Typing Confluence*

Uwe Nestmann†‡
INRIA Rocquencourt, France

Martin Steffen§
Christian-Albrechts-Universität Kiel, Germany

March 14, 1997

Abstract

We investigate confluence properties for concurrent systems of message-passing processes, because such properties have proved to be useful for a variety of applications, ranging from reasoning about concurrent objects to mobile and high-speed telecommunication protocols. Roughly, *confluence* means that for every two computations starting from a common system state, it is possible to continue the computations, so to reach a common state again. In order to prove confluence for a given system, we are required to demonstrate that for all states reachable by computation from the starting state, the 'flowing together' of possible computations is possible.

In this paper, we aim at proving confluence properties for concurrent systems without having to generate all reachable states. Instead, we use a type system that supports a static analysis of possible sources of non-confluence. In message-passing systems, confluence is invalidated whenever two processes compete for communication with another process. We may statically check the occurrence of such situations by reducing them to the concurrent access on a shared communication port. For the technical development, we focus on the setting of a polarized π-calculus, where we formalize the notion of *port-uniqueness* by means of overlapping-free context-redex decompositions. We then present a type system for port-uniqueness that, taking advantage of a subject reduction property, yields a sufficient criterion for guaranteeing confluence.

*This work was started while both authors were staying at the Universität Erlangen-Nürnberg, Germany. The main results and proofs have appeared previously in the first authors PhD thesis [Nes96].

†Corresponding author: INRIA Rocquencourt, Projet PARA, Domaine de Voluceau, Rocquencourt, B.P. 105, F-78153 Le Chesnay Cedex (France). Email: `Uwe.Nestmann@INRIA.fr`

‡Supported by an ERCIM fellowship and a grant of the DAAD-program HSPII-AUFE.

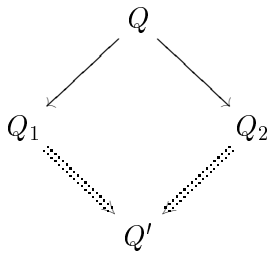
§Supported by the DFG, Sonderforschungsbereich 182, project C2.

1 Introduction with historical background

Notions of confluence have been explored in various forms with emphasis on the widely-investigated field of term rewriting systems for the modeling of sequential computation. With respect to the modeling of concurrent system behavior by means of labeled transition systems, much less work has been done as regards confluence. This section deals to a large extent with the exposition of the latter, with emphasis on computations of mobile processes, in order to clearly position the goal and the contribution of the current paper.

1.1 Sequential computation: Term rewriting systems

In the context of term rewriting systems like the λ-calculus, the investigation of confluence properties has a long tradition. There, confluence requires that possibly diverging paths of computation may always be brought together again. In the following, we use arrows $\rightarrow$ to denote single computation steps and $\Rightarrow$ to denote sequences of steps. Diagrams as below, read as follows: 'For all straight arrows, there are some dotted arrows (such that ...)'



A term Q is called (locally) confluent if any two derivatives Q_1 and Q_2 that are reachable by different rewriting steps, can be rewritten to the same term Q' again by possibly employing many more rewriting steps. Confluence is often explained by using completion diagrams of the depicted form where the upper half—the *span*—represents two diverging rewrite steps leading from Q to Q_1 and Q_2, and the lower half—the *completion*—represents the additional rewriting steps that enable us to finally reach the same term Q'. The main contribution of confluence is that it implies that the so-called normal forms—terms that exhibit no more reductions—are unique, if they exist. For systems that model the (sequential) computation of values, like in functional programming, this fact is useful, since it often allows an implementation to choose any of the possible execution paths.

1.2 Concurrent computation: Labeled transition systems

The semantics of concurrent systems is often modeled by means of labeled transition systems (LTS), where the transitions are carrying labels μ for denoting their visible effect, and the label τ indicates an invisible (internal) action. We will often use the standard relations:

$$\Rightarrow \stackrel{\text{def}}{=} \xrightarrow{\tau}{}^{*} \qquad \xrightarrow{\widehat{\mu}} \stackrel{\text{def}}{=} \begin{cases} \xrightarrow{\mu} & \text{if } \mu \neq \tau \\ \xrightarrow{\tau} \cup \text{id} & \text{if } \mu = \tau \end{cases} \qquad \stackrel{\widehat{\mu}}{\Longrightarrow} \stackrel{\text{def}}{=} \Rightarrow \xrightarrow{\widehat{\mu}} \Rightarrow$$

LTS are, in general, assigned via a structural operational semantics to the syntactic descriptions of process behavior as terms that are generated from a process algebra; we call *derivative* of term P a state that is reachable from P be a sequence of transitions. Terms model possibly infinite reactive systems, so there is not always a primary interest in reducing a process term to some state, where computation stops. Nevertheless, the knowledge about certain confluence properties of process systems may be of help in particular for verification purposes.

Due to the use of labeled transition systems, we have to deal with labeled confluence diagrams like the ones shown in Figures 1 and 2, which we sketch briefly below. Note that,

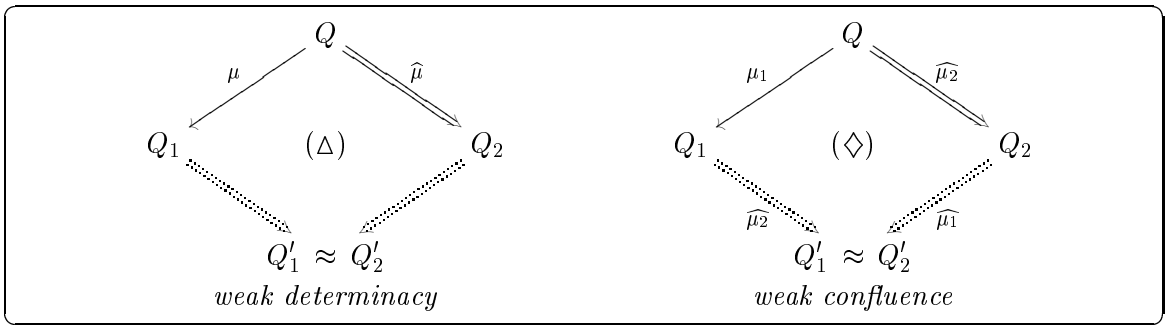


Figure 1: Confluence à la [Mil89]

since processes are rather equivalence classes (denoted in the figures by the equivalence relations $\asymp$ and $\approx$) of terms than terms themselves, the completion of confluence diagrams is often required only *up to* those equivalences.

Determinacy

In essence, determinacy means that any future system state is completely defined (up to some notion of equivalence, like weak bisimulation $\approx$) by the computation trace that leads to it. According to Tofts [Tof91], *determinacy in concurrent programming is like side-effect-freedom in functional programming.*

Milner tried to find a definition of determinacy that is preserved by his favorite equivalence relation, weak bisimulation, and that is furthermore preserved by CCS operators [Mil89]. By that, systems would satisfy 'good' properties for predicting their behavior by construction. The definition, from which Milner started, reads as: *a system P is* determinate *if all derivatives Q that are reachable after a given trace, are wekly bisimilar.* However, this definition specifies a property that is not preserved under parallel composition, if the parallel components would have shared access to channel names (as e.g. in $a^-|a^+$ denoting a sender on a and a receiver on a in parallel, which has two $\Rightarrow$-derivatives that are not bisimilar).

Milner strengthened the definition to *confluence*: *a system P is* weakly confluent *if each of its derivatives Q satisfies the $\triangle$- and $\diamondsuit$-diagram in Figure 1* (the shape of the spans is just convenient for proof purposes), such that the resulting property implies determinacy and is at least preserved under *confluent composition.* The latter is defined by requiring that channel names in parallel components are either disjoint (so no communication can occur on those names between the components), or the scope of shared channel names has to be immediately closed after putting the parallel composition together (so no intervening communication of a third party is possible).

Inertness

In [GS96], Groote and Sellink argue that the main use of confluence properties is the fact that it implies τ-inertness. A process P is *τ-inert* (or *τ-stable* in Tofts [Tof91]), if it satisfies:

$$\textit{If } P \xrightarrow{\tau} Q, \textit{ then } P \approx Q.$$

It has also been shown by Milner that his notions of confluence imply τ-inertness. Note the similarity to sequential computation, where reduction does not change the value of a term;

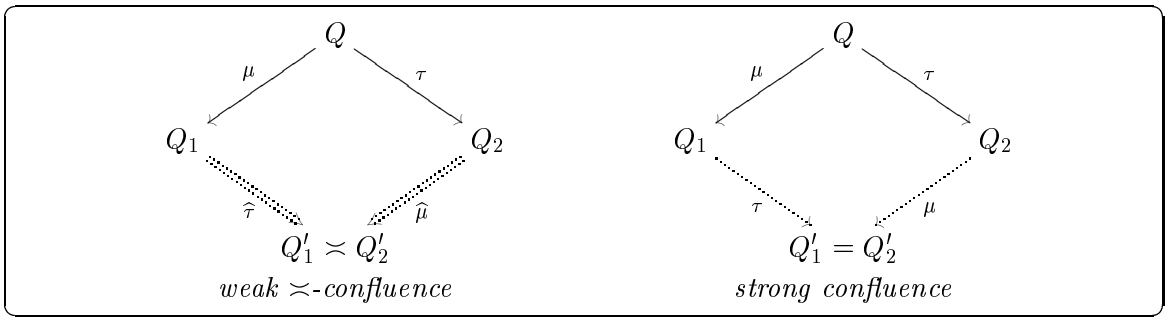


Figure 2: Confluence à la [GS96]

here, (internal) reduction does not change a term's behavior. This knowledge can be very useful for process verification since it allows to sometimes drastically reduce the state space by performing as many internal reductions as possible without changing the equivalence class of the process under investigation (see some examples in [GS96]).

With this motivation, the authors presented and investigated a couple of semantic notions of confluence, different from Milner's, in the context of arbitrary labeled transition systems with a distinguished label τ for internal actions; here, we only sketch two of them. No term language like CCS is assumed to denote the states of the transition system. According to [GS96], a transition system is *weak* $\asymp$*-confluent*, if each of its states Q satisfies the diagram on the left in Figure 2, where the span is always given as a pair of *different* (strong) transitions, at least one of which is internal. On the other hand, *strong confluence* was defined by means of the diagram on the right in Figure 2, where the confluent closure of the diagram is here required after just one step and up to identity.

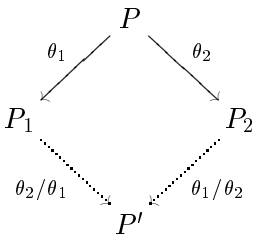
The main result of [GS96] is a characterization for τ-inert transition systems in terms of confluence, e.g. weak $\asymp$-confluence coincides with τ-inertness under the condition of well-foundedness with respect to τ-steps. We do not recapitulate that particular result, but we simply state that the above-mentioned notion of strong confluence at least implies τ-inertness, even in non-τ-well-founded transition systems.

Permutation

Sometimes, it is advantageous to know, whether two actions of some process that are enabled at the same time interfere, or not. If they do not interfere, then the two actions can be performed in either order. That information may be used to define an equivalence notion on computations that are represented by traces. Thus, equivalence classes of computations that are constructed as the quotient with respect to all possible permutations of actions can be regarded as *abstract computations* in the terminology of non-interleaving semantics [Mon93].

