &= A^\perp \otimes^\circ B^\perp & (\perp^\circ)^\perp &= \mathbf{1}^\circ \\ (A \otimes^\circ B)^\perp &= A^\perp \wp^\circ B^\perp & (\mathbf{1}^\circ)^\perp &= \perp^\circ \\ (\&^\circ \{l_i : A_i\}_{i \in I})^\perp &= \oplus^\circ \{l_i : A_i^\perp\}_{i \in I} & ?^\circ A^\perp &= !^\circ A^\perp \\ (\oplus^\circ \{l_i : A_i\}_{i \in I})^\perp &= \&^\circ \{l_i : A_i^\perp\}_{i \in I} & !^\circ A^\perp &= ?^\circ A^\perp \end{aligned}$$

Processes. Let P, Q range over processes, given in Definition 3. Let x, y range over channel endpoints, and \mathbf{m}, \mathbf{n} over channel endpoints of type either \perp° or $\mathbf{1}^\circ$.

Definition 3 (Processes). Processes (P, Q) are given by:

$$\begin{array}{llll}
P, Q ::= x[y].P & (\text{output}) & \mathbf{0} & (\text{inaction}) \\
& x(y).P & (P \mid Q) & (\text{composition}) \\
& x \triangleleft l_j.P & (\nu x^A y)P & (\text{sessionrestriction}) \\
& x \triangleright \{l_i : P_i\}_{i \in I} & (x[\cdot].\mathbf{0}) & (\text{emptyoutput}) \\
& x \rightarrow y^A & (x(\cdot).P) & (\text{emptyinput})
\end{array}$$

Process $x[y].P$ (respectively, $x(y).P$) outputs (respectively, inputs) y on channel endpoint x , and proceeds as P . Process $x \triangleleft l_j.P$ uses x to select l_j from a labelled choice process, typically being $x \triangleright \{l_i : P_i\}_{i \in I}$, and triggers P_j ; labels indexed by the finite set I are pairwise distinct. Process $x \rightarrow y^A$ forwards communications from x to y , the latter having type A . Processes also include the inaction process $\mathbf{0}$, the parallel composition of P and Q , denoted $P \mid Q$, and the double restriction constructor $(\nu x^A y)P$: the intention is that x and y denote dual session channel endpoints in P , and A is the type of x . Processes $x[\cdot].\mathbf{0}$ and $x(\cdot).P$ are the empty output and empty input, respectively. They denote the closure of a session from the viewpoint of each of the two communicating participants.

Notions of bound/free names in processes are standard; we write $\text{fn}(P)$ to denote the set of free names of P . Also, we write $P\{x/z\}$ to denote the (capture-avoiding) substitution of x for the free occurrences of z in P . Finally, we let \tilde{x} , which is different from x , denote a sequence x_1, \dots, x_n for $n > 0$.

Typing Rules. Typing contexts, ranged over by Γ, Δ, Θ , are sets of typing assumptions $x : A$. We write Γ, Δ for union, requiring the contexts to be disjoint. A typing judgement $P \vdash \Gamma$ means “process P is well typed using context Γ ”.

Before presenting the typing rules, we need some auxiliary definitions. Our priorities are based on the annotations used by Kobayashi [32], but simplified to single priorities *à la* Padovani [37]. They obey the following laws:

- (i) An action of priority \circ must be prefixed only by actions of priorities *strictly smaller than* \circ .
- (ii) Communication requires *equal priorities* for the complementary actions.

Definition 4 (Priority). The priority function $\text{pr}(\cdot)$ on types is given by:

$$\begin{array}{ll}
\text{pr}(A \wp^\circ B) = \text{pr}(A \otimes^\circ B) = \circ & \text{pr}(\perp^\circ) = \text{pr}(\mathbf{1}^\circ) = \circ \\
\text{pr}(\oplus^\circ \{l_i : A_i\}_{i \in I}) = \text{pr}(\&^\circ \{l_i : A_i\}_{i \in I}) = \circ & \text{pr}(\?^\circ A) = \text{pr}(!^\circ A) = \circ
\end{array}$$

Definition 5 (Lift). Let $t \in \mathbb{N}$. The lift operator $\uparrow^t(\cdot)$ on types is given by:

$$\begin{array}{ll}
\uparrow^t(A \wp^\circ B) = (\uparrow^t A) \wp^{(\circ+t)} (\uparrow^t B) & \uparrow^t \perp^\circ = \mathbf{1}^{(\circ+t)} \\
\uparrow^t(A \otimes^\circ B) = (\uparrow^t A) \otimes^{(\circ+t)} (\uparrow^t B) & \uparrow^t \mathbf{1}^\circ = \perp^{(\circ+t)} \\
\uparrow^t(\&^\circ \{l_i : A_i\}_{i \in I}) = \&^{(\circ+t)} \{l_i : \uparrow^t A_i\}_{i \in I} & \uparrow^t(\?^\circ A) = \?^{(\circ+t)} (\uparrow^t A) \\
\uparrow^t(\oplus^\circ \{l_i : A_i\}_{i \in I}) = \oplus^{(\circ+t)} \{l_i : \uparrow^t A_i\}_{i \in I} & \uparrow^t(!^\circ A) = !^{(\circ+t)} (\uparrow^t A)
\end{array}$$

We assume $\omega + t = \omega$ for all $t \in \mathbb{N}$.

The operator \uparrow^t is extended component-wise to typing contexts: $\uparrow^t \Gamma$.

$$\begin{array}{c}
\frac{}{x \rightarrow y^A \vdash x : A^\perp, y : A} \text{Ax} \quad \frac{P \vdash \Gamma \quad Q \vdash \Delta}{P \mid Q \vdash \Gamma, \Delta} \text{MIX} \quad \frac{P \vdash \Gamma, x : A, y : A^\perp}{(\nu x^A y) P \vdash \Gamma} \text{CYCLE} \\
\\
\frac{}{\mathbf{0} \vdash \emptyset} \emptyset \quad \frac{}{x[\cdot] : \mathbf{0} \vdash x : \mathbf{1}^\circ} \mathbf{1} \quad \frac{P \vdash \Gamma \quad \circ < \text{pr}(\Gamma)}{x().P \vdash x : \perp^\circ, \Gamma} \perp \\
\\
\frac{P \vdash \Gamma, y : A, x : B \quad \circ < \text{pr}(\Gamma)}{x(y).P \vdash \Gamma, x : A \wp^\circ B} \wp \quad \frac{P \vdash \Gamma, y : A, x : B \quad \circ < \text{pr}(\Gamma)}{x[y].P \vdash \Gamma, x : A \otimes^\circ B} \otimes \\
\\
\frac{\forall i \in I. (P_i \vdash \Gamma, x : A_i) \quad \circ < \text{pr}(\Gamma)}{x \triangleright \{l_i : P_i\}_{i \in I} \vdash \Gamma, x : \&^\circ \{l_i : A_i\}_{i \in I}} \& \quad \frac{P \vdash \Gamma, x : A_j \quad j \in I \quad \circ < \text{pr}(\Gamma)}{x \triangleleft l_j . P \vdash \Gamma, x : \oplus^\circ \{l_i : A_i\}_{i \in I}} \oplus \\
\\
\frac{P \vdash ?\Gamma, y : A \quad \circ < \text{pr}(?\Gamma)}{!x(y).P \vdash ?\Gamma, x : !^\circ A} ! \quad \frac{P \vdash \Gamma, y : A \quad \circ < \text{pr}(\Gamma)}{?x[y].P \vdash \Gamma, x : ?^\circ A} ? \\
\\
\frac{P \vdash \Gamma}{P \vdash \Gamma, x : ?^\circ A} \text{W} \quad \frac{P \vdash \Gamma, y : ?^\kappa A, z : ?^{\kappa'} A \quad \circ \leq \kappa \quad \circ \leq \kappa' \quad \circ < \text{pr}(\Gamma)}{P\{x/y, x/z\} \vdash \Gamma, x : ?^\circ A} \text{C}
\end{array}$$

Fig. 2. Typing rules for PCP.

