Linearity and the Pi-Calculus

NAOKI KOBAYASHI
University of Tokyo
BENJAMIN C. PIERCE
University of Pennsylvania
and
DAVID N. TURNER
An Teallach Limited

The economy and flexibility of the pi-calculus make it an attractive object of theoretical study and a clean basis for concurrent language design and implementation. However, such generality has a cost: encoding higher-level features like functional computation in pi-calculus throws away potentially useful information. We show how a linear type system can be used to recover important static information about a process's behavior. In particular, we can guarantee that two processes communicating over a linear channel cannot interfere with other communicating processes. After developing standard results such as soundness of typing, we focus on equivalences, adapting the standard notion of barbed bisimulation to the linear setting and showing how reductions on linear channels induce a useful "partial confluence" of process behaviors. For an extended example of the theory, we prove the validity of a tail-call optimization for higher-order functions represented as processes.

Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory

General Terms: Languages, Theory

Additional Key Words and Phrases: Concurrency, confluence, linear types, pi-calculus, process calculi

This is a revised and extended version of a paper presented at 23rd ACM SIGPAN-SIGACT Symposium on Principles of Programming Languages (POPL96), pp.358–371, 1996.
Author's addresses: N. Kobayashi, Department of Information Science, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan; email: koba@is.s.u-tokyo.ac.jp; B. C. Pierce, Department of Computer and Information Science, University of Pennsylvania, 200 South 33rd Street, Philadelphia, PA 19104-6389, USA; email: bcpierce@cis.upenn.edu; D. N. Turner, An Teallach Limited, Technology Transfer Center, King's Buildings, Edinburgh, EH9 3JL, UK, email: dnt@cruach.freeserve.co.uk.
Kobayashi was partially supported by Grant-in-Aid for Scientific Research of Japan numbers 06452389 and 07780232. Pierce was supported by EPSRC grant GR/K 38403. Turner was supported by EPSRC grant GR/J 527099 ("Save space with linear types").

1. INTRODUCTION

A long line of formal systems, from Hewitt's Actors [Agha 1986; Hewitt 1977] to modern process calculi like Milner, Parrow, and Walker's pi-calculus [Milner 1991; Milner et al. 1992], have popularized the idea that a range of concurrent programming idioms can be modeled by simple processes exchanging messages on channels. Besides their immediate applications in specification and verification of concurrent systems, these calculi have been used as the basis for concurrency features in numerous programming languages (e.g., [Cardelli 1986; Giacalone et al. 1989; Reppy 1991]), as foundations for theoretical study of language features like concurrent objects (e.g., [Honda and Tokoro 1991; Jones 1993; Kobayashi and Yonezawa 1994; 1995b; Vasconcelos 1994; Walker 1995]), and more recently as core programming languages in their own right [Fournet and Gonthier 1996; Fournet et al. 1996; Pierce and Turner 1995; 1997].

The small set of message-passing primitives used by the pi-calculus makes it attractive both as an object of theoretical study and as a basis for language design and implementation. However, such generality has a cost: the pi-calculus is an extremely low-level notation, and encoding higher-level features (such as functional computation) in pi-calculus can throw away potentially useful information. A common response to this observation is to consider larger calculi with, for example, functions as primitives. This ensures, at some cost in parsimony, that functions can be compiled efficiently and that the expected high-level rules for reasoning (either formally or informally) actually apply. A more radical approach—the one we are exploring here—is to avoid adding new primitives and instead recover information about special "modes of usage" from static type information.

1.1 Encoding Functional Computation

The following example illustrates how one can encode functions and function application (cf. [Milner 1990; Sangiorgi 1993]) in a pure message-passing calculus and also highlights some of the pitfalls of such encodings.

Suppose that *plusone* is a communication channel and that there is some running process that repeatedly reads pairs $[i, r]$ from *plusone*, with i a number and r a channel, and responds in each case by sending the value $i + 1$ on r. (Numbers themselves can be implemented using processes and channels, but that need not concern us here.) We may speak of *plusone* as the "location" of this incrementing process, since the channel *plusone* is the only means by which other processes can refer to it. To build another function using this one, suppose we are given a channel *plustwo* to serve as the location for the new process. We create a process that repeatedly reads pairs $[j, s]$ from *plustwo* and uses *plusone* twice, returning the final result, $j + 2$, along the channel s:

$$\mathit{plustwo}?^*[j,s].\,(\nu\ r_1, r_2)\,(\ \mathit{plusone}![j,r_1]\ \mid\ r_1?[k].\,\mathit{plusone}![k,r_2]\ \mid\ r_2?[l].\,s![l]\ )$$