Two different transitions $P \xrightarrow{\mu_1} P_1$ and $P \xrightarrow{\mu_2} P_2$ of some process are called *concurrently enabled*. In simple value-passing process systems, two concurrently enabled transitions are usually *concurrent* (or: independent), if they do not compete for a shared communication partner. Imagine $a^-.P_1 \mid b^-.P_2$ in constrast to $a^-.P_1 \mid a^-.P_2$, when put in parallel with $a^+.P_3$, where the dot denotes sequential composition of an action with a process: in the first pair, the sender on a will not be disturbed, while the second pair will behave nondeterministically due to their competition for the receiver on a without possibly ever reaching a common state again. For languages like CCS [Mil89], in the absence of choice operators, this is the only source of non-confluence [BC88, Pri96].

The approach of *proved transitions* $P \xrightarrow{\theta} P'$, as introduced in [BC88], represents a more explicit formulation of this idea. There, the whole inference proof for an action is coded within its label θ representing a *proof term* that provides maximal information about the transition. For any two concurrently enabled θ_i-labeled proved transitions, their concurrency can be checked by analyzing the structure of the proof terms θ_i, which we do not report here. Any pair of concurrent transitions gives rise to the rather strong confluence diagram for some term P', which is referred to as *diamond*. The notation θ_1/θ_2 indicates that the proof term for deriving θ_1 after θ_2 has occurred may be different from deriving θ_1 as the first action. (In CCS, P' is uniquely determined, and if P does not contain a choice operator, then $\theta_i/\theta_j = \theta_i$.) This definition of confluence takes a somewhat local point of view (cf local confluence systems) in that it addresses particular instances of concurrent transition pairs.



1.3 Mobile computation: Name-passing processes

In contrast to CCS, which can be considered as a merely signal-passing process calculus, value-passing process calculi allow for the exchange of data in interactions. If data items are channel names, then the calculi are usually called name-passing process calculi. The prototypical such calculus is the π-calculus [MPW92]. Note that, via name-passing, processes can dynamically change their interaction structure. We use the terms *name* and *channel* as synonyms because of their use as both data and carrier.

We use *restriction* $(\boldsymbol{\nu} x)\, P$ to restrict the scope of a name x to process P, *composition* $P_1|P_2$ to denote the parallel composition of P_1 and P_2, *output* $y^-\langle z\rangle.P$ to denote the emission of name z along channel y before behaving as P, *input* $y^+(x).P$ to denote the reception of a name along channel y and binding it to x in process P, and *guarded replication* $!\,\pi.P$ to denote many copies of process $\pi.P$ in parallel, where $\pi.P$ denotes either an output or an input. In $y^-\langle z\rangle$ and $y^+(x)$, y is called *subject*, while x and z are called *object*.

The theory of name-passing calculi is, in general, more complicated compared to other value-passing caluli, since the scope of channel names may be extended by passing bound names as data to other processes that were previously outside the name's scope. Milner's notions of confluence have been generalized and adapted to name-passing calculi by Liu and Walker [LW95]. Below, we discuss two issues strongly related to the confluence of name-passing systems that were the basis for our own investigation.

Uniqueness

In order to characterize 'friendly' name-passing system with 'good' properties, Milner formulated syntactic *naming invariants in process communities* that were based on the idea of unique access to channels [Mil93]. In particular, he defined the notions of (unique) *bearing* and *handling* of names. We rephrase them as follows:

- P *bears* x, if x occurs free in P as positive subject
- P *can handle* x, if x occurs free in P as negative subject

(of an unguarded prefix). Consider a system of components P_i and R_j, constructed by

$$S \equiv (\boldsymbol{\nu}\tilde{x})\,(\, P_1 \,|\, \cdots \,|\, P_m \,|\, !\,R_1 \,|\, \cdots \,|\, !\,R_n \,).$$

Then, S is friendly, if $\mathit{fn}(S) = \emptyset$ and no P_i or R_j contains a composition or replication.

A name x is *uniquely borne* in a friendly S, if:

1. At most one component bears x.
2. If $R_j = \pi_j.Q_j$, then Q_j does not bear x.
3. If S contains the input prefix $y^+(x).Q$, then Q does not bear x.

A name x is *uniquely handled* in a friendly S, if:

4. At most one component can handle x.
5. No R_j can handle x.
6. If S contains the output prefix $y^-\langle z\rangle.Q$, then Q does not contain z free.

Although the above is an interesting starting point for thinking about invariants in process communities—note that both uniqueness properties, bearing and handling, are preserved under reduction—the characterization are applicable only for process systems of a restricted shape. For example, condition 6 forbids a process after passing a port to use its complement later on. Furthermore, condition 3 excludes that the bearing of a name can be acquired. Nevertheless, the characterization was sufficient for improving the reasoning about sharing, determinacy, and confluence in object-oriented concurrent systems [PW95], and also about resource management in a telecommunication protocol for high-speed networks [Ora94].

Types

Instead of syntactically constrained operators for the construction of process systems, it is also possible to use standard operators and afterwards statically check, whether the constructed system is of a certain type. Type systems for process languages have been recently introduced and investigated to some extent, with emphasis on name-passing process calculi. The standard aim has usually been to prevent run-time errors that can result from using channels in an unintended or incorrect way. This line of work has been carried out for channels with

- *recursive sorts* [Mil93] that control the *type/number* of transmittable values,
- *polarities* [PS96] that control *how* a channel may be used (cf Section 7),
- *modes* [KPT96] that control *how often* a channel may be used.

Especially the last category of typing systems is of interest for studying confluence issues since it has been shown that *linear* channels [KPT96], i.e. channels that can be used at most once throughout their lifetime, give rise to 'partial' confluence properties: a possible communication on a linear channel can never interfere with any other currently possible communication, so the two can be performed in either order. However, these results have only been investigated for internal reductions. No hint is given as to accomplish labeled transitions, and the relation to Milner's notion of 'unique handles' and 'unique nearing' or notions of (labeled) confluence has only been explained quite informally.

Finally, it is worth mentioning the join-calculus [FG96] as the syntactic predecessor of [Ama97, Kob97, San96b] that incorporate the idea of unique (and sometimes persistent) receivers, which substantiallu facilitates distributed implementation, as a central language design principle. Also, with unique persistent receivers, simple partial confluence properties for reductions hold, since an always available (*uniformly receptive* [San96b]) receiver is ready to communicate with any pending message in either order.

2 Goal and overview of this paper

Instead of verifying confluence properties of processes by reasoning about all their (possibly infinite) derivatives, our primary goal is a typing approach, which allows the static checking of confluence properties without generating all derivatives. Our approach may be seen as a complement to previous typing approaches, where confluence properties have appeared as a side result of well-typedness.

We introduce polarized name-passing (§3) and a formalization of unique access in terms of process decomposition (§4). We provide a typing system, which satisfies a labeled subject reduction property, for guaranteeing unique access to channel names in polarized name-passing processes (§5). Labeled subject reduction is the key in the proof that *labeled diamond confluence* holds for arbitrary derivatives of well-typed processes (§6). We compare our approach to others based on the notion of (sub)typing on names and on typed observations (§7), before we end with some concluding remarks (§8).

3 Technical Preliminaries

In CCS, as in most process calculi with channel-based communication, each use of a name in subject position of an action is marked syntactically with some polarity in order to indicate in which direction the channel is used. In name-passing calculi like the π-calculus, names may also occur in object position; there, they names do not come *a priori* with a particular syntactic polarity. Usually, this is intended and does not cause any harm. For example, in

$$y^-\langle w\rangle \mid y^+(x).x^-\langle z\rangle$$

it is not visible for the sender on y which port of w is going to be needed on the other side, maybe even both. The receiver on y just inputs (some capabilities of) a name that it formally calls x, and in the example at least uses the output port of x to send the capabilities for z to some other process.

However, if we want to control the unique access to ports, we are required to explicitly keep track of their mobility within process systems caused by name-passing. In that case, we had better use a *polarized* setting, where the polarity of object names is sufficiently explicit.

3.1 Polarized name-passing

This section introduces a monadic polarized π-calculus, similar to the one in [Ode95], but with synchronous output and without matching. Polarized name-passing is made explicit by syntactically decorating all *objects* of actions, including the non-binding occurrences in outputs, with syntactic polarity tags $+$ for input and $-$ for output.[1] Within this approach, each time a name x is generated, its two unidirectional halves x^+ and x^- are generated. Note that there is no neutral polarity $\pm$ and no notion of channel type as known from the π-calculus with subtyping [PS96]; the relation to the latter is explained in more detail in Section 7. So, in this purely syntactically polarized π-calculus, it is no longer possible to transmit both halves of a name within one single communication. It was argued in [Ode95] that this is not a serious defect of the calculus since it can always be achieved by two subsequent communications;

[1] We use a different syntax, compared to [Ode95], which also allows to avoid the distinction of the three syntactic categories of names, variables and constants, in favor of names and ports.

moreover, in a polyadic calculus, one could pass both polarities at once and read them from different positions in an input variable tuple.

Definition 1 (Names, ports, and processes). *In addition to the standard calculus with two kinds of basic entities, i.e. 'names' and 'processes', we explicitly introduce 'ports' as a third kind of entity that is derived from names.*

Let $x, \ldots, z$ *range over some infinite set of* names $\mathbf{N}$. *Let the set of* ports $\mathbf{N}^\star$ *be defined as 'polarized names': names that carry one of the* polarities $-$ *and* $+$. *Polarities are complementary, as expressed by the involution operation on ports with* $\overline{x^-} := x^+$ *and* $\overline{x^+} := x^-$. *When using ports in process expressions, we let* $\star$ *range as a metavariable over* $\{+,-\}$.

Let $\mathbb{P}$ *denote the set of processes generated by the grammar:*

$$P \;::=\; \mathbf{0} \;\;\big|\;\; (\boldsymbol{\nu} x)\,P \;\;\big|\;\; y^+(x^\star).P \;\;\big|\;\; y^-\langle z^\star\rangle.P \;\;\big|\;\; P \mid P \;\;\big|\;\; !\,y^+(x^\star).P$$

Let $\mathbb{F}$ *denote the subset of finite, i.e. replication-free polarized processes. Let* $\mathsf{fn}(P)$ *and* $\mathsf{bn}(P)$ *be defined in the ususal way indicating of* P *its the free and bound occurrences of names. It is straightforward to generalize the usual definition of free names in processes of* $\mathbb{P}$ *to free ports of processes in* $\mathbb{P}$ *by taking into account that output object occurrences indicate that the sender is (or: was), in principle, capable to access this port:*

$$\begin{aligned}
\mathsf{fp}(\mathbf{0}) &\stackrel{\text{def}}{=} \emptyset \\
\mathsf{fp}(y^+(x^\star).P) &\stackrel{\text{def}}{=} \{y^+\} \cup (\mathsf{fp}(P)\backslash\{x^+, x^-\}) \\
\mathsf{fp}(y^-\langle z^\star\rangle.P) &\stackrel{\text{def}}{=} \{y^-, z^\star\} \cup \mathsf{fp}(P) \\
\mathsf{fp}(P_1|P_2) &\stackrel{\text{def}}{=} \mathsf{fp}(P_1) \cup \mathsf{fp}(P_2) \\
\mathsf{fp}((\boldsymbol{\nu} x)\,P) &\stackrel{\text{def}}{=} \mathsf{fp}(P) \setminus \{x^+, x^-\} \\
\mathsf{fp}(!\,P) &\stackrel{\text{def}}{=} \mathsf{fp}(P)
\end{aligned}$$

defines the set of free ports *of processes in* $\mathbb{P}$.