The typing rules are given in Fig. 2. Ax states that the forwarding process $x \rightarrow y^A$ is well typed if x and y have dual types, respectively A^\perp and A . MIX types the parallel composition of two processes P and Q in the union of their disjoint typing contexts. CYCLE is our key typing rule; it states that the restriction process is well typed, if the endpoints x and y have dual types, respectively A and A^\perp . By Definition 2, A and A^\perp also have the same priorities, enforcing law (ii) above. In classical logic this rule would be unsound, but in PLL it allows deadlock-free cycles. Rule \emptyset states that inaction is well typed in the empty context. Rules $\mathbf{1}$ and \perp type channel closure actions from the viewpoint of each participant. Rule \wp (respectively \otimes) types an input process $x(y).P$ (respectively, output process $x[y].P$), with y bound and x of type $A \wp^\circ B$ (respectively, $A \otimes^\circ B$). The priority \circ is strictly smaller than any priorities in the continuation process P , enforcing law (i) above. This is captured by $\circ < \text{pr}(\Gamma)$ in the premises of both rules, abbreviating “for all $z \in \text{dom}(\Gamma)$, $\circ < \text{pr}(\Gamma(z))$ ”. Rules $\&$ and \oplus type external and internal choice, respectively, and follow the previous two rules. Rule $!$ types a server and states that if P communicates along y following protocol A , then $!x(y).P$ communicates along x following protocol $!^\circ A$. The three remaining rules type different numbers of clients. Rule $?$ is for a single client: if P communicates along y following A , then $?x[y].P$ communicates along x following $?^\circ A$. Rule W is for no client: if P does not communicate along any channel following A , then it may be regarded as communicating along x following $?^\circ A$, for some priority \circ . Rule C is for multiple clients: if P communicates along y following $?^\kappa A$, and z following protocol $?^{\kappa'} A$, then $P\{x/y, x/z\}$ communicates along a single channel x following $?^\circ A$, where $\circ \leq \kappa$ and $\circ \leq \kappa'$. The last two conditions are necessary to deal with some cases in the proof of CYCLE-elimination (Theorem 1).

Lifting preserves typability, by an easy induction on typing derivations.

Lemma 1. *If $P \vdash \Gamma$ then $P \vdash \uparrow^t \Gamma$.*

We will use this result in the form of an admissible rule: $\frac{P \vdash \Gamma}{P \vdash \uparrow^t \Gamma} \uparrow^t$

The Design of PCP. We have included MIX and CYCLE, which allow derivation of both the standard CUT and the MULTICUT by Abramsky *et al.* [2].

$$\left. \begin{array}{c} \frac{\vdash \Gamma, A_1, \dots, A_n \quad \vdash \Delta, A_1^\perp, \dots, A_n^\perp}{\vdash \Gamma, \Delta, A_1, \dots, A_n, A_1^\perp, \dots, A_n^\perp} \text{MIX} \\ \frac{\vdash \Gamma, \Delta, A_1, \dots, A_n, A_1^\perp, \dots, A_n^\perp}{\vdash \Gamma, \Delta} \text{CYCLE}^n \end{array} \right\} \text{MULTICUT}$$

Conversely, MIX is the nullary case of MULTICUT, and CYCLE can be derived from AX and MULTICUT:

$$\left. \begin{array}{c} \frac{\vdash \Gamma, A, A^\perp \quad \overline{\vdash A^\perp, A}}{\vdash \Gamma} \text{AX} \\ \text{MULTICUT} \end{array} \right\} \text{CYCLE}$$

Having included MIX, we choose CYCLE instead of MULTICUT, as CYCLE is more primitive.

In the presence of MIX and CYCLE, there is an isomorphism between $A \otimes B$ and $A \wp B$ in CLL. Both $A \otimes B \multimap A \wp B$ and $A \wp B \multimap A \otimes B$, are derivable, where $C \multimap D \triangleq C^\perp \wp D$ in CLL. Equivalently, both $(A^\perp \wp B^\perp) \wp (A \wp B)$ and $(A^\perp \otimes B^\perp) \wp (A \otimes B)$ are derivable. For simplicity, let $\text{pr}(A) = \text{pr}(B) = \omega$; by duality also $\text{pr}(A^\perp) = \text{pr}(B^\perp) = \omega$.

$$\begin{array}{c} \frac{\overline{\vdash A^\perp, A} \quad \overline{\vdash B^\perp, B}}{\vdash A^\perp, B^\perp, A, B} \text{MIX} \\ \frac{\quad \text{o}_1 < \omega}{\vdash A^\perp \wp^{\text{o}_1} B^\perp, A, B} \wp \\ \frac{\quad \text{o}_2 < \text{o}_1}{\vdash A^\perp \wp^{\text{o}_1} B^\perp, A \wp^{\text{o}_2} B} \wp \\ \frac{\quad}{\vdash (A^\perp \wp^{\text{o}_1} B^\perp) \wp^{\text{o}} (A \wp^{\text{o}_2} B)} \wp \end{array} \quad \begin{array}{c} \frac{\overline{\vdash A^\perp, A} \quad \overline{\vdash B^\perp, B}}{\vdash A^\perp, B^\perp, A, B} \text{MIX} \\ \frac{\quad \text{o}_1 < \omega}{\vdash A^\perp \otimes^{\text{o}_1} B^\perp, A, B} \otimes \\ \frac{\quad \text{o}_2 < \omega}{\vdash A^\perp, B^\perp, A \otimes^{\text{o}_2} B} \otimes \\ \frac{\quad}{\vdash A^\perp \otimes^{\text{o}_1} B^\perp, A \otimes^{\text{o}_2} B, A^\perp, A, B^\perp, B} \text{MIX} \\ \frac{\quad}{\vdash A^\perp \otimes^{\text{o}_1} B^\perp, A \otimes^{\text{o}_2} B} \text{CYCLE}^2 \\ \frac{\quad}{\vdash (A^\perp \otimes^{\text{o}_1} B^\perp) \wp^{\text{o}} (A \otimes^{\text{o}_2} B)} \wp \end{array}$$

The above derivations *without* priorities show the isomorphism between $A \otimes B$ and $A \wp B$ in CLL, which does not hold in our PLL, in particular as $\text{o}_1 \neq \text{o}_2$. The distinction between \otimes and \wp , preserves the distinction between output and input in the term assignment. However, to simplify derivations, both typing rules (Fig. 2) have the same form. The usual tensor rule, where there are two separate derivations in the premise rather than just one, is derivable by using MIX.

Our type system performs priority-checking. Priorities can be inferred, as in Kobayashi's type system [32] and the tool TyPiCal [28]. We have opted for priority checking over priority inference, as the presentation is more elegant.

The following two examples illustrate the use of priorities. We first establish the structure of the typing derivation, then calculate the priorities. We conclude the section by showing the typing for the cyclic scheduler from Sect. 1.