The replicated input expression $\mathit{plustwo}?^*[j,s].\ \ldots$ describes a process that, each time it receives a pair $[j, s]$ on the channel *plustwo*, starts a fresh copy of its body running in parallel with all the other processes in the system. Each copy of the body (i.e., each invocation of the function) creates two fresh channels, r_1 and r_2, using the channel-creation operator ν. Then it sends the message $[j, r_1]$ on the channel *plusone* (written $\mathit{plusone}![j, r_1]$) and, in parallel, waits to receive a response k on the "continuation channel" r_1 (written $r_1?[k]. \dots$). When this arrives, it sends the message $[k, r_2]$ on *plusone* and waits for the response l along r_2. Finally, it sends l back along the continuation channel s to the original caller.

In this example, we used the convention that a function is a process reading messages from some channel f, while callers of the function are processes that send messages on f. Unfortunately, there is nothing in the pi-calculus notation that prevents another process from disrupting this protocol, either maliciously or accidentally, by *reading* from f instead of writing to it. The asymmetry of roles between function and caller has been lost in the translation. This lack of structure can lead to errors in more complex programs and blocks many useful optimizations and program transformations.

1.2 Arities and Polarities

The simplest type systems for the pi-calculus [Milner 1991] just track the arities of channels, in order to prevent situations where the tuple of arguments provided by a sender is not the same size as the tuple of bound variables in the receiver, as in $x![y, z] \mid x?[a, b, c]. P$. Each channel x is assigned a type of the form $\updownarrow[T_1, \dots, T_n]$, read "channel carrying $T_1, \dots, T_n$," describing the number of values in each message sent along x, as well as their types. For example, the channel *plustwo* would be given the type $\updownarrow[\mathit{Num}, \updownarrow[\mathit{Num}]]$, since it is used to communicate pairs where the first element is a number and the second is a channel that is used to send a single number.

This type system can be extended in many ways, for instance by adding polymorphism [Gay 1993; Liu and Walker 1995b; Turner 1995; Vasconcelos and Honda 1993]. Of more interest here, however, is the possibility of *refining* it so that the types of channels carry more information about how they are used. For example, we can straightforwardly capture the *polarity* (or directionality) of communications on channels like *plustwo* by introducing two new channel types: $![T_1, \dots, T_n]$ for channels that (in a given context) can only be used to send tuples of type $[T_1, \dots, T_n]$, and $?[T_1, \dots, T_n]$ for channels that can only be used to receive $[T_1, \dots, T_n]$-tuples [Odersky 1995; Pierce and Sangiorgi 1993]. From the point of view of the process implementing the increment function, the type of *plusone* now becomes $?[\mathit{Num}, ![\mathit{Num}]]$—that is, it can only be used to read messages, where each message must consist of a number and a channel that can be used to send a response. From the point of view of the users of this function, *plusone* has type $![\mathit{Num}, ![\mathit{Num}]]$: it can only be used to send messages, and each message must again consist of a number and a channel that can be used for sending numbers.

This refinement is not only useful for preventing programming mistakes; it also yields more powerful techniques for reasoning about programs. For example, in an early draft of [Milner 1990], Milner had proposed two encodings of the call-by-value lambda-calculus into the pi-calculus. Unfortunately, the simpler of the two encodings turned out not to preserve beta-equivalence, because of the possibility that the external environment might use channels generated by the encoding in the wrong direction [Sangiorgi 1992]. Pierce and Sangiorgi [1993] showed how the validity of beta-reduction could be established by typing the encoding using input-only and output-only channels. In essence, refining the type system reduces the number of contexts in which a given process can correctly be placed, thus making it easier for two processes to be equivalent.

However, some useful information cannot be captured by simple polarized channel types. For example, we cannot express the fact that the result channel that is passed to *plusone* is used just once. Without this information, we are not justified in replacing our *plustwo* example with a more efficient "tail-call-optimized" form:

$$\mathit{plustwo}?^{*}[j,s].\,(\nu\ r)\,(\ \mathit{plusone}![j,r]\ \mid\ r?[k].\,\mathit{plusone}![k,s]\ )$$

To see that this version does not have the same behavior as the original in all contexts, suppose that the increment process actually signals twice on its result channel. The tail-call-optimized implementation above makes this fact visible to its own caller, generating two messages on the result channel s, whereas the original implementation sends just one message on s.