Note that in the input case of the definition of $\mathsf{fp}(y^+(x^\star).P)$, we have to remove both x-ports because the binding addresses the *name* x, thus both polarities.

Labeled transition semantics

The operational semantics for processes with polarized name-passing is given as the smallest labeled transition relation generated by the rules in Table 1, where *labels* μ are of the form

$$\tau \;\;\big|\;\; y^+(x^\star) \;\;\big|\;\; y^-\langle z^\star\rangle \;\;\big|\;\; y^-(\boldsymbol{\nu} z^\star)$$

for denoting internal transitions, inputs, outputs, and bound outputs. Substitution $P\{^z/_x\}$ denotes P with all free occurrences of x replaced by z, silently assuming that bound names have been α-converted, if necessary, in order to avoid name-clashes. This definition is according with standard descriptions for late instantiation [San96a], except for the case of communication, as in rules *COM* and *CLOSE*, where the object names z and x are required to be tagged with the same polarity $\star$. Rule *ALPHA* makes explicit, where α-conversions are necessary in the derivation of transitions.

Since we are using a synchronous π-calculus (without matching), we use a standard definition of bisimulation.[2] Here, we introduce the weak variant of *ground* bisimulation [San96a].

[2] The main contribution of [Ode95] was the definition and application of a non-standard equivalence, called *polarized bisimulation*. Due to a smaller class of observers, based on a restricted notion of matching, it is a coarser notion than *asynchronous bisimulation* [HY95].

Definition 2. *A binary relation $\mathcal{R}$ on processes $\mathbb{P}$ is called* weak bisimulation, *if*

- $(P, Q) \in \mathcal{R}$ *and* $P \xrightarrow{\mu} P'$ *with* $\mathsf{bn}(\mu) \notin \mathsf{fn}(P|Q)$ *implies that there is* $Q \stackrel{\mu}{\Longrightarrow} Q'$ *such that* $(P', Q') \in \mathcal{R}$,

and vice versa. Two processes P, Q are called weakly bisimilar, if there is a weak bisimulation $\mathcal{R}$ with $(P, Q) \in \mathcal{R}$. Let $\approx$ denote the largest weak bisimulation.

It is well-known that ground bisimulations are, in general, not preserved under name-substitutions. However, we use bisimulation for only one example in this paper, where a particular bisimulation relation is preserved. Hence, we omit further technical material on bisimulations in name-passing process calculi; it can be found elsewhere (cf [Pri96, San96a]).

ACT: $\quad \pi.P \xrightarrow{\pi} P$

REP: $\quad !\pi.P \xrightarrow{\pi} P \mid !\pi.P$

OPEN: $\quad \dfrac{P \xrightarrow{y^-\langle z^\star\rangle} P'}{(\boldsymbol{\nu} z)\, P \xrightarrow{y^-(\boldsymbol{\nu} z^\star)} P'}$ *if* $y \neq z$

COM$_1^*$: $\quad \dfrac{P_1 \xrightarrow{y^-\langle z^\star\rangle} P_1' \qquad P_2 \xrightarrow{y^+(x^\star)} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'\{^z/_x\}}$

CLOSE$_1^*$: $\quad \dfrac{P_1 \xrightarrow{y^-(\boldsymbol{\nu} z^\star)} P_1' \qquad P_2 \xrightarrow{y^+(x^\star)} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} (\boldsymbol{\nu} z)\,(\, P_1' \mid P_2'\{^z/_x\}\,)}$ *if* $z \notin \mathsf{fn}(P_2)$

. . . . . .

PAR$_1^*$: $\quad \dfrac{P_1 \xrightarrow{\mu} P_1'}{P_1 \mid P_2 \xrightarrow{\mu} P_1' \mid P_2}$ *if* $\mathsf{bn}(\mu) \cap \mathsf{fn}(P_2) = \emptyset$

RES: $\quad \dfrac{P \xrightarrow{\mu} P'}{(\boldsymbol{\nu} x)\, P \xrightarrow{\mu} (\boldsymbol{\nu} x)\, P'}$ *if* $x \notin \mathsf{n}(\mu)$

ALPHA: $\quad \dfrac{P \xrightarrow{\mu} P'}{Q \xrightarrow{\mu} P'}$ *if* $P =_\alpha Q$

**: and the evident symmetric rules for parallel composition*

Table 1: Operational semantics for $\mathbb{P}$

3.2 Process structure

Certain laws on process terms, like the ones denoted by the symbol $\equiv$ in Table 2, have been recognized as having merely structural content [MS92, HY95]; they are valid with respect to all different kinds of behavioral congruences, equivalences and preorders, including bisimulation. The upper set of laws in Table 2 allow us to rearrange the components of a term such that a syntactically *distributed* τ-redex, i.e. a τ-redex with sender and receiver not being syntactic neighbors within some parallel composition, may always be brought together—by structural transformation—to form a *local* τ-redex. The lower set of laws in Table 2, i.e. the elimination of scope restrictions, and garbage collection of inaccessible guards are also often considered as structural laws, but are not needed for the purposes of this paper.

Eliminating name-clashes

In order to syntactically avoid name-clashes, we may require names that are introduced by restriction subterms, also the bound names of input prefix subterms are pairwise distinct among each other and with all other names occurring in the process.

Definition 3 (α-freeness). *A process P is called* α-free, *if all of its subterms of the form $(\boldsymbol{\nu} x)\,Q$ and $y^+(x^\star).Q$ use pairwise distinct names x and, furthermore, $\mathsf{fn}(P) \cap \mathsf{bn}(P) = \emptyset$.*

The operational semantics includes the rule *ALPHA* dealing with the necessary α-conversion of names that occur bound in transition labels and might cause capture when using the derivative in some parallel context. The following fact records some cases, where α-conversion may be avoided. We write $P \xrightarrow{\mu}_{\not\alpha} P'$ if the derivation does not involve rule *ALPHA*.

Fact 4. *Let $P \in \mathbb{P}$.*

- *There is an α-free process $\widehat{P}$ with $P =_\alpha \widehat{P}$.*
- *If P is α-free and $P \xrightarrow{\mu}_{\not\alpha} P'$, then P' is α-free.*
- *If $P \xrightarrow{\mu} P'$, then there is $\widehat{P}$ with $P =_\alpha \widehat{P} \xrightarrow{\mu}_{\not\alpha} P'$.*

Accordingly, we may disregard the use of *ALPHA*-rules in the derivation of transitions since there is always some α-congruent process term that allows to derive the transition without an application of *ALPHA*. Instead of always silently assuming that suitable α-conversions have

α-conversion	P	$\equiv$	Q	*if* $P =_\alpha Q$
associativity	$P \mid (Q \mid R)$	$\equiv$	$(P \mid Q) \mid R$	
commutativity	$P \mid Q$	$\equiv$	$Q \mid P$	
restriction	$(\boldsymbol{\nu} y)\,(\boldsymbol{\nu} x)\,P$	$\equiv$	$(\boldsymbol{\nu} x)\,(\boldsymbol{\nu} y)\,P$	
scope extrusion	$(\boldsymbol{\nu} y)\,P \mid Q$	$\equiv$	$(\boldsymbol{\nu} y)\,(P \mid Q)$	*if* $y \notin \mathsf{fn}(Q)$
scope elimination	$(\boldsymbol{\nu} x)\,Q$	$\equiv$	Q	*if* $x \notin \mathsf{fn}(Q)$
garbage collection	$(\boldsymbol{\nu} y)\,y^-\langle z^\star\rangle.P$	$\equiv$	$\mathbf{0}$	
	$(\boldsymbol{\nu} y)\,y^+(x^\star).P$	$\equiv$	$\mathbf{0}$	

Table 2: Structural laws

been performed, as often done in the literature, we indicate explicitly whether we make use of it. However, we assume that transitions of α-free processes are derived without *ALPHA*.

Process contexts

We introduce the notion of context as a means of structurally decomposing process terms into subterms and surrounding terms, where the positions of the former in the latter are often called *holes*. We do not admit every operator of the process language to be applied to the inner subterm, but only restriction and parallel composition. *Unary* process contexts are processes with *exactly one* hole, written $[\cdot]$, where a process may be plugged in.

Definition 5 (Process contexts). *Let $P \in \mathbb{P}$.* Unary *process contexts are generated by:*

$$C[\cdot] \;::=\; [\cdot] \;\;\big|\;\; P \,|\, C[\cdot] \;\;\big|\;\; C[\cdot] \,|\, P \;\;\big|\;\; (\boldsymbol{\nu} a)\, C[\cdot]$$

Let $C[\cdot,\cdot]$, $C[\cdot,\cdot,\cdot]$, and $C[\cdot,\cdot,\cdot,\cdot]$ be typical contexts with exactly 2, 3, or 4, holes.

Note that the paths from the root of a context to some hole only traverse restriction and composition. Multiple-hole contexts have also been used in [Pri96] to denote the positions of redexes inside process terms. Here, we implicitly assume that the holes are distinguished according to their position as parameter. We could also use metavariables that each occur ecactly once in the defined context expression. A more formal definition is omitted in favor of simplicity in the presentation.

Whereas bound names in processes are not essential and can be arbitrarily renamed as long as capture is avoided, the situation is different with contexts: *hole-binding* restrictions shall not be renamed since otherwise the insertion of a process that contains free names matching the restrictions for the hole might result in completely different processes.

Definition 6 (Free and hole-binding names). *Let $C[\cdot] \in \mathbb{C}^{(1)}$ be a unary context. Then the functions $\mathsf{fn}(C[\cdot])$ and $\mathsf{hn}(C[\cdot])$, as defined by*

$$\begin{array}{rcl@{\qquad}rcl}
\mathsf{fn}([\cdot]) & \stackrel{\text{def}}{=} & \emptyset & \mathsf{hn}([\cdot]) & \stackrel{\text{def}}{=} & \emptyset \\
\mathsf{fn}(C[\cdot]|P) & \stackrel{\text{def}}{=} & \mathsf{fn}(C[\cdot]) \cup \mathsf{fn}(P) & \mathsf{hn}(C[\cdot]|P) & \stackrel{\text{def}}{=} & \mathsf{hn}(C[\cdot]) \\
\mathsf{fn}(P|C[\cdot]) & \stackrel{\text{def}}{=} & \mathsf{fn}(P) \cup \mathsf{fn}(C[\cdot]) & \mathsf{hn}(P|C[\cdot]) & \stackrel{\text{def}}{=} & \mathsf{hn}(C[\cdot]) \\
\mathsf{fn}((\boldsymbol{\nu} a)\, C[\cdot]) & \stackrel{\text{def}}{=} & \mathsf{fn}(C[\cdot]) \setminus \{a\} & \mathsf{hn}((\boldsymbol{\nu} a)\, C[\cdot]) & \stackrel{\text{def}}{=} & \mathsf{hn}(C[\cdot]) \cup \{a\}
\end{array}$$

denote of a context $C[\cdot]$ its free and hole-binding names, respectively.