Example 2 (Cyclic process: deadlock-free). Consider the following process

$$P \triangleq (\nu x_1 y_1)(\nu x_2 y_2)[x_1(v).x_2(w).R \mid y_1[\mathbf{n}].y_2[\mathbf{n}'].Q]$$

where $R \triangleq x_1().v().x_2().w().\mathbf{0}$ and $Q \triangleq y_1[.].\mathbf{0} \mid \mathbf{n}[.].\mathbf{0} \mid y_2[.].\mathbf{0} \mid \mathbf{n}'[.].\mathbf{0}$. First, we show the typing derivation for the left-hand side of the parallel, $x_1(v).x_2(w).R$:

$$\frac{\frac{\frac{\frac{\mathbf{0} \vdash \emptyset \quad \kappa_4 < \kappa_3 < \kappa_2 < \kappa_1 \quad \emptyset}{\mathbf{0} < \kappa_4} \quad \perp^4}{R \vdash x_1 : \perp^{\kappa_4} \quad v : \perp^{\kappa_3}, x_2 : \perp^{\kappa_2}, w : \perp^{\kappa_1}} \quad \perp^4}{x_2(w).R \vdash x_1 : \perp^{\kappa_4}, v : \perp^{\kappa_3}, x_2 : \perp^{\kappa_1} \wp^{\circ_1} \perp^{\kappa_2}} \quad \wp}{x_1(v).x_2(w).R \vdash x_2 : \perp^{\kappa_1} \wp^{\circ_1} \perp^{\kappa_2}, x_1 : \perp^{\kappa_3} \wp^{\circ_2} \perp^{\kappa_4}} \quad \wp \quad \circ_2 < \circ_1 \quad \wp \quad (1)$$

Now, the typing derivation for the right-hand side of the parallel, $y_1[\mathbf{n}].y_2[\mathbf{n}'].Q$, and recall that $\kappa_4 < \kappa_3 < \kappa_2 < \kappa_1$:

$$\frac{\frac{\frac{\frac{y_1[.].\mathbf{0} \vdash y_1 : \mathbf{1}^{\kappa_4} \quad \mathbf{1} \quad \mathbf{n}[.].\mathbf{0} \vdash \mathbf{n} : \mathbf{1}^{\kappa_3} \quad \mathbf{1} \quad y_2[.].\mathbf{0} \vdash y_2 : \mathbf{1}^{\kappa_2} \quad \mathbf{1} \quad \mathbf{n}'[.].\mathbf{0} \vdash \mathbf{n}' : \mathbf{1}^{\kappa_1} \quad \mathbf{1}}{y_1[.].\mathbf{0} \mid \mathbf{n}[.].\mathbf{0} \mid y_2[.].\mathbf{0} \mid \mathbf{n}'[.].\mathbf{0} \vdash y_1 : \mathbf{1}^{\kappa_4}, \mathbf{n} : \mathbf{1}^{\kappa_3}, y_2 : \mathbf{1}^{\kappa_2}, \mathbf{n}' : \mathbf{1}^{\kappa_1}} \quad \circ_3 < \kappa_4 \quad \text{Mix}^3}{y_2[\mathbf{n}'].Q \vdash y_1 : \mathbf{1}^{\kappa_4}, \mathbf{n} : \mathbf{1}^{\kappa_3}, y_2 : \mathbf{1}^{\kappa_1} \otimes^{\circ_3} \mathbf{1}^{\kappa_2} \quad \circ_4 < \circ_3 \quad \otimes}{y_1[\mathbf{n}].y_2[\mathbf{n}'].Q \vdash y_2 : \mathbf{1}^{\kappa_1} \otimes^{\circ_3} \mathbf{1}^{\kappa_2}, y_1 : \mathbf{1}^{\kappa_3} \otimes^{\circ_4} \mathbf{1}^{\kappa_4}} \quad \otimes \quad \otimes \quad (2)$$

Finally, the typing derivation for process P is as follows:

$$\frac{\frac{\frac{(1) \quad (2)}{x_1(v).x_2(w).R \mid y_1[\mathbf{n}].y_2[\mathbf{n}'].Q \vdash \quad \text{Mix}}{x_2 : \perp^{\kappa_1} \wp^{\circ_1} \perp^{\kappa_2}, x_1 : \perp^{\kappa_3} \wp^{\circ_2} \perp^{\kappa_4}, y_2 : \mathbf{1}^{\kappa_1} \otimes^{\circ_3} \mathbf{1}^{\kappa_2}, y_1 : \mathbf{1}^{\kappa_3} \otimes^{\circ_4} \mathbf{1}^{\kappa_4}} \quad \circ_1 = \circ_3}{(\nu x_2 y_2)[x_1(v).x_2(w).R \mid y_1[\mathbf{n}].y_2[\mathbf{n}'].Q] \vdash \quad \text{CYCLE}} \quad \text{CYCLE}}{(\nu x_1 y_1)(\nu x_2 y_2)[x_1(v).x_2(w).R \mid y_1[\mathbf{n}].y_2[\mathbf{n}'].Q] \vdash \emptyset \quad \text{CYCLE}} \quad \text{CYCLE}$$

The system of equations

$$\circ_2 < \circ_1 \quad \circ_4 < \circ_3 \quad \circ_1 = \circ_3 \quad \circ_2 = \circ_4$$

can be solved by the assignment $\circ_1 = \circ_3 = 1$ and $\circ_2 = \circ_4 = 0$.

Example 3 (Cyclic process: deadlocked!). Now consider the process

$$P' = (\nu x_1 y_1)(\nu x_2 y_2)[x_1(v).x_2(w).R \mid y_2[\mathbf{n}'].y_1[\mathbf{n}].Q]$$

where $R = x_1().v().x_2().w().\mathbf{0}$ and $Q = y_1[].\mathbf{0} \mid \mathbf{n}[].\mathbf{0} \mid y_2[].\mathbf{0} \mid \mathbf{n}'[].\mathbf{0}$. Notice that the order of actions on channels y_1 and y_2 is now swapped, thus causing a deadlock! If we tried to construct a typing derivation for process P' , we would have for the right-hand side of the parallel the following:

$$\frac{\frac{\frac{y_1[].\mathbf{0} \vdash y_1 : \mathbf{1}^{\kappa_4} \quad \mathbf{1}}{y_1[].\mathbf{0} \mid \mathbf{n}[].\mathbf{0} \vdash y_1 : \mathbf{1}^{\kappa_4}, \mathbf{n} : \mathbf{1}^{\kappa_3}} \quad \mathbf{1}}{y_1[\mathbf{n}].Q \vdash \mathbf{n}' : \mathbf{1}^{\kappa_1}, y_2 : \mathbf{1}^{\kappa_2}, y_1 : \mathbf{1}^{\kappa_3} \otimes^{\mathbf{o}_4} \mathbf{1}^{\kappa_4} \quad \mathbf{o}_3 < \mathbf{o}_4} \quad \otimes}{y_2[\mathbf{n}'].y_1[\mathbf{n}].Q \vdash y_1 : \mathbf{1}^{\kappa_3} \otimes^{\mathbf{o}_4} \mathbf{1}^{\kappa_4}, y_2 : \mathbf{1}^{\kappa_1} \otimes^{\mathbf{o}_3} \mathbf{1}^{\kappa_2}} \quad \otimes}{\frac{\frac{y_2[].\mathbf{0} \vdash y_1 : \mathbf{1}^{\kappa_2} \quad \mathbf{1} \quad \mathbf{n}'[].\mathbf{0} \vdash \mathbf{n}' : \mathbf{1}^{\kappa_1} \quad \mathbf{1}}{y_2[].\mathbf{0} \mid \mathbf{n}'[].\mathbf{0} \vdash y_1 : \mathbf{1}^{\kappa_2}, \mathbf{n}' : \mathbf{1}^{\kappa_1}} \quad \mathbf{1}}{y_1[].\mathbf{0} \mid \mathbf{n}[].\mathbf{0} \mid y_2[].\mathbf{0} \mid \mathbf{n}'[].\mathbf{0} \vdash y_1 : \mathbf{1}^{\kappa_4}, \mathbf{n} : \mathbf{1}^{\kappa_3}, y_2 : \mathbf{1}^{\kappa_2}, \mathbf{n}' : \mathbf{1}^{\kappa_1} \quad \mathbf{o}_4 < \kappa_4} \quad \text{MIX}^3} \quad \otimes}$$