1.3 Linear Types

We propose here a more refined pi-calculus type system, where channel types have not only *polarities*, which determine the directions in which they can be used, but also *multiplicities*, which control how many times they can be used. For example, the type $!^{1}[\mathit{Num}]$ describes a channel that can be used exactly once to send a message containing a number, while $!^{\omega}[\mathit{Num}]$ describes a channel that can be used any number of times. In a pi-calculus with linear channel types, we can take the type of *plusone* and *plustwo*, from the point of view of callers, to be $!^{\omega}[\mathit{Num}, !^{1}[\mathit{Num}]]$, capturing the fact that each of them can be invoked any number of times but that the result channel in each invocation is only used once. Under this more accurate typing, the original and tail-call-optimized versions of our *plustwo* example are provably equivalent in the sense that they behave the same in all well-typed contexts, as we show in Section 6. Actually, the refined typings of *plusone* and *plustwo* are instances of a general observation: using techniques similar to those in Pierce and Sangiorgi [1993] and Turner [1995], it is easy to show that *every* result channel used in the pi-calculus encoding of a lambda-term can be assigned a linear type.

Not only are some useful equivalences gained in the presence of linear types; even the mechanics of reasoning about processes can be streamlined, since (in the absence of nonlocal operators like choice) two processes communicating over a linear channel can neither interfere with nor be affected by other communicating processes. This stands in sharp contrast to the usual situation in the pi-calculus, where the possibility that two senders can be in competition for one receiver or vice versa introduces an essential element of nondeterminism. The expression $x![y] \mid x![z] \mid x?[a].\,a![\,]$ can reduce either to $x![z] \mid y![\,]$ or to $x![y] \mid z![\,]$, and these cannot be reduced further so as to reach a common point. On the other hand, there can never be two senders or two receivers on a linear channel, and so such race conditions can never arise. Indeed, linear communications exhibit a *partial confluence* property reminiscent of the Church-Rosser property of the lambda-calculus, as we show in Section 4.4.

Linear typing enables a variety of compiler optimizations. For example, the implementation of communication on a linear channel can be simplified, since a linear channel can only be in one of two states when a communication occurs: there may be either zero or one processes of each polarity waiting to communicate, but never more than one and never another one of the same polarity. Section 7 describes in more detail how linear types can be utilized in the compilation of pi-calculus.

Linear channel types can also help detect common programming mistakes. For example, it is often the case that a process must signal on some channel to indicate that it is leaving a "critical section." Suppose c is the channel upon which we are supposed to signal completion. If the critical section is complex, it is easy to forget to signal completion on c, or to signal twice on c. However, if c is given a linear type, our typing rules will ensure that c is used exactly once.

Note, though, that we must be careful about precisely what "exactly once" means here: typing is preserved by reduction (cf. Section 4.3), so a process containing exactly one use of c must either communicate on c and evolve to a process that cannot use c, or else make some other step and become a process that still contains one use of c, or else do nothing at all. This guarantees that c will never be used twice, but not that c will eventually be used, since the subprocess in which c occurs might diverge or deadlock before it actually communicates on c. For example, the process $(\nu z : \updownarrow^{\omega}[\,])\, z?[\,].\, c![\,]$ creates a fresh channel z, reads from it, and then sends on c. Since z is fresh, there can never be any sender on z, and the send on c will never be reached. This possibility of *partial* deadlock or divergence in the pi-calculus gives linear typing a slightly different force than in lambda-calculus, where every linear value *will* be used unless the whole program fails to yield any result.

We do not discuss type inference issues in this article (Section 8 mentions some related work on this topic).

1.4 Overview

Our goals in this article are (1) a precise definition of a pi-calculus with linear types and (2) formal justifications of the claims sketched here about typing and program equivalences. The novel aspects of this development are the technical treatment of linear channel types (which, since they have both input and output ends, actually comprise *two* linear capabilities) and, more importantly, the study of behavioral equivalences in the presence of linear types.

After introducing some notational conventions in Section 2, we define the typing relation (Section 3) and show that it is preserved by reduction (Section 4); along the way, we remark on properties such as confluence. In Section 5, we adapt the standard notion of *barbed congruence* to the present setting. Section 6 uses barbed bisimulation to prove the two versions of *plustwo* equivalent. Section 7 discusses applications of the system studied here, and Section 8 discusses related work. Throughout the article, sections marked "Technical properties" develop results needed in subsequent proofs but of limited interest in themselves; these may be skipped on a first reading.