As abbreviation, we also write $\mathsf{fn}(C)$ and $\mathsf{hn}(C)$ for $\mathsf{fn}(C[\cdot])$ and $\mathsf{hn}(C[\cdot])$. As a generalization to multi-hole contexts, let $\mathsf{hn}_i(C)$ denote the hole-binding names of C concerning its i-th hole, silently assuming that we apply the function only for contexts with enough holes.

Decomposition

We may precisely reconstruct aspects of the structure of a process from its transitions: each inference tree induces a decomposition of the process into the redex that generates the transition, and its context. By writing "there is $P = C[\widetilde{Q}]$" we always implicitly mean that there is some context $C[\cdot]$ with appropriate arity and processes $\widetilde{Q}$ such that $P = C[\widetilde{Q}]$. We call a context-redex construction a *decomposition*.

Since, in our language $\mathbb{P}$, only guards and replicated guards may give rise to transitions, it suffices to consider contexts that are constructed from restriction and parallel composition. For simplicity, we only present the decomposition for transitions of finite processes, i.e. $P \in \mathbb{F}$. However, restriction in the context of bound name-passing is no longer a static operator. Let us therefore assume that P is α-free in order to deal with derivatives in the case of labels that carry bound names. Let $C[\cdot]$ be the context of a decomposition of some α-free process with $z \in \mathit{hn}(C)$. Then, $C^{\not z}[\cdot]$ denotes the context after erasing the restriction operator (note that, due to α-freeness, there is exactly one) for name z; we omit the straightforward inductive definition. Let $\mathbb{F}_{\not\alpha}$ denote the subset of α-free finite processes.

Fact 7 (Decomposition of redexes). *Let $P \in \mathbb{F}_{\not\alpha}$.*

1. *If $P \xrightarrow{y^+(x)} P'$, then there is $P = C[y^+(x).Q]$ with $P' = C[Q]$.*
2. *If $P \xrightarrow{y^-\langle z\rangle} P'$, then there is $P = C[y^-\langle z\rangle.Q]$ with $P' = C[Q]$.*
3. *If $P \xrightarrow{y^-(\boldsymbol{\nu} z)} P'$, then there is $P = C[y^-\langle z\rangle.Q]$ with $P' = C^{\not z}[Q]$.*
4. *If $P \xrightarrow{\tau} P'$, then there is $P = C[y^-\langle z\rangle.Q_1, y^+(x).Q_2]$ with $P' = C[Q_1, Q_2\{^z/_x\}]$, if $z \notin \mathit{hn}_1(C)$, and $P' \equiv (\boldsymbol{\nu} z)\, C^{\not z}[Q_1, Q_2\{^z/_x\}]$, otherwise.*

Note that $y \in \mathit{hn}(C)$ in the case of bound output. In the case of a communication (case *4.*) that is derived by *CLOSE* ($z \in \mathit{hn}_1(C)$), the construction with the restriction in outermost position guarantees that both continuations of the local sender- and receiver-redexes lie within its scope. For example, if a communication along the channel y can be derived by

$$Q := y^-\langle z\rangle.Q_0 \qquad \text{and} \qquad C[\cdot] := Q_1 \mid (\boldsymbol{\nu} z)\,(\,[\cdot] \mid Q_2\,)),$$

i.e. with $z \in \mathit{hn}(C)$, then it does not matter whether the matching receiver is sitting inside Q_1 (with the τ-step derived by *CLOSE*) or Q_2 (with the τ-step derived by *COM*), since by moving the restriction on z outermost we get the derivation

$$P := C[Q] \;\equiv\; (\boldsymbol{\nu} z)\,(\,C^{\not z}[Q]\,) \;\xrightarrow{\tau}\; (\boldsymbol{\nu} z)\,(\,Q_1' \mid (Q_0 \mid Q_2')\,)$$

which captures both cases adequately, where the continuation of the receiver is either Q_1' (where a scope-extrusion for z is necessary) or Q_2' (where the scope of z may remain as it is).

If we wanted to generalize the decomposition to infinite processes $P \in \mathbb{P}_{\not\alpha}$, we would have to include additional clauses like the following as a replacement of the above case *1.*

1. If $P \xrightarrow{y^+(x)} P'$, then either
   (a) there is $P = C[y^+(x).Q]$ and $P' = C[Q]$, or
   (b) there is $P = C[!y^+(x).Q]$ and $P' = C[Q \mid !y^+(x).Q]$.

For the communication case, we would get three subcases, determined by either none, one, or both of the redexes being replicated. So, the generalization to infinite processes is not substantially different from the finite case, but would unnecessarily complicate the notation.

Within the approach of proved transitions (cf Introduction), it is quite simple to precisely denote the subterm of P from which the transition is generated by reading the proof term θ as the path from the root node of the process tree (according to its algebraic structure) to the (local) redex(es). For our purposes, it will suffice to know about the existence of decompositions, so we may work with the less verbose standard labeled transitions μ.

4 No sharing: From port-uniqueness to confluence

In this section, we provide a technique to check diamond confluence for spans of concurrently enabled transitions, based on the induced decomposition into process contexts and redexes: If no redexes are shared, then the transitions are concurrent and, hence, can be performed in either order. The formalization of the notion of *concurrent transitions* in terms of process decomposition motivates why it is useful to characterize the absence of sharing by means of unique access to ports.

Confluence is a semantic property since it is stating the possibility of actions *for all possible derivatives* of a process, according to the transitions as generated from its operational semantics. With [Mil89], the essence of confluence relies on the following intuition: *At any time in the evolution of some process, no occurrence of an action precludes a concurrently enabled action.* So, mainly, confluence addresses the question whether two 'concurrently enabled' transitions are 'concurrent' or whether they interfere.

Shared ports Milner's confluence-preserving operators represent a constructive attempt to ensure that concurrently enabled transitions can never interfere. Confluent composition $P \mid_{\tilde{x}} Q := (\boldsymbol{\nu}\tilde{x})(\, P|Q \,)$ prevents pre-emption by governing its components' ports at the interface to their respective environment: it forbids

1. the shared use of the same port (expressed in $\mathit{fp}(P) \cap \mathit{fp}(Q) = \emptyset$), and
2. the unrestricted use of complementary ports (expressed in $\overline{\mathit{fp}(P)} \cap \mathit{fp}(Q) \subseteq \{\tilde{x}\}$).

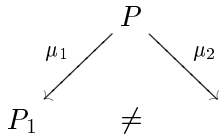
The first condition can be interpreted as requiring the invariant of 'port-uniqueness': *At any time in the evolution of some process, each of its ports may be accessed by at most one enabled action.* This invariant only guarantees to exclude mutual pre-emption of internal actions of a process; it does not take the processes' environment into account. The latter is fulfilled by the second condition on confluent composition; it imposes the additional structural constraint that complementary access to ports must be restricted, i.e. not visible to the outside, so it prevents from mutual pre-emption among visible and internal actions.

Shared redexes The idea is to use decompositions of processes into redex(es) and (multiple-hole) context as an intermediate vehicle between the characterization of port-uniqueness and the proof of confluence properties for spans of concurrently enable transitions. So, the roadmap for this section is the following: (1) characterize the possible decompositions of a process according to each of its possible pairs of concurrently enabled transitions; (2) identify the cases, where no redexes of the involved transitions are shared; (3) prove that in those cases diamond confluence holds. The formalization of port-uniqueness on top of process decompositions is deferred to the next section (cf Lemmas 18 and 21).

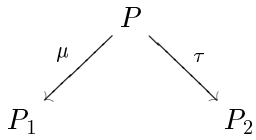
Depending on the kind of transitions—visible or internal—and by interpreting the distributed redex of internal transitions as a pair of local redexes, the necessary contexts have 2, 3, or 4 holes. Two different visible transitions cannot share a common redex; if at least one internal transition is involved, then sharing might occur. Furthermore, for visible transitions, we always know that its channel name is not captured by any hole-binding restriction. In the (polarized) π-calculus, a subtlety arises in the cases of scope extrusion, resulting in bound outputs via *OPEN* and τ-steps via *CLOSE*. In both cases, the derivative involves, according to Fact 7, that the context is manipulated to take the extrusion into account, whereas in the

latter the context has to be closed again for both parties of the communication. As in Fact 7, we only denote the τ-derivatives up to structural transformation ($\equiv$).

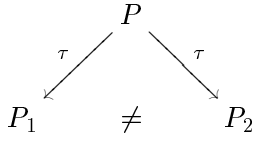
Fact 8 (Decomposition of spans). *Let $P \in \mathbb{F}_{\not\downarrow}$.*

1. *If*  *for $\mu_1 \neq \tau \neq \mu_2$, then there is*

   (a) $P = C[\pi_1.Q_1, \pi_2.Q_2]$
   with $\mathsf{chan}(\mu_i) = \mathsf{chan}(\pi_i) \notin \mathsf{hn}_i(C)$ *for* $i \in \{1,2\}$,
   $P_1 = C^{\not\downarrow}[Q_1, \pi_2.Q_2]$, *if* $\pi_1 = y^-\langle z\rangle$ *and* $z \in \mathsf{hn}_1(C)$
   $P_1 = C[Q_1, \pi_2.Q_2]$, *otherwise, and*
   $P_2 = C^{\not\downarrow}[\pi_1.Q_1, Q_2]$, *if* $\pi_2 = y^-\langle z\rangle$ *and* $z \in \mathsf{hn}_2(C)$
   $P_2 = C[\pi_1.Q_1, Q_2]$, *otherwise.*

2. *If*  *for $\mu \neq \tau$, then there is either of:*

   (a) $P = C[\pi.Q_1, y^-\langle z\rangle.Q_2, y^+(x).Q_3]$
   for some $x, y, z \in \mathbf{N}$ *with* $\mathsf{chan}(\mu) = \mathsf{chan}(\pi) \notin \mathsf{hn}_1(C)$,
   $P_1 = C^{\not\downarrow_1}[Q_1, y^-\langle z\rangle.Q_2, y^+(x).Q_3]$, *if* $\pi = y_1{}^-\langle b\rangle$ *and* $b \in \mathsf{hn}_1(C)$,
   $P_1 = C[Q_1, y^-\langle z\rangle.Q_2, y^+(x).Q_3]$, *otherwise, and*
   $P_2 = C[\pi.Q_1, Q_2, Q_3\{^z/_x\}]$, *if* $z \notin \mathsf{hn}_2(C)$,
   $P_2 \equiv (\boldsymbol{\nu} z)\, C^{\not\downarrow}[\pi.Q_1, Q_2, Q_3\{^z/_x\}]$, *otherwise.*
   (b) $P = C[y^-\langle z\rangle.Q_1, y^+(x).Q_2]$
   for some $x, y, z \in \mathbf{N}$ *with* $\mathsf{chan}(\mu) = y \notin \mathsf{hn}_1(C) \cup \mathsf{hn}_2(C)$,
   $P_1 = C[Q_1, y^+(x).Q_2]$, *and*
   $P_2 = C[Q_1, Q_2\{^z/_x\}]$, *if* $z \notin \mathsf{hn}_2(C)$,
   $P_2 \equiv (\boldsymbol{\nu} z)\, C^{\not\downarrow}[Q_1, Q_2\{^z/_x\}]$, *otherwise.*
   (c) $P = C[y^-\langle z\rangle.Q_1, y^+(x).Q_2]$
   for some $x, y, z \in \mathbf{N}$ *with* $\mathsf{chan}(\mu) = y \notin \mathsf{hn}_1(C) \cup \mathsf{hn}_2(C)$,
   $P_1 = C[y^-\langle z\rangle.Q_1, Q_2]$, *and*
   $P_2 = C[Q_1, Q_2\{^z/_x\}]$, *if* $z \notin \mathsf{hn}_2(C)$,
   $P_2 \equiv (\boldsymbol{\nu} z)\, C^{\not\downarrow}[Q_1, Q_2\{^z/_x\}]$, *otherwise.*