Then, the system of equations

$$\mathbf{o}_2 < \mathbf{o}_1 \quad \mathbf{o}_3 < \mathbf{o}_4 \quad \mathbf{o}_1 = \mathbf{o}_3 \quad \mathbf{o}_2 = \mathbf{o}_4$$

has no solution because it requires $\mathbf{o}_2 < \mathbf{o}_3$ and $\mathbf{o}_3 < \mathbf{o}_2$, which is impossible.

Example 1 continued (Cyclic Scheduler)

$$\begin{aligned} \text{Sched} &\triangleq \dots(\nu a_i b_i) \dots (\nu c_i d_{(i+1) \bmod n}) (A_0 \mid A_1 \mid \dots \mid A_{n-1} \mid P_0 \mid P_1 \mid \dots \mid P_{n-1}) \\ A_0 &\triangleq c_0[\mathbf{n}_0].d_0(x_0).a_0[\mathbf{m}_0].\text{close}_0 \\ A_i &\triangleq d_i(x_i).a_i[\mathbf{m}_i].c_i[\mathbf{n}_i].\text{close}_i \quad i \in \{1, \dots, n-1\} \\ P_i &\triangleq b_i(y_i).Q_i \quad i \in \{0, \dots, n-1\} \end{aligned}$$

By applying the typing rules in Fig. 2 we can derive $\text{Sched} \vdash \emptyset$, since it is a closed process, and assign the following types and priorities:

$$\begin{array}{llll} c_0 : \mathbf{1} \otimes^0 \mathbf{1} & d_0 : \perp \wp^{2(n-1)} \perp & a_0 : \mathbf{1} \otimes^{2(n-1)+1} \mathbf{1} & \text{for } A_0 \\ d_i : \perp \wp^{2i-2} \perp & a_i : \mathbf{1} \otimes^{2i-1} \mathbf{1} & c_i : \mathbf{1} \otimes^{2i} \mathbf{1} & \text{for } A_i, 0 < i < n \\ b_0 : \perp \wp^{2(n-1)+1} \perp & b_i : \perp \wp^{2i-1} \perp & & \text{for } P_0 \text{ and } P_i, 0 < i < n \end{array}$$

The priorities of types \perp and $\mathbf{1}$ could be easily assigned as Example 2. As the priority of d_{i+1} is $2(i+1) - 2 = 2i$, we can connect it to a_i with a `Cycle`.

3 Operational Semantics of PCP

In this section we define structural equivalence, the principal β -reduction rules and commuting conversions. The detailed derivations can be found in [18].

We define structural equivalence to be the smallest congruence relation satisfying the following axioms. `SC-AX-SWP` allows swapping channels in the forwarding process. `SC-AX-Cycle` states that cycle applied to a forwarding process is equivalent to inaction. This allows elimination of unnecessary cycles. Axioms `SC-MIX-NIL`, `SC-MIX-COMM` and `SC-MIX-ASC` state that parallel composition uses the inaction as the neutral element and is commutative and associative. `SC-Cycle-EXT` is the standard scope extrusion rule. `SC-Cycle-SWP` allows swapping channels and `SC-Cycle-COMM` states the commutativity of restriction¹.

¹ Note that associativity of restriction is derived from `SC-MIX-COMM` and `SC-Cycle-COMM`.

SC-AX-SWP	$x \rightarrow y^A \vdash x : A^\perp, y : A \equiv y \rightarrow x^{A^\perp} \vdash x : A^\perp, y : A$
SC-AX-CYCLE	$(\nu x^{A^\perp} y) x \rightarrow y^A \vdash \emptyset \equiv \mathbf{0} \vdash \emptyset$
SC-MIX-NIL	$\mathbf{0} \mid P \vdash \Gamma \equiv P \vdash \Gamma$
SC-MIX-COMM	$P \mid Q \vdash \Gamma, \Delta \equiv Q \mid P \vdash \Gamma, \Delta$
SC-MIX-ASC	$P \mid (Q \mid R) \vdash \Gamma, \Delta, \Theta \equiv (P \mid Q) \mid R \vdash \Gamma, \Delta, \Theta$
SC-CYCLE-EXT	$(\nu x^A y)(P \mid Q) \vdash \Gamma, \Delta \equiv P \mid (\nu x^A y)Q \vdash \Gamma, \Delta \quad x, y \notin \text{fn}(P)$
SC-CYCLE-SWP	$(\nu x^A y)P \vdash \Gamma \equiv (\nu y^{A^\perp} x)P \vdash \Gamma$
SC-CYCLE-COMM	$(\nu x^A y)(\nu z^B w)P \vdash \Gamma \equiv (\nu z^B w)(\nu x^A y)P \vdash \Gamma$

The core of the operational semantics consists of β -reductions. In π -calculus terms these are communication steps; in logical terms they are CYCLE-elimination steps. $\beta_{\otimes \wp}$ is given in Fig. 3 to illustrate priorities. It simplifies a cycle connecting x of type $A \otimes B$ and y of type $A \wp B$, which corresponds to communication between an output on x and an input on y , respectively. Both actions have priority \mathbf{o} , which is strictly smaller than any priorities in their typing contexts, respecting the fact that they are top-level prefixes. The remaining β -reductions are summarised below. β_{AXCYCLE} simplifies a CYCLE involving an axiom. $\beta_{\mathbf{1}\perp}$ closes and eliminates channels. $\beta_{\oplus \&}$, similarly to $\beta_{\otimes \wp}$, simplifies a communication between a selection and a branching. $\beta_{!?}$ simplifies a cycle between one server of type $!^\circ A$ and one client of type $?^\circ A$. The last two rules differ in the number of clients involved: rule $\beta_{!W}$ considers no clients, whether $\beta_{!C}$ considers multiple clients.