2. NOTATIONAL PRELIMINARIES

We describe our linear type system in terms of a fragment of the polyadic pi-calculus [Milner 1991], omitting the choice and matching operators (as in Milner's mini pi-calculus [Milner 1990]) and using only asynchronous communication (as in Honda and Tokoro's ν-calculus [Honda and Tokoro 1991]). For simplicity, we use a replicated input operator (the full pi-calculus replication operator poses no problems for the linear type system, but complicates the operational semantics and proofs of behavioral properties). We add booleans and conditional expressions as primitives, since their typing behavior interacts with linearity in an interesting way.

Our type system and reduction semantics can easily be extended to the full pi-calculus, though some of the results we developed must be treated carefully. For example, in the presence of output guards, Example 6.4 is not valid. However, if we only allow output guards on nonlinear channels, the example does remain valid. Similarly, if we add a choice operator to our calculus, Theorems 4.4.3 and 5.11 do not hold, but we can recover them by restricting choice to nonlinear channels.

2.1 Syntax

Definition 2.1.1. The metavariable m ranges over the *multiplicities* 1 and ω. The metavariable p ranges over *polarities*. Formally, these are subsets of the set of capabilities $\{i, o\}$, but for readability we use the abbreviations $\updownarrow = \{i, o\}$, $! = \{o\}$, $? = \{i\}$, and $\emptyset = \{\}$. The polarities $\emptyset$ and $\updownarrow$ are called *even*.

Definition 2.1.2. The set of types is

$$\begin{array}{lll} T & ::= & p^m[\tilde{T}] & \text{channel types} \\ & & \mathit{Bool} & \text{booleans.} \end{array}$$

For example, the type $!^1[\mathit{Bool}]$ describes a channel that can be used exactly once to send a message containing a boolean, while $\updownarrow^1[\mathit{Bool}]$ describes a channel that can be used once for input and once for output. For brevity, sequences of types, etc., are written $\tilde{T}$ instead of $T_1, \ldots, T_n$. A type of the form $p^1[\tilde{T}]$, where $p \neq \emptyset$, is called *limited*. All other types are *unlimited*. An example of an unlimited type is $!^\omega[\mathit{Bool}, !^1[\mathit{Bool}]]$, which describes a channel upon which we can send any number of messages, where the components of each message must be a boolean and a linear output channel.

Definition 2.1.3. The set of process expressions is

$$\begin{array}{lll} P & ::= & P \mid Q & \text{parallel composition} \\ & & x![\tilde{y}] & \text{output atom} \\ & & x?[\tilde{y}].\, P & \text{input prefix} \\ & & x?^*[\tilde{y}].\, P & \text{replicated input prefix} \\ & & (\nu x : T)\, P & \text{channel creation} \\ & & \text{if } x \text{ then } P \text{ else } Q & \text{conditional.} \end{array}$$

To avoid writing too many parentheses, we give ν a higher precedence than $|$ and make the bodies of inputs extend as far to the right as possible, so that $x?^*[\,].\,(\nu y : \updownarrow^\omega[\,])\, x![\,] \mid y![\,]$ means $x?^*[\,].\,(((\nu y : \updownarrow^\omega[\,])\, x![\,]) \mid y![\,])$. We write $(\nu \tilde{x}{:}\tilde{T})\, P$ to abbreviate $(\nu x_1 : T_1) \ldots (\nu x_n : T_n)\, P$. The "inert process" $\mathbf{0}$ is defined as $\mathbf{0} = (\nu x : \updownarrow^\omega[\,])\, x![\,]$.

The free and bound variables of a process expression are defined in the usual way, with x having scope P in the process $(\nu x : T)\, P$ and $\tilde{y}$ having scope P in the processes $x?[\tilde{y}].\, P$ and $x?^*[\tilde{y}].\, P$.

2.2 Type Environments

Definition 2.2.1. A *type environment* Γ is a set of bindings of distinct variables to types, containing at least the bindings *true* : *Bool* and *false* : *Bool*. By convention, we write type environments as $x_1 : T_1, \ldots , x_n : T_n$ or just $\tilde{x}{:}\tilde{T}$, omitting the bindings of the booleans. The domain of a type environment, written $dom(\Gamma)$, is the set of variables it binds. Type environments are extended with bindings for new variables by writing $\Gamma, x : T$ or $\Gamma, \tilde{x}{:}\tilde{T}$. We write $\Gamma(x) = T$ to mean that $\Gamma = \Gamma_1, x : T$ for some Γ_1. A type environment is unlimited if each of its bindings is.