3. *If*  *, then there is either of:*

   (a) $P = C[a^-\langle z_a\rangle.Q_1, a^+(x_a).Q_2, b^-\langle z_b\rangle.Q_3, b^+(x_b).Q_4]$
   for some $a, b, x_i, z_i \in \mathbf{N}$,
   $P_1 = C[Q_1, Q_2\{^{z_a}/_{x_a}\}, b^-\langle z_b\rangle.Q_3, b^+(x_b).Q_4]$, *if* $z_a \notin \mathsf{hn}_1(C)$
   $P_1 \equiv (\boldsymbol{\nu} z_a)\, C^{\not\downarrow_a}[Q_1, Q_2\{^{z_a}/_{x_a}\}, b^-\langle z_b\rangle.Q_3, b^+(x_b).Q_4]$, *otherwise, and*
   $P_2 = C[a^-\langle z_a\rangle.Q_1, a^+(x_a).Q_2, Q_3, Q_4\{^{z_b}/_{x_b}\}]$, *if* $z_b \notin \mathsf{hn}_1(C)$
   $P_2 \equiv (\boldsymbol{\nu} z_b)\, C^{\not\downarrow_b}[a^-\langle z_a\rangle.Q_1, a^+(x_a).Q_2, Q_3, Q_4\{^{z_b}/_{x_b}\}]$, *otherwise.*

(b) $P = C[a^-\langle z_1\rangle.Q_1, a^-\langle z_2\rangle.Q_2, a^+(x).Q_3]$
for some $a, x, z_i \in \mathbf{N}$,
$P_1 = C[Q_1, a^-\langle z_2\rangle.Q_2, Q_3\{^{z_1}/_x\}]$, *if* $z_1 \notin \mathsf{hn}_1(C)$,
$P_1 \equiv (\boldsymbol{\nu} z_1)\, C^{\not\ni_1}[Q_1, a^-\langle z_2\rangle.Q_2, Q_3\{^{z_1}/_x\}]$, *otherwise, and*
$P_2 = C[a^-\langle z_1\rangle.Q_1, Q_2, Q_3\{^{z_1}/_x\}]$, *if* $z_2 \notin \mathsf{hn}_2(C)$
$P_2 \equiv (\boldsymbol{\nu} z_2)\, C^{\not\ni_2}[a^-\langle z_1\rangle.Q_1, Q_2, Q_3\{^{z_1}/_x\}]$, *otherwise.*

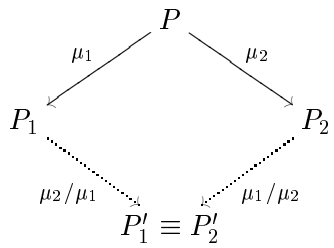
(c) $P = C[a^+(x_1).Q_1, a^+(x_2).Q_2, a^-\langle z\rangle.Q_3]$
for some $a, x_i, z \in \mathbf{N}$.
$P_1 = C[Q_1\{^{z}/_{x_1}\}, a^+(x_2).Q_2, Q_3]$, *if* $z \notin \mathsf{hn}_3(C)$
$P_1 \equiv (\boldsymbol{\nu} z)\, C^{\not\ni}[Q_1\{^{z}/_{x_1}\}, a^+(x_2).Q_2, Q_3]$, *otherwise, and*
$P_2 = C[a^+(x_1).Q_1, Q_2\{^{z}/_{x_2}\}, Q_3]$, *if* $z \notin \mathsf{hn}_3(C)$
$P_2 \equiv (\boldsymbol{\nu} z)\, C^{\not\ni}[a^+(x_1).Q_1, Q_2\{^{z}/_{x_2}\}, Q_3]$, *otherwise.*

Note that in case 1.a $\mu_1 = \mu_2$ and $\mathsf{chan}(\mu_1) = \mathsf{chan}(\mu_2)$ is possible. Similarly, in case 2.a $\mathsf{chan}(\mu) = a$ and in case 3.a $a = b$ are possible. The cases 2.b, 3.b, and 3.c, respectively, represent overlapping redexes, i.e. sharing; the a-cases indicate 'concurrency' of the transitions.

Definition 9 (Overlap and concurrency). *Let* $P \in \mathbb{F}_{\not\alpha}$. *Any span decomposition of* P *matching one of the a-cases of Fact 8 is called* overlapping-free. *Two transitions are* concurrent, *if their span induces an overlapping-free decomposition.*

Concurrent transitions, as defined above, are precisely characterized by the fact that they do not share (local) redexes, which could be either send- or receive-guards. (Remember that the cases for replicated guards only increase the number of cases without requiring different analysis.) Therefore, concurrent transitions should be executable in either order, which we state as property of closing 'diamonds'. Sometimes, it is necessary to adjust the second of two subsequent labeled transitions; as in the Introduction, we record this fact by the notation μ_2/μ_1 for the case where μ_2 is performed after the concurrently enabled μ_1.

Definition 10 ($\diamondsuit$-Confluence). *Let* $P \in \mathbb{F}_{\not\alpha}$. *If* $P \xrightarrow{\mu_1} P_1$ *and* $P \xrightarrow{\mu_2} P_2$ *are concurrently enabled transitions with* $P_1 \neq P_2$, *and the diagram to the right can be closed for some process* P' *(up to structural congruence* $\equiv$*), then the transition pair is called* $\diamondsuit$-confluent. *A process* P *is called* $\diamondsuit$*-confluent, if for all of its derivatives* Q *all of* Q*'s concurrently enabled pairs of transitions are* $\diamondsuit$*-confluent.*



In the (polarized) π-calculus, if two concurrently enabled bound outputs carry the same object, then, when performing both one after another, the second output will be performed as a free output (*object precedence* in [Pri96]). Therefore, let $y_1{}^-\langle \boldsymbol{\nu} z\rangle \,/\, y_2{}^-\langle \boldsymbol{\nu} z\rangle := y_1{}^-\langle z\rangle$, and $\mu_1 \,/\, \mu_2 := \mu_1$, otherwise. This is the only case for α-free processes, where two concurrently enabled actions may influence each others occurrence without disabling each other.

Lemma 11 (Diamond). *Let* $P \in \mathbb{F}_{\not\alpha}$. *If* $P \xrightarrow{\mu_1} P_1$ *and* $P \xrightarrow{\mu_2} P_2$ *are concurrent transitions, then they are* $\diamondsuit$*-confluent.*

Proof. By subsequently performing the transitions (which is possible since they are concurrent) and passing them through the context $C[\cdot]$ by applying *PAR* and *RES* along the (static) operators of $C[\cdot]$, according to the case analysis along the decomposition lemma 8. □

5 Typing port-uniqueness

We introduce a static typing system for process terms that guarantees unique access of the process to its ports. As main result, we prove subject reduction and precise decompositions with respect to port-uniqueness, which allow us to derive a strong diamond lemma that is valid for all concurrently enabled transitions of all derivatives of well-typed processes.

Our type system allows us to derive judgements of the form $\Delta \vdash P$ where Δ is a typing context and $P \in \mathbb{P}$ is a processq, Typing contexts are simply represented as set of ports.

Definition 12 (Typing contexts). *A* typing context Δ *is a set of ports; we use the usual set operations* $\in, \subseteq, \cup, \cap, \setminus$, *and* $\oplus, \ominus$, *as operations on contexts.*

Regarding contexts as functions $\Delta : \mathbf{N} \to 2^{\{+,-\}}$ *from names to sets of polarities, we also use the notation* $\Delta(y)$ *for denoting the polarities which are associated with name* y *in context* Δ, *e.g.* $\Delta(x) = \{-\} :\Leftrightarrow (x^- \in \Delta \wedge x^+ \notin \Delta)$. *Furthermore, let* $\mathsf{dom}(\Delta)$ *denote the set of names that occur polarized in* Δ, *i.e.* $\mathsf{dom}(\Delta) := \{x \in \mathbf{N} \mid \Delta(x) \neq \emptyset\}$.

The intuition of typing judgements $\Delta \vdash P$ is twofold: (1) process P is currently allowed to access only the ports in Δ, and (2) it accesses these ports uniquely. Subject reduction then provides a mechanism to inherit that property onto P's derivatives.

The inference system for judgements is given by the rules in Table 3. *U-NIL* behaves well in all contexts since it does not access any port at all. *U-PAR* is in direct analogy to confluent

$$\textit{U-NIL:}\quad \frac{}{\Delta \vdash \mathbf{0}} \qquad \textit{if } \Delta \subseteq \mathbf{N}^\star$$

$$\textit{U-PAR:}\quad \frac{\Delta_1 \vdash P_1 \qquad \Delta_2 \vdash P_2}{\Delta_1 \oplus \Delta_2 \vdash P_1 | P_2} \qquad \textit{if } \Delta_1 \cap \Delta_2 = \emptyset$$

$$\textit{U-OUT:}\quad \frac{\Delta \ominus z^\star \vdash P}{\Delta \vdash y^-\langle z^\star\rangle.P} \qquad \textit{if } y^- \in \Delta \textit{ and } z^\star \in \Delta$$

$$\textit{U-INP:}\quad \frac{\Delta \oplus x^\star \vdash P}{\Delta \vdash y^+(x^\star).P} \qquad \textit{if } y^+ \in \Delta \textit{ and } x \notin \mathsf{dom}(\Delta)$$

$$\textit{U-REP:}\quad \frac{\{x^\star\} \vdash P}{\{y^+\} \vdash \,!\, y^+(x^\star).P}$$

$$\textit{U-RES:}\quad \frac{\Delta \oplus \{y^+, y^-\} \vdash P}{\Delta \vdash (\boldsymbol{\nu} y)\, P} \qquad \textit{if } y \notin \mathsf{dom}(\Delta)$$

$$\textit{U-ALPHA:}\quad \frac{\Delta \vdash \widehat{P}}{\Delta \vdash P} \qquad \textit{if } P =_\alpha \widehat{P}$$

Table 3: Typing rules: Port-uniqueness in $\mathbb{F}$

composition: components in a parallel composition have to use disjoint sets of ports. In both *U-INP* and *U-OUT* the channel y has to be allowed by Δ with the required polarity, *U-INP* indicates that the continuation of an input prefix has acquired an additional capability, and *U-OUT* remembers that the continuation of an output prefix has lost some capability—this is necessary in name-passing settings, since it ensures that the sender's continuation of some port will no longer be allowed to use that port itself. Sending a port has to be forgetful for the sender. *U-REP* is the most complicated rule: because replication is like parallel composition, processes may only have one free name—the one on which they listen—and must forget it after having used it in order to maintain the invariant of port-uniqueness; this initial name can of course be used for receiving further names for interacting wiht its environment. *U-RES* introduces two complementary halves of a name upon restriction, while checking that the generated name is fresh. *U-ALPHA* allows to rename bound names in order to adjust processes to match the side-condition of rules *U-RES* and *U-INP*. Note that the rules *U-OUT* and *U-INP* are strictly more generous than the conditions on unique handling and bearing of a name [Mil93] with respect to the two conditions that we mentioned in the Introduction.