$$\begin{aligned}
\beta_{\text{AXCYCLE}} & (\nu y^A z)(x \rightarrow y^A \mid P) \vdash \Gamma, x : A^\perp \longrightarrow P\{x/z\} \vdash \Gamma, x : A^\perp \\
\beta_{\mathbf{1}\perp} & (\nu x^A y)(x[\cdot].\mathbf{0} \mid y().P) \vdash \Gamma \longrightarrow P \vdash \Gamma \\
\beta_{\oplus \&} & (\nu x^{\oplus \circ \{l_i : B_i\}_{i \in I}} y)(x \triangleleft l_j.P \mid y \triangleright \{l_i : Q_i\}_{i \in I}) \vdash \Gamma, \Delta \longrightarrow \\
& (\nu x^{B_j} y)(P \mid Q_j) \vdash \Gamma, \Delta \\
\beta_{!?} & (\nu x^{!^\circ A} y)(!x(v).P \mid ?y[w].Q) \vdash ?\Gamma, \Delta \longrightarrow (\nu v^A w)(P \mid Q) \vdash ?\Gamma, \Delta \\
\beta_{!W} & (\nu x^{!^\circ A} y)(!x(v).P \mid Q) \vdash ?\Gamma, \Delta \longrightarrow Q \vdash ?\Gamma, \Delta \\
\beta_{!C} & (\nu x^{!^\circ A} y)(!x(v).P \mid Q\{y/y', y/y''\}) \vdash ?\Gamma, \Delta \longrightarrow \\
& (\nu x'^{!^\circ A} y')(!x'(v').P' \mid (\nu x''^{!^\circ A} y'')(!x''(v'').P'' \mid Q)) \vdash ?\Gamma, \Delta
\end{aligned}$$

$$\begin{array}{c}
\frac{\mathbf{o} < \text{pr}(\Gamma) \quad P \vdash \Gamma, v : A, x : B}{x[v].P \vdash \Gamma, x : A \otimes B} \otimes \quad \frac{\mathbf{o} < \text{pr}(\Delta) \quad Q \vdash \Delta, w : A^\perp, y : B^\perp}{y(w).Q \vdash \Delta, y : A^\perp \wp B^\perp} \wp \\
\frac{x[v].P \mid y(w).Q \vdash \Gamma, \Delta, x : A \otimes B, y : A^\perp \wp B^\perp}{(\nu x^{A \otimes B} y)(x[v].P \mid y(w).Q) \vdash \Gamma, \Delta} \text{CYCLE} \quad \text{MIX} \\
\longrightarrow \quad \frac{P \vdash \Gamma, v : A, x : B \quad Q \vdash \Delta, w : A^\perp, y : B^\perp}{P \mid Q \vdash \Gamma, \Delta, v : A, x : B, w : A^\perp, y : B^\perp} \text{MIX} \\
\frac{\quad}{(\nu v^A w)(\nu x^B y)(P \mid Q) \vdash \Gamma, \Delta} \text{CYCLE}^2
\end{array}$$

Fig. 3. β -reduction for \otimes and \wp .

Commuting conversions, following [12,41], allow communication prefixes to be moved to the conclusion of a typing derivation, corresponding to pulling them out of the scope of `CYCLE` rules. In order to account for the sequence of `CYCLES`, here we use $\tilde{\cdot}$. Due to this movement, if a prefix on a channel endpoint x with priority \circ is pulled out at top level, then to preserve priority conditions in the typing rules in Fig. 2, it is necessary to increase priorities of all actions after the prefix on x . This increase is achieved by using $\uparrow^{\circ+1}(\cdot)$ in the typing contexts.

$$\begin{array}{l}
\kappa_{\perp} \quad (\nu \tilde{x}^{\tilde{A}} \tilde{y})(x().P \mid Q) \vdash \Gamma, \Delta, x: \perp^{\circ} \longrightarrow \\
\qquad \qquad \qquad x().[(\nu \tilde{x}^{\tilde{A}} \tilde{y})(P \mid Q)] \vdash \uparrow^{\circ+1} \Gamma, \uparrow^{\circ+1} \Delta, x: \perp^{\circ} \\
\kappa_{\otimes} \quad (\nu \tilde{x}^{\tilde{A}} \tilde{y})(x[v].P \mid Q) \vdash \Gamma, \Delta, x: A \otimes^{\circ} B \longrightarrow \\
\qquad \qquad \qquad x[v].[(\nu \tilde{x}^{\tilde{A}} \tilde{y})(P \mid Q)] \vdash (\uparrow^{\circ+1} \Gamma), (\uparrow^{\circ+1} \Delta), x: (\uparrow^{\circ+1} A) \otimes^{\circ} (\uparrow^{\circ+1} B) \\
\kappa_{\wp} \quad (\nu \tilde{x}^{\tilde{A}} \tilde{y})(x(w).P \mid Q) \vdash \Gamma, \Delta, x: A \wp^{\circ} B \longrightarrow \\
\qquad \qquad \qquad x(w).[(\nu \tilde{x}^{\tilde{A}} \tilde{y})(P \mid Q)] \vdash (\uparrow^{\circ+1} \Gamma), (\uparrow^{\circ+1} \Delta), x: (\uparrow^{\circ+1} A) \wp^{\circ} (\uparrow^{\circ+1} B) \\
\kappa_{\oplus} \quad (\nu \tilde{x}^{\tilde{A}} \tilde{y})(x \triangleleft l_j.P \mid Q) \vdash \Gamma, \Delta, x: \oplus^{\circ} \{l_i : B_i\}_{i \in I} \longrightarrow \\
\qquad \qquad \qquad x \triangleleft l_j.[(\nu \tilde{x}^{\tilde{A}} \tilde{y})(P \mid Q)] \vdash (\uparrow^{\circ+1} \Gamma), (\uparrow^{\circ+1} \Delta), x: \oplus^{\circ} \{l_i : \uparrow^{\circ+1} B_i\}_{i \in I} \\
\kappa_{\&} \quad (\nu \tilde{x}^{\tilde{A}} \tilde{y})(x \triangleright \{l_i : P_i\}_{i \in I} \mid Q) \vdash \Gamma, \Delta, x: \&^{\circ} \{l_i : B_i\}_{i \in I} \longrightarrow \\
\qquad \qquad \qquad x \triangleright \{l_i : (\nu \tilde{x}^{\tilde{A}} \tilde{y})(P_i \mid Q)\}_{i \in I} \vdash (\uparrow^{\circ+1} \Gamma), (\uparrow^{\circ+1} \Delta), x: \&^{\circ} \{l_i : \uparrow^{\circ+1} B_i\}_{i \in I} \\
\kappa_{?} \quad (\nu \tilde{x}^{\tilde{A}} \tilde{y})(?x[w].P \mid Q) \vdash \Gamma, \Delta, x: ?^{\circ} A \longrightarrow \\
\qquad \qquad \qquad ?x[w].[(\nu \tilde{x}^{\tilde{A}} \tilde{y})(P \mid Q)] \vdash (\uparrow^{\circ+1} \Gamma), (\uparrow^{\circ+1} \Delta), x: ?^{\circ} (\uparrow^{\circ+1} A) \\
\kappa_{!} \quad (\nu \tilde{x}^{\tilde{?^{\circ} A} \tilde{y}})(!x(v).P \mid Q) \vdash ?\Gamma, \Delta, x: !^{\circ} A \longrightarrow \\
\qquad \qquad \qquad !x(v).[(\nu \tilde{x}^{\tilde{?^{\circ} A} \tilde{y}})(P \mid Q)] \vdash (\uparrow^{\circ+1} \Gamma), (\uparrow^{\circ+1} \Delta), x: !^{\circ} (\uparrow^{\circ+1} A)
\end{array}$$

Finally, we give the following additional reduction rules: closure under structural equivalence, and two congruence rules, for restriction and for parallel.

$$\begin{array}{ll}
\text{CLOSE-EQUIV} & P \equiv Q \quad Q \longrightarrow R \quad R \equiv S \text{ implies } P \longrightarrow S \\
\text{CONG-CYCLE} & P \longrightarrow Q \text{ implies } (\nu x^A y)P \longrightarrow (\nu x^A y)Q \\
\text{CONG-MIX} & P \longrightarrow Q \text{ implies } P \mid R \longrightarrow Q \mid R
\end{array}$$

4 Results for PLL and PCP

4.1 `CYCLE`-Elimination for PLL

We start with results for `CYCLE`-elimination for PLL; thus here we refer to A, B as propositions, rather than types. The detailed proofs can be found in [18].