We always consider a process relative to a particular type environment, whose domain must contain all the free variables of the process. We regard names of bound variables as inessential and perform silent alpha-conversion as necessary to ensure that the name of a bound variable is always different from the names of free variables and other bound variables.

2.3 Operations on Types

Definition 2.3.1. The *combination* of two types T_1 and T_2, written $T_1 + T_2$, is defined as follows (in all cases other than those listed below, $T_1 + T_2$ is undefined):

$$\begin{array}{rcl} Bool + Bool & = & Bool \\ p^{\omega}[\tilde{T}] + q^{\omega}[\tilde{T}] & = & (p \cup q)^{\omega}[\tilde{T}] \\ p^{1}[\tilde{T}] + q^{1}[\tilde{T}] & = & (p \cup q)^{1}[\tilde{T}] \ \text{ if } p \cap q = \emptyset \end{array}$$

(This operation is a particular case of the "composition of types" studied by Honda [1996]. See Section 8 for further comparisons.) The + operator is extended pointwise to type environments:

$$\begin{array}{rcl} dom(\Gamma_1 + \Gamma_2) & = & dom(\Gamma_1) \cup dom(\Gamma_2) \\ (\Gamma_1 + \Gamma_2)(x) & = & \begin{cases} \Gamma_1(x) + \Gamma_2(x) & \text{if } x \in dom(\Gamma_1) \cap dom(\Gamma_2) \\ \Gamma_1(x) & \text{if } x \in dom(\Gamma_1) \text{ and } x \notin dom(\Gamma_2) \\ \Gamma_2(x) & \text{if } x \in dom(\Gamma_2) \text{ and } x \notin dom(\Gamma_1) \end{cases} \end{array}$$

We write $\Gamma + \tilde{x}{:}\tilde{T}$ as an abbreviation for $\Gamma + x_1 : T_1 + \cdots + x_n : T_n$.

3. TYPING

Our linear channel types are inspired by Girard's linear logic [Girard 1987; Girard et al. 1989], a "resource-conscious" refinement of classical logic, and by numerous proposals for functional languages based on linear logic (e.g., [Abramsky 1993; Baker 1992; Mackie 1994; Turner et al. 1995; Wadler 1991]). A crucial element in these systems is the careful treatment of the typing environments under which expressions are judged to be well typed. For example, to form a pair of two values v and w under some type environment Γ in a linear lambda-calculus, we do not simply check that v and w are well typed in Γ

$$\frac{\Gamma \vdash v \in V \qquad \Gamma \vdash w \in W}{\Gamma \vdash \langle v, w \rangle \in V \times W}$$

as in an ordinary typed lambda-calculus. Instead, we split the type environment in the conclusion into two parts, representing the resources consumed by v and w separately:

$$\frac{\Gamma_1 \vdash v \in V \qquad \Gamma_2 \vdash w \in W}{\Gamma_1 + \Gamma_2 \vdash \langle v, w \rangle \in V \times W}$$

Similarly, in a linear type system for the pi-calculus, we must split the type environment in the rule for parallel composition,

$$\frac{\Gamma_1 \vdash P \qquad \Gamma_2 \vdash Q}{\Gamma_1 + \Gamma_2 \vdash P \,|\, Q} \tag{E-Par}$$

so that typing $P \,|\, Q$ consumes the sum of the resources used in the proofs that P and Q are well typed. We only consider instances of this rule scheme where $\Gamma_1 + \Gamma_2$ is well defined. The typing judgment $\Gamma \vdash P$ thus acquires additional force: "P is well typed under the typing assumptions in Γ and, moreover, uses each linear capability in Γ just once."

To ensure that all the linear bindings in the environment get used somewhere, we check at all the leaves of the typing derivation—i.e., at all instances of the typing rule for output expressions—that all the remaining bindings in the current environment have either multiplicity ω or polarity $\emptyset$:

$$\frac{\Gamma \ \ \text{unlimited}}{\Gamma + x : !^m[\tilde{T}] + \tilde{y}{:}\tilde{T} \vdash x![\tilde{y}]} \tag{E-Out}$$

That is, if all the bindings in Γ are unlimited, then $x![\tilde{y}]$ is well typed in Γ augmented (in case x or any of the $\tilde{y}$ are linear) with the capabilities consumed by the output itself. Note that the capabilities expressed by the types $\tilde{T}$ are consumed by the output, since the arguments $\tilde{y}$ are being passed to the receiver of the communication.