Lemma 13 (Basic properties of typing). *Let $\Delta \vdash P$. Then:*

1. $\mathsf{fn}(P) \subseteq \mathsf{dom}(\Delta)$.
2. *If $x^\star \in \mathbf{N}^\star \setminus \Delta$, then $\Delta \oplus x^\star \vdash P$.*
3. *If $x^\star \in \Delta$ with $x^\star \notin \mathsf{fp}(P)$, then $\Delta \ominus x^\star \vdash P$.*
4. *If $x \in \mathbf{N}$, then $\Delta \setminus \{x^+, x^-\} \vdash (\boldsymbol{\nu} x)\, P$.*
5. *If $P \xrightarrow{y^-\langle z^\star\rangle}$ or $P \xrightarrow{y^-(\boldsymbol{\nu} z^\star)}$, then $y^- \in \Delta$.*
6. *If $P \xrightarrow{y^+(x^\star)}$, then $y^+ \in \Delta$.*
7. *If $P \equiv \widehat{P}$, then $\Delta \vdash \widehat{P}$.*

Proof. Straightforward by induction. □

Note that $\Delta_1 \oplus \Delta_2 \vdash P_1|P_2$ with $\Delta_1 \cap \Delta_2 = \emptyset$ does not imply that also $\mathsf{dom}(\Delta_1) \cap \mathsf{dom}(\Delta_2) = \emptyset$. This is necessary, if we do not want to forbid any communication between parallel components.

It is important to state precisely, whether a substitution σ preserves a typing judgement $\Delta \vdash P$, i.e. whether $\Delta \vdash P$ implies $\Delta\sigma \vdash P\sigma$. Let name-substitutions σ be extended to $(x^\star)\sigma \stackrel{\text{def}}{=} x\sigma^u$ and $\Delta\sigma \stackrel{\text{def}}{=} \{\, (x^\star)\sigma \mid x^\star \in \Delta \,\}$. Problematic substitutions are those which destroy the uniqueness information about ports, i.e. those which collapse formerly different ports. So, in order to be compatible with a typing judgement using Δ, we require that a substitution σ be injective with respect to that Δ.

Definition 14 (Compatible substitutions). *A substitution σ is called* Δ-injective *(or:* Δ-compatible*), if $x\sigma = y\sigma$ implies either $x = y$, or $x \neq y$ and $\Delta(x) \cap \Delta(y) = \emptyset$.*

In order to prove that σ is Δ-injective, check for all $x \neq y$ with $x\sigma = y\sigma$ that $\Delta(x) \cap \Delta(y) = \emptyset$. For example, if $\Delta = \{y^-, y^+\}$, then every substitution σ is Δ-injective. If $\Delta = \{y^-, x^-\}$, then both $\{{}^{y}/_{x}\}$ and $\{{}^{x}/_{y}\}$ are not Δ-injective.

Lemma 15. *If $\Delta \vdash P$ and σ is Δ-injective, then $\Delta\sigma \vdash P\sigma$.*

Proof. By checking that each of the typing rules is either directly preserved by Δ-injective substitutions, or can be adjusted by means of *U-ALPHA* to rename some bound names. □

5.1 Context transitions

A context transition $\Delta \xrightarrow{\mu} \Delta'$, as defined by the rules in Table 4 reminiscent of the side-conditions of the rules *U-INP* and *U-OUT*, specifies of a process constrained by the context Δ that it is allowed to engage in the action μ, and that its continuation process will be constrained by Δ'. This intuition should be immediately clear for the first three rules. But, also for *C-BOUT*, it is straightforward since bound output means that two halves of the newly created name are generated, and exactly one of them is given away, i.e. the continuation context $\Delta' = \Delta \oplus \overline{z^\star}$ is in fact the result of $(\Delta \oplus \{z^+, z^-\}) \ominus z^\star$.

Lemma 16. *Let* $\Delta \xrightarrow{\mu} \Delta'$.

1. *(a) If* $\mu = y^-\langle z^\star\rangle$, *then* $y^- \in \Delta$ *and* $\Delta' = \Delta \ominus z^\star$.
   (b) If $\mu = y^-(\boldsymbol{\nu} z^\star)$, *then* $y^- \in \Delta$ *and* $\Delta' = \Delta \oplus \overline{z^\star}$.
   (c) If $\mu = y^+(x^\star)$, *then* $y^+ \in \Delta$ *and* $\Delta' = \Delta \oplus x^\star$.
   (d) If $\mu = \tau$, *then* $\Delta' = \Delta$.
2. *(a) If* $x^\star \in \mathbf{N}^\star \setminus \Delta$, *then* $\Delta \oplus x^\star \xrightarrow{\mu} \Delta' \oplus x^\star$.
   (b) If $x^\star \in \Delta$ *with* $x \notin \mathsf{n}(\mu)$, *then* $\Delta \ominus x^\star \xrightarrow{\mu} \Delta' \ominus x^\star$.
   (c) If $z \notin \mathsf{n}(\mu)$, *then* $\Delta \setminus \{z^+, z^-\} \xrightarrow{\mu} \Delta' \setminus \{z^+, z^-\}$.

The first set of propositions in this lemma describe the backwards-generation of side-conditions from the existence of a context transition. The second set records weakening and two versions of strengthening for context transitions. All propositions follow directly from the definition.

We show that typability $\Delta \vdash P$ is preserved under transition $P \xrightarrow{\mu} P'$. However, the context for well-typedness changes under visible transitions from Δ to some Δ'. Similar to the side conditions in the definition of bisimulation in the context of name-passing, we assume that bound names in a label are 'sound' with respect to the context Δ that constrains the actions of process P via typability. This is necessary for the proof, but is not critical for the proposition itself since we can always rename bound names in transitions in order to match the required side conditions.

$$\textit{C-TAU:}\quad \frac{}{\Delta \xrightarrow{\tau} \Delta}$$

$$\textit{C-INP:}\quad \frac{y^+ \in \Delta \qquad x \notin \mathsf{dom}(\Delta)}{\Delta \xrightarrow{y^+(x^\star)} \Delta \oplus x^\star}$$

$$\textit{C-OUT:}\quad \frac{y^- \in \Delta \qquad z^\star \in \Delta}{\Delta \xrightarrow{y^-\langle z^\star\rangle} \Delta \ominus z^\star}$$

$$\textit{C-BOUT:}\quad \frac{y^- \in \Delta \qquad z \notin \mathsf{dom}(\Delta)}{\Delta \xrightarrow{y^-(\boldsymbol{\nu} z^\star)} \Delta \oplus \overline{z^\star}}$$

Table 4: Context transitions

Lemma 17 (Subject reduction). *Let $P \in \mathbb{P}$ and $\Delta \subseteq \mathbf{N}^\star$.*
If $\Delta \vdash P \xrightarrow{\mu} P'$ with $\mathsf{bn}(\mu) \cap \mathsf{dom}(\Delta) = \emptyset$, then $\Delta \xrightarrow{\mu} \Delta' \vdash P'$ for some $\Delta' \subseteq \mathbf{N}^\star$.

Proof. By induction on the inference of $P \xrightarrow{\mu} P'$ and exploiting the last derivation step of $\Delta \vdash P$ in each case. □

In the polarized π-calculus, it is crucial to require that not only the binding object occurrences of inputs, but also the non-binding object occurrences of outputs are tagged. Otherwise, the type system would have to 'guess' for the derivation of prefixes which of the polarities is received and sent, respectively. Then, we would no longer be able to prove subject reduction. For example, $P = y^-\langle z\rangle.z^-\langle w\rangle \,|\, y^+(x).x^-\langle w\rangle$, is typable in some Δ, if the type system was allowed to guess the polarities for z in the left component and for x in the right component independently from each other. However, subject reduction would not hold since $P \xrightarrow{\tau} P' := z^-\langle w\rangle \,|\, z^-\langle w\rangle$ is possible, whereas there is no Δ' such that $\Delta' \vdash P'$ is derivable. Consequently, we need to reject such processes P.

We show that typability $\Delta \vdash P$ provides precise structural information about P. Here, and in the remainder of the paper, we assume that $P \in \mathbb{F}_{\not\alpha}$ is α-free, which enables us to decompose processes due to existing transitions without involving the issue of α-conversion.

Lemma 18 (Port-unique decomposition I). *Let $\Delta \vdash P$.*

1. *If $P \xrightarrow{y^-\langle z^\star\rangle} P'$ and $y^+ \notin \Delta$,*
   then there is $P = C[\, y^-\langle z^\star\rangle.Q \,]$ with $P' = C[\, Q \,]$
   and $y \notin \mathsf{fn}(C) \cup \mathsf{bn}(C)$ and $z \notin \mathsf{bn}(C)$.
2. *If $P \xrightarrow{y^-(\boldsymbol{\nu} z^\star)} P'$ and $y^+ \notin \Delta$,*
   then there is $P = C[\, y^-\langle z^\star\rangle.Q \,]$ with $P' = C^{\not z}[\, Q \,]$
   and $y \notin \mathsf{fn}(C) \cup \mathsf{bn}(C)$ and $z \notin \mathsf{fn}(C)$, but $z \in \mathsf{bn}(C)$,
3. *If $P \xrightarrow{y^+(x^\star)} P'$ and $y^- \notin \Delta$,*
   then there is $P = C[\, y^+(x^\star).Q \,]$ with $P' = C[\, Q \,]$
   and $y \notin \mathsf{fn}(C) \cup \mathsf{bn}(C)$ and $x \notin \mathsf{fn}(C) \cup \mathsf{bn}(C)$.
4. *If $P \xrightarrow{\tau} P'$,*
   then there is $P = C[\, y^-\langle z^\star\rangle.Q_1, y^+(x^\star).Q_2 \,]$ for some $y \in \mathbf{N}$ with
   $P' \equiv (\boldsymbol{\nu} z)\, C^{\not z}[\, Q_1, Q_2\{^z/_x\} \,]$, if $z \in \mathsf{bn}_1(C)$,
   $P' = C[\, Q_1, Q_2\{^z/_x\} \,]$, otherwise, and
   either $y \in \mathsf{bn}_1(C) \cap \mathsf{bn}_2(C)$, or $y \notin \mathsf{fn}(C) \cup \mathsf{bn}_1(C) \cup \mathsf{bn}_2(C)$.
5. *If $P \xrightarrow{\tau} P'$,*
   then either $P \equiv C[\, (\boldsymbol{\nu} y)\,(\, y^-\langle z^\star\rangle.Q_1 \,|\, y^+(x^\star).Q_2 \,)\,]$,
   or $P \equiv C[\, y^-\langle z^\star\rangle.Q_1 \,|\, y^+(x^\star).Q_2 \,]$ with $y \notin \mathsf{fn}(C) \cup \mathsf{bn}(C)$.

where $C[\cdot]$ and $C[\cdot,\cdot]$ are process contexts.