Definition 6. *The degree function $\partial(\cdot)$ on propositions is defined by:*

$$\begin{array}{l}
- \partial(1^{\circ}) = \partial(\perp^{\circ}) = 1 \\
- \partial(A \otimes^{\circ} B) = \partial(A \wp^{\circ} B) = \partial(A) + \partial(B) + 1 \\
- \partial(\&^{\circ} \{l_i : A_i\}_{i \in I}) = \partial(\oplus^{\circ} \{l_i : A_i\}_{i \in I}) = \sum_{i \in I} \{\partial(A_i)\} + 1 \\
- \partial(?^{\circ} A) = \partial(!^{\circ} A) = \partial(A) + 1.
\end{array}$$

Definition 7. *A MAXICUT is a maximal sequence of MIX and CYCLE rules, ending with a CYCLE rule.*

Maximality means that the rules applied immediately before a MAXICUT are any rules in Fig. 2, other than MIX or CYCLE. The order in which MIX and CYCLE rules are applied within a MAXICUT is irrelevant. However, Proposition 1, which follows directly from structural equivalence (Sect. 3), allows us to simplify a MAXICUT.

Proposition 1 (Canonical MAXICUT). *Given an arbitrary MAXICUT, it is always possible to obtain from it a canonical MAXICUT consisting of a sequence of only MIX rules followed by a sequence of only CYCLE rules.*

Definition 8. *A single-Mix MAXICUT contains only one MIX rule.*

A_1, \dots, A_n, A are MAXICUT propositions if they are eliminated by a MAXICUT. The degree of a sequence of CYCLES is the sum of the degrees of the eliminated propositions.

The degree of a MAXICUT is the sum of the degrees of the CYCLES in it.

The degree of a proof π , $d(\pi)$, is the sup of the degrees of its MAXICUTS, implying $d(\pi) = 0$ if and only if proof π has no CYCLES.

The height of a proof π , $h(\pi)$, is the height of its tree, and it is defined as $h(\pi) = \sup(h(\pi_i))_{i \in I} + 1$, where $\{\pi_i\}_{i \in I}$ are the subproofs of π .

MAXICUT has some similarities with the derived MULTICUT: it generalises MULTICUT in the number of MIXES, and a single-Mix MAXICUT is an occurrence of MULTICUT.

The core of CYCLE-elimination for our PLL, as for CUT-elimination for CLL [10, 25], is the Principal Lemma (Lemma 3), which eliminates a CYCLE by either (i) replacing it with another CYCLE on simpler propositions, or (ii) pushing it further up the proof tree. Item (i) corresponds to (the logical part of) β -reductions (Sect. 3); and (ii) corresponds to (the logical part of) commuting conversions (Sect. 3).

Exceptionally, $\beta_{!C}$ reduces the original proof in a way that neither (i) nor (ii) are respected. In order to cope with this case, we introduce Lemma 2, which is inspired by Lemma B.1.3 in Bräuner [10], and adapted to our PLL. Lemma 2 allows us to reduce the degree of a proof ending with a single-Mix MAXICUT and having the same degree as the whole proof, and where the last rule applied on the left hand-side immediate subproof is $!$. Let $[n]$ denote the set $\{1, \dots, n\}$.

Lemma 2 (Inspired by B.1.3 in Bräuner [10]). *Let τ be a proof of the following form, ending with a single-Mix MAXICUT:*

$$\begin{array}{c}
 \begin{array}{c} \pi \\ \vdots \\ \circ < \text{pr}(\Gamma) \\ \forall i \in [n] : \circ < \circ_i \\ \vdash \Gamma, ?^{o_1} A_1, \dots, ?^{o_n} A_n, A \end{array} \\
 \hline
 \vdash \Gamma, ?^{o_1} A_1, \dots, ?^{o_n} A_n, !^\circ A \quad ! \\
 \hline
 \vdash \Gamma, \Delta, ?^{o_1} A_1, \dots, ?^{o_n} A_n, !^\circ A, !^{o_1} A_1^\perp, \dots, !^{o_n} A_n^\perp, ?^\circ A^\perp \\
 \hline
 \vdash \Gamma, \Delta
 \end{array}
 \quad
 \begin{array}{c}
 \begin{array}{c} \pi' \\ \vdots \\ \circ < \text{pr}(\Delta) \\ \forall i \in [n] : \circ < \circ_i \quad \forall j \in [k] : \circ \leq \kappa_j \\ \vdash \Delta, !^{o_1} A_1^\perp, \dots, !^{o_n} A_n^\perp, (?^{\kappa_j} A^\perp)_{j \in [k]} \end{array} \\
 \hline
 \vdash \Delta, !^{o_1} A_1^\perp, \dots, !^{o_n} A_n^\perp, ?^\circ A^\perp \\
 \hline
 \vdash \Gamma, \Delta
 \end{array}
 \quad
 \begin{array}{c}
 C^{k-1} \\
 \text{MIX} \\
 \text{CYCLE}
 \end{array}
 \end{array}$$

where $d(\pi) < d(\tau)$ and $d(\pi') < d(\tau)$. Then, there is a proof τ' of $\vdash ?\Gamma, \Delta$ such that $d(\tau') < d(\tau)$.

Proof. Induction on $h(\pi')$, with a case-analysis on the last rule applied in π' . \square

Lemma 3 (The Principal Lemma). *Let τ be a proof of $\vdash \Gamma$, ending with a canonical MAXICUT:*

$$\frac{\frac{\pi_1 \dots \pi_m}{\vdash \Gamma, A_1, \dots, A_n, A, A_1^\perp, \dots, A_n^\perp, A^\perp} \text{MIX}}{\vdash \Gamma} \text{CYCLE}$$

such that for all $i \in [m]$, $d(\pi_i) < d(\tau)$. Then there is a proof τ' of $\vdash \uparrow^t \Gamma$, for some $t \geq 0$, such that $d(\tau') < d(\tau)$.

Proof. The proof is by induction on $\sum_{i \in [m]} h(\pi_i)$. Let r_i be the last rule applied in π_i , for $i \in [m]$ and let C_{r_i} be the proposition introduced by r_i . Consider the proposition with the *smallest* priority. If the proposition is not unique, just pick one. Let this proposition be C_{r_k} . Then, π_k is the following proof: $\frac{\dots}{\vdash \Gamma', C_{r_k}} r_k$. We proceed by cases on π_k .