Conversely, an input expression is well typed if its body is well typed in an environment where the capability to input from x has been removed (if it is linear) and the capabilities associated with the values $\tilde{z}$ to be read from x have been added.

$$\frac{\Gamma, \tilde{z}{:}\tilde{T} \vdash P}{\Gamma + x : ?^m[\tilde{T}] \vdash x?[\tilde{z}].\, P} \tag{E-In}$$

The rule for replicated input is nearly the same, except that the type of x must be unlimited and, since the body may be instantiated many times, we must ensure that the common part of the type environment does not contain any linear bindings.

$$\frac{\Gamma, \tilde{z}{:}\tilde{T} \vdash P \qquad \Gamma \ \text{unlimited}}{\Gamma + x : ?^\omega[\tilde{T}] \vdash x?^*[\tilde{z}].\, P} \tag{E-RIn}$$

The rule for ν-expressions adds the channel type $p^m[\tilde{T}]$ for x to the type environment and checks that the process P is well formed in the resulting type environment.

$$\frac{\Gamma, x : p^m[\tilde{T}] \vdash P}{\Gamma \vdash (\nu x : p^m[\tilde{T}])\, P} \tag{E-New}$$

The rule for checking conditional expressions is a little different from the others (it corresponds to one of the additive operators of linear logic, while the rest of our operators belong to the multiplicative part). We know that the "then" and "else" parts of an if-clause will never both be executed, so any linear bindings in the environment should be used up by both parts, instead of being allocated to one part or the other.

$$\frac{\Gamma \vdash P_1 \qquad \Gamma \vdash P_2}{\Gamma + b : \mathit{Bool} \vdash \text{if } b \text{ then } P_1 \text{ else } P_2} \tag{E-If}$$

Like some linear type systems, we omit the linear logic rule of *dereliction*, which allows an unlimited channel to be coerced to a linear one. If our type system had a subtyping relation, we would similarly ban the coercion $p^w[\tilde{T}] \leq p^1[\tilde{T}]$. The worlds of linear and unlimited values are never allowed to mix.

Although we are not concerned with algorithmic issues in this article, it is worth noting that these rules give rise to a typechecking algorithm in a fairly straightforward way. Instead of "guessing" how to split the capabilities in the environment in rules like E-Par, we pass the whole environment to the lefthand premise, crossing off capabilities as they are used up and passing whatever is left to the right-hand premise. There is some cost associated with checking the "Γ unlimited" side conditions, but a few simple tricks keep this small in practice.

Intuitively, a Γ-Δ substitution is a substitution that maps a process with free variables in $dom(\Gamma)$ to a process with free variables in $dom(\Delta)$, being careful not to map two linear variables with the same polarity into the same variable and not to leave any linear capabilities in Γ unaccounted for. For example, the function that maps x to a and both y and z to b, written $[a/x, b/y, b/z]$, is a $(x : \updownarrow^1[\tilde{T}], y : ?^1[\tilde{T}], z : !^1[\tilde{T}])$-$(a : \updownarrow^1[\tilde{T}], b : \updownarrow^1[\tilde{T}])$ substitution, but $[a/x, a/y, b/z]$ is not.

Definition 3.1. A Γ-Δ-*substitution* σ is a function from $dom(\Gamma)$ to $dom(\Delta)$ such that, for each $y \in dom(\Delta)$, the type $\Delta(y)$ can be written as $(\Sigma_{(x \in dom(\Gamma) \wedge \sigma(x) = y)} \Gamma(x)) + U$ for some unlimited type U. We write $\sigma[\tilde{y}/\tilde{z}]$ for the extension of σ with $[\tilde{y}/\tilde{z}]$.

Technical Properties

Lemma 3.2 (Weakening). *If $\Gamma \vdash P$, T is unlimited, and $\Gamma + x : T$ is well defined then $\Gamma + x : T \vdash P$.*

Proof. A straightforward induction on typing derivations. □

Lemma 3.3 (Strengthening). *If $\Gamma, x : T \vdash P$ and x is not free in P then T is unlimited and $\Gamma \vdash P$.*

Proof. A straightforward induction on typing derivations. In the base case, the E-Out rule ensures that any variable not free in P has an unlimited type and that the bindings on such variables can be removed. The remaining cases can be proved by a simple inductive argument. □

Lemma 3.4. *If σ is a $(\Delta_1 + \Delta_2)$-Γ substitution, then there are Γ_1, Γ_2 such that $\Gamma_1 + \Gamma_2 = \Gamma$ and σ is both a Δ_1-Γ_1 and a Δ_2-Γ_2 substitution.*