Proof. By induction on the inference of the transition starting from P and exploiting the derivation of $\Delta \vdash P$. □

Note that subject names y do not occur free in $C[\cdot]$; this expresses uniqueness. However, the object part of labels also exhibits interesting properties in each case. In particular, note that in the case of free output the object may still occur free in $C[\cdot]$, the reason being that an action may always carry at most one (polarized) half of a name. In the case where the object is bound, it must not occur free in $C[\cdot]$ in order to avoid capture.

5.2 Closing up

Now, we interpret the information of the context Δ that y occurs with both polarities such that, after restricting P on y, the resulting $(\boldsymbol{\nu} y)\, P$ will exhibit certain behavioral properties. Closing typed processes in their context is completely analogous to the π_0-calculus.

Definition 19 (Closure). *Let Δ be an arbitrary context. Then, the set of names that occur in Δ with both polarities, defined as*

$$\Delta^{\pm} \stackrel{\text{def}}{=} \{x \in \mathbf{N} \mid \{x^-, x^+\} \subseteq \Delta\},$$

is called the closure *of Δ. If $\Delta \vdash P$ then*

$$\Delta(P) \stackrel{\text{def}}{=} (\boldsymbol{\nu} \Delta^{\pm})\, P$$

is called the Δ-closure *of P; if $\Delta^{\pm} = \emptyset$, then P is called* Δ-closed.
Let $\Delta \Vdash P$ denote $\Delta \vdash P$ where P is Δ-closed.

We show that labeled transition preserves the closure of both contexts and processes.

Lemma 20 (Closed subject reduction). *Let $P \in \mathbb{P}$ and $\Delta \subseteq \mathbf{N}^\star$.*

1. *If Δ is closed and $\Delta \xrightarrow{\mu} \Delta'$, then Δ' is closed.*
2. *If $\Delta \Vdash P \xrightarrow{\mu} P'$ and $\mathsf{bn}(\mu) \cap \mathsf{dom}(\Delta) = \emptyset$, then $\Delta \xrightarrow{\mu} \Delta' \Vdash P'$.*

Proof. By inspection of the defining rules for context transitions: Closedness can only be invalidated by either adding to a port, which is already contained in the context, its complement, or by adding two complementary ports. The latter is completely excluded by restricting communications to one-port-at-a time. The former cannot happen since increase of context by *C-INP* and *C-BOUT* always assumes that the name of the new port is fresh. □

Here, it is essential that *double-polarity* communication is forbidden; its permission would invalidate the preservation of closedness. If a process $P := y^+(x^{\pm}).(\, x^-\langle z\rangle \,|\, x^+(w)\,)$ would be allowed to receive both polarities of x in one communication. Then P would be typable in our system with $\Delta = \{y^+, z^\star\}$, and P would be Δ-closed. However, by allowing

$$P \xrightarrow{y^+(x^{\pm})} x^-\langle z^\star\rangle \,|\, x^+(w^\star) =: P' \quad \text{with} \quad \Delta \xrightarrow{y^+(x^{\pm})} \Delta \oplus \{x^+, x^-\} =: \Delta'$$

P' would be typable in Δ', but P' would not be Δ'-closed. Note that the problem is not present, if we use two subsequent communications to receive the needed ports, since we would have to use two different input variables to do that, which would then also be distinguished in Δ'. For further discussion, see Section 7.

Closed judgements $\Delta \Vdash P$ exhibit more information about the internal structure of process P than the corresponding 'open' judgements $\Delta \vdash P$.

Lemma 21 (Port-unique decomposition II). *Let $\Delta \Vdash P$.*

1. *If $P \xrightarrow{y^-\langle z^\star\rangle} P'$,*
   then there is $P = C[\, y^-\langle z^\star\rangle.Q\,]$ with $P' = C[\, Q\,]$
   and $y \notin \mathsf{fn}(C) \cup \mathsf{hn}(C)$ and $z \notin \mathsf{hn}(C)$.

2. *If* $P \xrightarrow{y^-(\boldsymbol{\nu} z^\star)} P'$,
then there is $P = C[\, y^-\langle z^\star\rangle.Q\,]$ *with* $P' = C^{\not z}[\, Q\,]$
and $y \notin \mathsf{fn}(C) \cup \mathsf{hn}(C)$ *and* $z \notin \mathsf{fn}(C)$, *but* $z \in \mathsf{hn}(C)$,
3. *If* $P \xrightarrow{y^+(x^\star)} P'$,
then there is $P = C[\, y^+(x^\star).Q\,]$ *with* $P' = C[\, Q\,]$
and $y \notin \mathsf{fn}(C) \cup \mathsf{hn}(C)$ *and* $x \notin \mathsf{fn}(C) \cup \mathsf{hn}(C)$.
4. *If* $P \xrightarrow{\tau} P'$,
then there is $P = C[\, y^-\langle z^\star\rangle.Q_1, y^+(x^\star).Q_2\,]$ *for some* $y \in \mathbf{N}$ *with*
$P' \equiv (\boldsymbol{\nu} z)\, C^{\not z}[\, Q_1, Q_2\{^z/_x\}\,]$, *if* $z \in \mathsf{hn}_1(C)$,
$P' = C[\, Q_1, Q_2\{^z/_x\}\,]$, *otherwise, and* $y \in \mathsf{hn}_1(C) \cap \mathsf{hn}_2(C)$.
5. *If* $P \xrightarrow{\tau} P'$, *then* $P \equiv C[\, (\boldsymbol{\nu} y)\,(\, y^-\langle z^\star\rangle.Q_1 \mid y^+(x^\star).Q_2\,)\,]$
and $P' \equiv C[\, (\boldsymbol{\nu} y)\,(\, Q_1 \mid Q_2\{^z/_x\}\,)\,]$.

where $C[\cdot]$ *and* $C[\cdot,\cdot]$ *are process contexts.*

Proof. By application of Lemma 18, followed by the exclusion of possible cases due to the closure information of the judgement. □

6 Behavioral properties of well-typed processes

We collect the behavioral properties that hold for closed typable processes. We are especially interested in the notion of confluence and whether they are guaranteed by typability.

Theorem 22 (τ-inertness). *If* $\Delta \Vdash P \xrightarrow{\tau} P'$, *then* $P \approx P'$.

Proof. Because of Lemma 21, we have a decomposition $P \equiv C[\, Q\,]$ for some process context $C[\cdot]$ and $Q = (\boldsymbol{\nu} y)\,(\, y^-\langle z^\star\rangle.Q_1 \mid y^+(x^\star).Q_2\,)$ with $P' \equiv C[Q']$ for $Q' = (\boldsymbol{\nu} y)\,(\, Q_1 \mid Q_2\{^z/_x\}\,)$. Observe that, due to the structure of Q, the transition

$$Q = (\boldsymbol{\nu} y)\,(\, y^-\langle z^\star\rangle.Q_1 \mid y^+(x^\star).Q_2\,) \xrightarrow{\tau} (\boldsymbol{\nu} y)\,(\, Q_1 \mid Q_2\{^z/_x\}\,) = Q'$$

is the only one that Q exhibits. Immediately, we know that $Q \approx Q'$. Since bisimulation is preserved by restriction and parallel composition, we also have $P \equiv C[Q] \approx C[Q'] \equiv P'$, and since this analysis is stable under name-substitution, we conclude the proof. □

We show that typability of a process witnesses the absence of transitions that are 'concurrently enabled', but not 'concurrent'. Then, we know that redexes are never shared and all concurrently enabled actions can be performed in either order.

Proposition 23 (Concurrency). *Let* $\Delta \Vdash P$. *If* $P \xrightarrow{\mu_1} P_1$ *and* $P \xrightarrow{\mu_2} P_2$ *are concurrently enabled transitions, then they are concurrent.*

Proof. According to Definition 9, We check that only a-cases for the decomposition of the span (cf Fact 8) are allowed by $\Delta \Vdash P$. □

Theorem 24 ($\diamondsuit$-Confluence). *Let* $\Delta \Vdash P$. *Then* P *is* $\diamondsuit$*-confluent.*

Proof. By Proposition 23, the diamond property (Lemma 11), and labeled subject reduction (Lemma 20) for closed processes. □

In [Nes96], we have also studied typing systems for a merely signal-passing setting like choice-free CCS (typing rules are much simpler, since no names are passed around) and proved that both Milner's weak confluence and Groote-Sellink's strong confluence hold.

7 Related typing approaches

The polarized π-calculus [Ode95] may be seen as a simplified syntactic reformulation, of the subtyped π-calculus [PS96], which is based on typing and subtyping of channels. Consequently, no type system is provided for preventing run-time errors, i.e. for guaranteeing that the side-condition in the *COM*/*CLOSE*-rules is always met for typable processes. For the purpose of [Ode95], and also for ours, the untyped version with the additional side-condition sufficed. Similarly, our typing system for port-uniqueness in the polarized π-calculus corresponds to some extension of the subtyping system. For the presentation in this paper, we preferred to use the polarized calculus since it provides a rather simple and minimal solution.

Subtyping In the π-calculus [PS96] with subtyping, the main idea is that directionality modes (i.e. polarities) define channel *types* T. Moreover, the bidirectional mode b (corresponds to $\pm$) is a natural *subtype* for both unidirectionality modes r and w (corresponds to $+$ and $-$): a bidirectional channel may be safely used in any context that would expect any of the its unidirectional counterparts.

The technical development based on this idea is using a (polyadic) calculus with recursively defined type-levels by associating with each level some *polarity* (one of $+$, $-$, or $\pm$). The outermost polarity (denoted by $\mathcal{I}(\mathsf{T})$) indicates the direction in which the channel itself may be used within the scope of its binding; the inner polarities specify the polarities of the channels' possible objects. All binding occurrences of names are typed; the typed binding of x in $y^+(x{:}[[\mathsf{T}]^{\pm}]^-).P$ describes that P would be enabled to send some name z of type $[\mathsf{T}]^{\pm}$ along x, i.e. along the name that P might have received from y; a receiver of z may use this name for both sending and receiving names of type T. The typing system in [PS96] records all introduced name-bindings of a process within typing assumptions Γ and checks that each channel access according to the structure of the process term is permitted, and that parallel compositions can be controlled by correctly combining type-bindings. The following simplified rule specifies run-time errors. Let Γ_i be functions from names to names.

$$\frac{\Gamma_1 \vdash P_1 \xrightarrow{y^-\langle z\rangle} P_1' \qquad \Gamma_2 \vdash P_2 \xrightarrow{y^+(x:\mathsf{X})} P_2'}{\Gamma_1 + \Gamma_2 \vdash P_1 \mid P_2 \xrightarrow{\tau} \text{error}} \quad \text{if } \neg \begin{pmatrix} & \mathcal{I}(\,\Gamma_1(y)\,) \leq - \\ \wedge & \mathcal{I}(\,\Gamma_2(y)\,) \leq + \\ \wedge & \Gamma_1(z) \leq \mathsf{X} \end{pmatrix}$$

If any of the three side-conditions is violated, we may infer a run-time error. The first two clauses refer to the correct *current* access to the channel y: each party must be allowed to perform its local access. The third clause refers to correct *future* accesses of the receiver P_2' to its received channel x: the sender must be able (w.r.t. its typing context) to supply at least the capabilities for z that P_2 requires in order to be enabled to use x correctly.