– r_k is \otimes on one of the MAXICUT propositions A_1, \dots, A_n, A . Without loss of generality, suppose r_k is applied on A , meaning $A = E \otimes F$ for some E and F and $\circ \geq 0$. By \otimes rule in Fig. 2, $\circ < \text{pr}(\Gamma')$. Since A is a MAXICUT proposition, by Definition 2, $A^\perp = E^\perp \wp^\circ F^\perp$. Since $\circ < \text{pr}(\Gamma')$ and $\text{pr}(A^\perp) = \circ$, it must be

that A^\perp is in another proof, say π_h : $\frac{\dots}{\vdash \Gamma'', E^\perp \wp^\circ F^\perp} r_h$

Consider the case where r_h is a multiplicative, additive, exponential or \perp rule in Fig. 2. Suppose r_h is applied on C_{r_h} which is not A^\perp . All the mentioned rules require $\text{pr}(C_{r_h}) < \text{pr}(\Gamma'', E^\perp \wp^\circ F^\perp \setminus C_{r_h})$, implying $\text{pr}(C_{r_h}) < \text{pr}(E^\perp \wp^\circ F^\perp) = \text{pr}(E \otimes F) = \circ$. This contradicts the fact that \circ is the smallest priority. Hence, r_h must be a \wp introducing A^\perp .

We construct proof τ_A ending with a single-MIX MAXICUT applied on *at least* A :

$$\frac{\frac{\frac{\pi_\otimes}{\vdash \Gamma', E, F} \circ < \text{pr}(\Gamma')}{\vdash \Gamma', E \otimes F} \otimes \frac{\frac{\pi_\wp}{\vdash \Gamma'', E^\perp, F^\perp} \circ < \text{pr}(\Gamma'')}{\vdash \Gamma'', E^\perp \wp^\circ F^\perp} \wp}{\vdash \Gamma', \Gamma'', E \otimes F, E^\perp \wp^\circ F^\perp} \text{MIX}}{\vdash \Gamma'''} \text{CYCLE}$$

Then, by structural equivalence, we can rewrite τ in terms of τ_A . By applying $\beta_{\otimes \wp}$ on τ_A (only considering the logical part), we obtain a proof τ'_A such that $d(\tau'_A) < d(\tau_A) \leq d(\tau)$, because $\partial(E) + \partial(F) < \partial(E \otimes F)$. We can then construct τ' by substituting τ'_A for τ_A in τ , which concludes this case.

Theorem 3 (Top-Level Deadlock-Freedom). *If $P \vdash \Gamma$ and P is a CYCLE, then there is some Q such that $P \longrightarrow^* Q$ and Q is not a CYCLE.*

Proof. The interpretation of Lemma 3 for PCP is that either (i) a top-level communication occurs, corresponding to a β -reduction, or (ii) commuting conversions are used to push CYCLE further inwards in a process. Consequently, iterating Lemma 3 results in eliminating top-level CYCLES. \square

Eliminating all CYCLES, as specified by Theorem 1, would correspond to a semantics in which reduction occurs under prefixes, as discussed by Wadler [41]. In order to achieve this, we would need to introduce additional congruence rules, such as:

$$\frac{P \longrightarrow Q}{x(y).P \longrightarrow x(y).Q}$$

and similarly for other actions. Reductions of this kind are not present in the π -calculus, and we also omit them in our framework.

However, we can eliminate all CYCLES in a proof of $\vdash \emptyset$, corresponding to full deadlock-freedom for closed processes. Kobayashi's type system [32] satisfies the same property.

Theorem 4 (Deadlock-Freedom for Closed Processes). *If $P \vdash \emptyset$, then either $P \equiv \mathbf{0}$ or there is Q such that $P \longrightarrow Q$.*

Proof. This follows from Theorems 2 and 3, because if $Q \vdash \emptyset$ and Q is not a CYCLE then Q must be a parallel composition of $\mathbf{0}$ processes. \square

5 Related Work and Conclusion

CYCLE and MULTICUT rules were explored by Abramsky *et al.* [2–4] in the context of $*$ -autonomous categories. That work is not directly comparable with ours, as it only presented a typed semantics for CCS-like processes and did not give a type system for a language or a term assignment for a logical system. Atkey *et al.* [5] added a MULTICUT rule to CP, producing an isomorphism between \otimes and \wp , but they did not consider deadlock-freedom.

In Kobayashi's original type-theoretic approach to deadlock-freedom [29], priorities were abstract tags from a partially ordered set. In later work abstract tags were simplified to natural numbers, and priorities were replaced by pairs of obligations and capabilities [30, 32]. The latter change allows more processes to be typed, at the expense of a more complex type system. Padovani [36] adapted Kobayashi's approach to session types, and later on he simplified it to a single priority for linear π -calculus [37]. Then, the single priority technique can be transferred to session types by the encoding of session types into linear types [16, 17, 19, 33]. For simplicity, we have opted for single priorities, as Padovani [37].

The first work on progress for session types, by Dezani-Ciancaglini *et al.* [15, 22], guaranteed the property by allowing only one active session at a time. Later work [21] introduced a partial order on channels in Kobayashi-style [29].

Bettini *et al.* [9] applied similar ideas to multiparty session types. The main difference with our work is that we associate priorities with individual communication operations, rather than with entire channels. Carbone *et al.* [13] proved that progress is a compositional form of lock-freedom and introduced a new technique for progress in session types by adopting Kobayashi’s type system and the encoding of session types [19]. Vieira and Vasconcelos [40] used single priorities and an abstract partial order in session types to guarantee deadlock-freedom.

The linear logic approach to deadlock-free session types started with Caires and Pfenning [12], based on dual intuitionistic linear logic, and was later formulated for classical linear logic by Wadler [41]. All subsequent work on linear logic and session types enforces deadlock-freedom by forbidding cyclic connections. In their original work, Caires and Pfenning commented that it would be interesting to compare process typability in their system with other approaches including Kobayashi’s and Dezani-Ciancaglini’s. However, we are aware of only one comparative study of the expressivity of type systems for deadlock-freedom, by Dardha and Pérez [20]. They compared Kobayashi-style typing and CLL typing, and proved that CLL corresponds to Kobayashi’s system with the restriction that only single cuts, not multicuts, are allowed.

In this paper, we have presented a new logic, priority-based linear logic (PLL), and a term assignment system, priority-based CP (PCP), that increase the expressivity of deadlock-free session type systems, by combining Caires and Pfenning’s linear logic-based approach and Kobayashi’s priority-based type system. The novel feature of PLL and PCP is `CYCLE`, which allows cyclic process structures to be formed if they do not violate ordering conditions on the priorities of prefixes. Following the propositions-as-types paradigm, we prove a `CYCLE`-elimination theorem analogous to the standard `CUT`-elimination theorem. As a result of this theorem, we obtain deadlock-freedom for a class of π -calculus processes which is larger than the class typed by Caires and Pfenning. In particular, these are processes that typically share more than one channel in parallel.

There are two main directions for future work. First, develop a type system for a functional language, priority-based GV, and translate it into PCP, along the lines of Lindley and Morris’ [34] translation of GV [41] into CP. Second, extend PCP to allow recursion and sharing [6], in order to support more general concurrent programming, while maintaining deadlock-freedom, as well as termination, or typed behavioural equivalence.

Acknowledgements. We are grateful for suggestions and feedback from the anonymous reviewers and colleagues: Wen Kokke, Sam Lindley, Roly Perera, Frank Pfenning, Carsten Schürmann and Philip Wadler.