Proof. By the definition of Δ-Γ substitution, for each $y \in dom(\Gamma)$ we have

$$\Gamma(y) = (\Sigma_{x \in dom(\Delta_1 + \Delta_2) \wedge \sigma(x) = y} \, (\Delta_1 + \Delta_2)(x)) + U$$

$$= (\Sigma_{x\in dom(\Delta_1)\wedge\sigma(x)=y}\ \Delta_1(x)) + (\Sigma_{x\in dom(\Delta_2)\wedge\sigma(x)=y}\ \Delta_2(x)) + U$$
$$= ((\Sigma_{x\in dom(\Delta_1)\wedge\sigma(x)=y}\ \Delta_1(x)) + U) + ((\Sigma_{x\in dom(\Delta_2)\wedge\sigma(x)=y}\ \Delta_2(x)) + U).$$

Let $\Gamma_1(y) = (\Sigma_{x\in dom(\Delta_1)\wedge\sigma(x)=y}\Delta_1(x)) + U$ and $\Gamma_2(y) = (\Sigma_{x\in dom(\Delta_2)\wedge\sigma(x)=y}\Delta_2(x)) + U$. Then σ is both a Δ_1-Γ_1 and a Δ_2-Γ_2 substitution. □

LEMMA 3.5. *If σ is a $(\Delta + x : T)$-Γ substitution and $\sigma(x) = y$ then there exists a Λ such that $\Gamma = \Lambda + y : T$ and σ is a Δ-Λ substitution.*

PROOF. We have, by definition, that for each $z \in dom(\Gamma)$,

$$\Gamma(z) = \begin{cases} (\Sigma_{u\in dom(\Delta)\wedge\sigma(u)=z}\ \Delta(u)) + U_z & \text{if } z \neq y \\ (\Sigma_{u\in dom(\Delta)\wedge\sigma(u)=z}\ \Delta(u)) + T + U_z & \text{if } z = y. \end{cases}$$

So we can easily construct a suitable Λ. □

LEMMA 3.6 (SUBSTITUTION PRESERVES TYPING). *If $\Delta \vdash P$ and σ is a Δ-Γ substitution, then $\Gamma \vdash \sigma P$.*

PROOF. A straightforward induction on typing derivations. The parallel composition case uses Lemma 3.4. The input and output cases use Lemma 3.5. □

4. OPERATIONAL SEMANTICS

Following Milner [1991], the operational semantics of processes is presented in two steps. First, we define a *structural congruence* relation $P \equiv Q$, capturing the fact that, for example, the order of the branches in a parallel composition has no effect on its behavior. Then we define a *reduction* relation $P \longrightarrow Q$, specifying how processes evolve through communications between subprocesses.

4.1 Structural Congruence

Structural congruence plays an important technical role in simplifying the definition of the reduction relation. For example, we intend that both the process $x![\,] \,|\, (x?[\,].\,P)$ and the process $(x?[\,].\,P) \,|\, x![\,]$ reduce to P. Since these two will be structurally congruent, it suffices to write the reduction rule only for the first case and to stipulate, in general, that if P contains some possibility of communication, then so does any expression structurally congruent to P.

The first two structural congruence rules state that parallel composition is commutative and associative.

$$P \,|\, Q \equiv Q \,|\, P \qquad \text{(S-COMMUT)}$$

$$(P \,|\, Q) \,|\, R \equiv P \,|\, (Q \,|\, R) \qquad \text{(S-ASSOC)}$$

The third rule is called *scope extrusion* in the pi-calculus literature:

$$(\nu x : T)\, P \,|\, Q \equiv (\nu x : T)\, (P \,|\, Q) \quad x \text{ not free in } Q \qquad \text{(S-EXTR)}$$

Informally, it says that the scope of the channel x, which starts out private to the process P, can be extended to include Q. The side-condition ensures that Q does not already have a free channel named x. (This condition can always be satisfied by renaming x, if necessary, before applying the scope extrusion rule.) For example, the process $((\nu x : T)\, w![x]) \,|\, w?[z].\, z![]$ may be transformed to $(\nu x : T)\,(w![x] \,|\, w?[z].\, z![])$. Finally, for technical convenience, we allow adjacent channel bindings to be switched:

$$(\nu x : T)\,(\nu y : U)\,P \equiv (\nu y : U)\,(\nu x : T)\,P \quad x \neq y \qquad \text{(S-SWITCH)}$$

Formally, the relation $\equiv$ is the least congruence closed under these four rules, meaning that $P \equiv Q$ if Q can be obtained by applying the rules any number of times, in either direction, to arbitrary subexpressions of P.