Towards port-uniqueness In the standard π-calculus, $P = y^-\langle z\rangle.z^-\langle w\rangle \mid y^+(x).x^-\langle w\rangle$ with $P \rightarrow P' := z^-\langle w\rangle \mid z^-\langle w\rangle$ is a critical example: P would be typable, according to the system presented in this paper, when the type system 'guesses' that the left component emits z^+ (such that its continuation is allowed to use z^- as required) and the right component receives x^- such that the composition becomes typable, while P' is not typable, thus subject reduction is invalidated. Using explicit polarities, we prevented the left component from cheating (passing on the wrong polarity to the right component by tagging the output z as z^+ in order to be typable itself); the resulting polarized term would no longer exhibit the reduction to P' since the side-condition on matching object tags would not be fulfilled.

The subtyping calculus guarantees correct use of polarities according to the above side-conditions for deriving communications. There, the critical example would be rephrased $P = y^-\langle z\rangle.z^-\langle w\rangle \mid y^+(x{:}\mathsf{X}).x^-\langle w\rangle$, and with $\Gamma = \{w{:}\mathsf{W}, y{:}\mathsf{Y}, z{:}\mathsf{Z}\} \vdash P$, we would know that

- $\mathsf{Y} \leq [\mathsf{Z}]^-$ for the output on y, and $\mathsf{Z} \leq [\mathsf{W}]^-$ for the output on z, and
- $\mathsf{Y} \leq [\mathsf{X}]^+$ for the input from y, and $\mathsf{X} \leq [\mathsf{W}]^-$ for the output on x,

where the two conditions for Y provide a rather useful connection of the types Z and X: by Lemma 2.4.8 in [PS96], we know that $\mathsf{Z} \leq \mathsf{X}$.

An extension of the type system towards uniqueness (on top of that for checking correct use of polarity) would be similar to our system, i.e. from the typing context of output continuations we remove the capabilities that are sent, and parallel composition has to be strengthened appropriately. Equivalent to explicit polarities, the above $\mathsf{Z} \leq \mathsf{X}$ then helps in recognizing the critical example as ill-typed since the output-capability for z would have to be removed by the first output in the left component such that its continuation will no longer be typable.

Closing up vs. typed observations In a typed calculus, it is more natural to adapt the notion of observation to the idea of typing. Consequently, visible transitions shall only be regarded as observable if the outside is a 'legal' environment, i.e. allowing a well-typed composition of the observing context and the observed process [PS96, KPT96, PS97, San96b, Ama97]. As a consequence, typed observations give rise to coarser equivalences that have also proven to be useful in various cases.

With respect to unique use of ports, a channel is then only observable if not both of its ends may be used by the observed process itself, such that the complement could in principle be provided by the observing context. We have forbidden double-polarity output since subject reduction for closed judgements was invalidated. Closing up was necessary for constraining the visible transitions in the spans of confluence diagrams. So, if we adopt a typed observation principle for confluence instead of 'closing up', we may admit double-polarity outputs.

Linear(ized) types In contrast to channels with linear types, which can be used at most once throughoput their lifetime, our notion of unique access corresponds to channels with *linearized* types, as suggested in [KPT96, THK94, KNY95], which can be reused after each (unique) usage. In [KPT96], an encoding of linearized types by means of linear types, mixed with an encoding of synchronous into asynchronous output, was proposed: subsequent access to some unique channel would be translated into the use of subsequently created fresh linear channels. However, linearized types and the corresponding encoding have not yet been studied in more detail, in particular, not with respect to confluence issues.

In the very recent approaches of [San96b, Ama97], only the receiving end of a channel is required to be accessed uniquely, much as with the notion of unique bearing due to [Mil93] and with 'definitions' in the join-calculus [FG96]. In our system, both ends are controlled, but the subsequent behaviors are not restrained. Similar ideas can be found with a 'linear' variant of *receptiveness* in [San96b] and with *mutex channels* in [Kob97]: the former additionally requires that receptors always have the same behavior; the latter additionally requires that messages are always present, while requiring nothing for receptors.

8 Conclusion

We have shown how to set up a type system in the polarized π-calculus that statically guarantees unique access to ports throughout a processes' lifetime. We have also discussed in detail, how to adopt the idea of controlling port-uniqueness in the setting of a calculus with subtyping. The generalization of our setting to polyadic messages is straightforward. The only complication is the distinction between free and bound objects in polyadic outputs, where free outputs decrease the typing context, while bound outputs (as in the monadic case) increase it.

We have derived strong diamond confluence properties for well-typed processes and also gave a very simple proof of τ-inertness for well-typed processes. Also, it is straightforward to deal with partial confluence in the sense that the diamond property holds only for spans, where at least one transition is generated for communication on a unique port (and under the side-condition that the other transition is not complementing the former).

Of course, we cannot recognize all confluent process systems by means of a static typing system since all interesting notions of confluence, and also port-uniqueness and other name disciplines, for concurrent systems of message-passing processes are undeciable. However, a combination of type systems for unique/linearized channels with others for

- linear channels with at most one sender and one receiver,
- functional channels with one persistent receiver and multiple senders, and
- mutex channels with one persistent message and mutiple receivers,

might be 'sufficiently complete' by covering enough cases for practical purposes.

In order to get even closer to completeness, it is likely to be advantageous to include causal information within type systems. For example, let $P = (\boldsymbol{\nu} b)\,(a^-\langle z\rangle.b^-\langle y\rangle \mid b^+(x).a^-\langle w\rangle)$. Although P is not typable since a^- occurs on both sides of the composition, P is port-unique (and confluent) since the occurrence of a^- in the left component has *precedence* (in the terminology of [Pri96]) over the occurrence of a^- in the right component. Thus, at any time, only one of them will be enabled. We believe that an adaptation of the recently developed more graphical notions of types as in [Yos96, Kob97] could be exploited for an enhancement our type system, since they take causal information into account.

Acknowledgments

We would like to thank Cédric Fournet for reading a draft of this paper, Benjamin Pierce and Davide Sangiorgi for discussions on the topic in an early stage of the work.

References

[Ama97] R. M. Amadio. *An Asynchronous Model of Locality, Failure, and Process Mobility.* Rapport interne, LIM, Marseille, 1997.

[BC88] G. Boudol and I. Castellani. *A Non-Interleaving Semantics for CCS Based on Proved Transitions.* Fundamenta Informaticae, XI:433–452, 1988.

[DP92] P. Degano and C. Priami. *Proved Trees.* In Kuich [Kui92], pages 629–640.

[FG96] C. Fournet and G. Gonthier. *The Reflexive Chemical Abstract Machine and the Join-Calculus.* In POPL'96, pages 372–385, 1996.

[GS96] J. F. Groote and M. P. A. Sellink. *Confluence for Process Verification.* Theoretical Computer Science, 170(1–2):47–82, 1996.

[HY95] K. Honda and N. Yoshida. *On Reduction-Based Process Semantics.* Theoretical Computer Science, 152(2):437–486, 1995.

[KNY95] N. Kobayashi, M. Nakade, and A. Yonezawa. *Static Analysis of Communication for Asynchronous Concurrent Programming Languages.* In SAS'95, LNCS 983, pages 225–242. 1995.

[Kob97] N. Kobayashi. *A Partially Deadlock-Free Typed Process Calculus.* To appear in LICS'97. Computer Society Press, 1997.

[KPT96] N. Kobayashi, B. C. Pierce, and D. N. Turner. *Linearity and the Pi-Calculus.* In POPL'96, pages 358–371, 1996.

[Kui92] W. Kuich, editor. *Nineteenth Colloquium on Automata, Languages and Programming (ICALP)*, LNCS 623. Springer, 1992.

[LW95] X. Liu and D. Walker. *Confluence of Processes and Systems of Objects.* In TAPSOFT'95, LNCS 915, pages 217–231. Springer, 1995.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[Mil93] R. Milner. *The Polyadic π-Calculus: A Tutorial.* In Proceedings of Logic and Algebra of Specification, International NATO Summer School (Marktoberdorf, 1991). Springer, 1993.

[Mon93] U. Montanari. *True Concurrency: Theory and Practice.* In Mathematics of Program Construction, pages 14–17. Springer, 1993. Abstract of Invited Lecture. LNCS 669.

[MPW92] R. Milner, J. Parrow, and D. Walker. *A Calculus of Mobile Processes, Part I/II.* Information and Computation, 100:1–77, 1992.

[MS92] R. Milner and D. Sangiorgi. *Barbed Bisimulation.* In Kuich [Kui92], pages 685–695.

[Nes96] U. Nestmann. *On Determinacy and Nondeterminacy in Concurrent Programming.* PhD thesis, Technische Fakultät, Universität Erlangen, 1996. Arbeitsbericht 29(14) des IMMD.

[Ode95] M. Odersky. *Polarized Name Passing.* In FST&TCS'95, LNCS 1026. Springer, 1995.

[Ora94] F. Orava. *On the Formal Analysis of Telecommunication Protocols.* PhD thesis, Department of Computer Science, Uppsala University, Sweden, 1994.

[Pri96] C. Priami. *Enhanced Operational Semantics for Concurrency.* PhD thesis, Dipartimento di Informatica, Università di Pisa-Genova-Udine, 1996. Available as PhD number TD-08/96.

[PS96] B. C. Pierce and D. Sangiorgi. *Typing and Subtyping for Mobile Processes.* Mathematical Structures in Computer Science, 6(5):409–454, 1996.

[PS97] B. C. Pierce and D. Sangiorgi. *Behavioral Equivalence in the Polymorphic Pi-Calculus.* In POPL'97, pages 241–255, 1997.

[PW95] A. Philippou and D. Walker. *On Sharing and Determinacy in Concurrent Systems.* In CONCUR'95 (Philadelphia), LNCS 962, pages 456–470. Springer, 1995.

[San96a] D. Sangiorgi. *A Theory of Bisimulation for the π-calculus.* Acta Informatica, 33:69–97, 1996.

[San96b] D. Sangiorgi. *The Name Discipline of Uniform Receptiveness.* Technical report, INRIA, Sophia-Antipolis, 1996. Submitted for publication.

[THK94] K. Takeuchi, K. Honda, and M. Kubo. *An Interaction-Based Language and its Typing Sustem.* In PARLE'94, LNCS 817, pages 398–413. Springer, 1994.

[Tof91] C. Tofts. *Proof Methods and Pragmatics for Parallel Programming.* PhD thesis, LFCS, University of Edinburgh, 1991. Available as ECS-LFCS-91-140.

[Yos96] N. Yoshida. *Graph Types for Monadic Mobile Processes.* In FST&TCS'96, LNCS 1180, pages 371–386. Springer, 1996.