References

1. Abramsky, S.: Proofs as processes. *Theor. Comput. Sci.* **135**(1), 5–9 (1994)
2. Abramsky, S., Gay, S.J., Nagarajan, R.: Interaction categories and the foundations of typed concurrent programming. In: Broy, M. (ed.) *Proceedings of the NATO Advanced Study Institute on Deductive Program Design*, pp. 35–113 (1996)

3. Abramsky, S., Gay, S., Nagarajan, R.: A type-theoretic approach to deadlock-freedom of asynchronous systems. In: Abadi, M., Ito, T. (eds.) TACS 1997. LNCS, vol. 1281, pp. 295–320. Springer, Heidelberg (1997). <https://doi.org/10.1007/BFb0014557>
4. Abramsky, S., Gay, S.J., Nagarajan, R.: A specification structure for deadlock-freedom of synchronous processes. *Theor. Comput. Sci.* **222**(1–2), 1–53 (1999)
5. Atkey, R., Lindley, S., Morris, J.G.: Conflation confers concurrency. In: Lindley, S., McBride, C., Trinder, P., Sannella, D. (eds.) A List of Successes That Can Change the World. LNCS, vol. 9600, pp. 32–55. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30936-1_2
6. Balzer, S., Pfenning, F.: Manifest sharing with session types. In: Proceedings of the ACM on Programming Languages, vol. 1(ICFP), pp. 37:1–37:29 (2017)
7. Barber, A.: Dual intuitionistic linear logic. Technical report ECS-LFCS-96-347, University of Edinburgh (1996). www.lfcs.inf.ed.ac.uk/reports/96/ECS-LFCS-96-347
8. Bellin, G., Scott, P.J.: On the pi-calculus and linear logic. *Theor. Comput. Sci.* **135**(1), 11–65 (1994)
9. Bettini, L., Coppo, M., D’Antoni, L., De Luca, M., Dezani-Ciancaglini, M., Yoshida, N.: Global progress in dynamically interleaved multiparty sessions. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 418–433. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85361-9_33
10. Bräuner, T.: Introduction to linear logic. Technical report BRICS LS-96-6, Basic Research Institute in Computer Science, University of Aarhus (1996)
11. Caires, L., Pérez, J.A.: Linearity, control effects, and behavioral types. In: Yang, H. (ed.) ESOP 2017. LNCS, vol. 10201, pp. 229–259. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54434-1_9
12. Caires, L., Pfenning, F.: Session types as intuitionistic linear propositions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 222–236. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15375-4_16
13. Carbone, M., Dardha, O., Montesi, F.: Progress as compositional lock-freedom. In: Kühn, E., Pugliese, R. (eds.) COORDINATION 2014. LNCS, vol. 8459, pp. 49–64. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43376-8_4
14. Carbone, M., Lindley, S., Montesi, F., Schürmann, C., Wadler, P.: Coherence generalises duality: a logical explanation of multiparty session types. In: CONCUR. LIPIcs, vol. 59, pp. 33:1–33:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2016)
15. Coppo, M., Dezani-Ciancaglini, M., Yoshida, N.: Asynchronous session types and progress for object oriented languages. In: Bonsangue, M.M., Johnsen, E.B. (eds.) FMOODS 2007. LNCS, vol. 4468, pp. 1–31. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72952-5_1
16. Dardha, O.: Recursive session types revisited. In: BEAT. EPTCS, vol. 162, pp. 27–34 (2014)
17. Dardha, O.: Type Systems for Distributed Programs: Components and Sessions. Atlantis Studies in Computing, vol. 7. Atlantis Press, Paris (2016). <https://doi.org/10.2991/978-94-6239-204-5>
18. Dardha, O., Gay, S.J.: A new linear logic for deadlock-free session typed processes. In: 21st International Conference on Foundations of Software Science and Computation Structures, FoSSaCS 2018 (Extended Version). <http://www.dcs.gla.ac.uk/~ornela/publications/DG18-Extended.pdf>
19. Dardha, O., Giachino, E., Sangiorgi, D.: Session types revisited. In: PPDP, pp. 139–150. ACM (2012)

20. Dardha, O., Pérez, J.A.: Comparing deadlock-free session typed processes. In: EXPRESS/SOS. EPTCS, vol. 190, pp. 1–15 (2015)
21. Dezani-Ciancaglini, M., de'Liguoro, U., Yoshida, N.: On progress for structured communications. In: Barthe, G., Fournet, C. (eds.) TGC 2007. LNCS, vol. 4912, pp. 257–275. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78663-4_18
22. Dezani-Ciancaglini, M., Mostrous, D., Yoshida, N., Drossopoulou, S.: Session types for object-oriented languages. In: Thomas, D. (ed.) ECOOP 2006. LNCS, vol. 4067, pp. 328–352. Springer, Heidelberg (2006). https://doi.org/10.1007/11785477_20
23. Gay, S.J., Vasconcelos, V.T.: Linear type theory for asynchronous session types. *J. Funct. Program.* **20**(1), 19–50 (2010)
24. Girard, J.: Linear logic. *Theor. Comput. Sci.* **50**, 1–102 (1987)
25. Girard, J.-Y., Taylor, P., Lafont, Y.: *Proofs and Types*. Cambridge University Press, New York (1989)
26. Honda, K.: Types for dyadic interaction. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 509–523. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57208-2_35
27. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0053567>
28. Kobayashi, N.: TyPiCal: type-based static analyzer for the pi-calculus. www.kb.is.s.u-tokyo.ac.jp/~koba/typical
29. Kobayashi, N.: A partially deadlock-free typed process calculus. *ACM Trans. Program. Lang. Syst.* **20**(2), 436–482 (1998)
30. Kobayashi, N.: A type system for lock-free processes. *Inf. Comput.* **177**(2), 122–159 (2002)
31. Kobayashi, N.: Type systems for concurrent programs. In: Aichernig, B.K., Maibaum, T. (eds.) *Formal Methods at the Crossroads. From Panacea to Foundational Support*. LNCS, vol. 2757, pp. 439–453. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40007-3_26
32. Kobayashi, N.: A new type system for deadlock-free processes. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 233–247. Springer, Heidelberg (2006). https://doi.org/10.1007/11817949_16
33. Kobayashi, N.: Type systems for concurrent programs. Extended version of [31], Tohoku University (2007)
34. Lindley, S., Morris, J.G.: A semantics for propositions as sessions. In: Vitek, J. (ed.) ESOP 2015. LNCS, vol. 9032, pp. 560–584. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46669-8_23
35. Milner, R.: *Communication and Concurrency*. Prentice Hall, Upper Saddle River (1989)
36. Padovani, L.: From lock freedom to progress using session types. In: PLACES. EPTCS, vol. 137, pp. 3–19 (2013)
37. Padovani, L.: Deadlock and lock freedom in the linear π -Calculus. In: CSL-LICS, pp. 72:1–72:10. ACM (2014)
38. Takeuchi, K., Honda, K., Kubo, M.: An interaction-based language and its typing system. In: Halatsis, C., Maritsas, D., Philokyprou, G., Theodoridis, S. (eds.) PARLE 1994. LNCS, vol. 817, pp. 398–413. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58184-7_118
39. Toninho, B., Caires, L., Pfenning, F.: Dependent session types via intuitionistic linear type theory. In: PPDP, pp. 161–172. ACM (2011)

40. Torres Vieira, H., Thudichum Vasconcelos, V.: Typing progress in communication-centred systems. In: De Nicola, R., Julien, C. (eds.) COORDINATION 2013. LNCS, vol. 7890, pp. 236–250. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38493-6_17
41. Wadler, P.: Propositions as sessions. In: ICFP, pp. 273–286. ACM (2012)
42. Wadler, P.: Propositions as types. *Commun. ACM* **58**(12), 75–84 (2015)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