Technical Properties

LEMMA 4.1.1. *If $P \equiv Q$ then $\Gamma \vdash P$ iff $\Gamma \vdash Q$.*

PROOF. By induction on the proof of $P \equiv Q$. The base cases consist of the four rules for structural congruence given above, plus reflexivity. The induction steps arise from the definition of structural congruence as a congruence relation; they include rules allowing structural congruence to be applied to each subterm of each process constructor, plus symmetry and transitivity. Most cases are trivial: we present only the cases for the rule S-EXTR (in both directions).

For the forward implication, suppose that $\Gamma \vdash (\nu x : T)\, P_1 \,|\, P_2$. It must be the case that there exist Γ_1 and Γ_2 such that $\Gamma_1, x : T \vdash P_1$ and $\Gamma_2 \vdash P_2$ where $\Gamma_1 + \Gamma_2 = \Gamma$. If $x \in \mathit{dom}(\Gamma_2)$, we can alpha-convert it (a special case of Lemma 3.6). Once we have that $x \notin \mathit{dom}(\Gamma_2)$ we can prove that $(\Gamma_1, x : T) + \Gamma_2$ is well defined, and we can use E-PAR to prove that $(\Gamma_1, x : T) + \Gamma_2 \vdash P_1 \,|\, P_2$, or equivalently, $\Gamma, x : T \vdash P_1 \,|\, P_2$. The result follows using the E-NEW rule.

For the reverse implication, suppose that $\Gamma \vdash (\nu x : T)\,(P_1 \,|\, P_2)$. It must be the case that there exist Γ_1 and Γ_2 such that $\Gamma_1 \vdash P_1$ and $\Gamma_2 \vdash P_2$, where $\Gamma_1 + \Gamma_2 = \Gamma, x : T$. If $x \notin \mathit{dom}(\Gamma_2)$, then the result follows using the E-NEW and E-PAR rules. Otherwise, it must be the case that $\Gamma_1 = \Phi_1, x : T_1$, and $\Gamma_2 = \Phi_2, x : T_2$, where $\Phi_1 + \Phi_2 = \Gamma$ and $T_1 + T_2 = T$. Since x is not free in P_2, Lemma 3.3 proves that T_2 must be unlimited and $\Phi_2 \vdash P_2$. Therefore, using Lemma 3.2, we also have $\Phi_1, x : T \vdash P_1$, and the result follows using the E-NEW and E-PAR rules. $\square$

Definition 4.1.2. A process is *guarded* if it is an input, output, replicated input, or conditional. A process $(\nu \tilde{x}{:}\tilde{T})\,(P_1 \,|\, \cdots \,|\, P_n)$ is in *normal form* if $P_1, \ldots, P_n$ are guarded processes.

LEMMA 4.1.3. *For any process P there is some Q in normal form such that $P \equiv Q$.*

PROOF. A simple induction on the structure of P. Most cases are trivial. In the parallel composition case we can use the S-EXTR, S-COMMUT, and S-ASSOC rules to move all ν-bindings not guarded by an input to the top level. $\square$

LEMMA 4.1.4. *If $(\nu \tilde{x}{:}\tilde{T})\,(P_1 \,|\, \cdots \,|\, P_n)$ and $(\nu \tilde{y}{:}\tilde{U})\,(Q_1 \,|\, \cdots \,|\, Q_m)$ are structurally congruent, and both are normal forms, then $m = n$ and $\tilde{x}{:}\tilde{T}$ is a permutation of $\tilde{y}{:}\tilde{U}$. Moreover, for some permutation $Q_1', \ldots, Q_n'$ of $Q_1, \ldots, Q_n$, we have $P_i \equiv Q_i'$ for $1 \leq i \leq n$.*

PROOF. The result follows from the fact that structural congruence neither introduces nor eliminates guarded subexpressions or ν-bindings, and does not move subexpressions across input prefixes. $\square$

4.2 Reduction

The pi-calculus reduction relation is usually written $P \longrightarrow Q$, meaning that P can evolve to Q by making a single, atomic step of communication. We use the same basic relation, but annotate it with some extra information that we need later for stating properties of the semantics. We write $P \xrightarrow{\alpha} Q$ where α is either the name of a channel, indicating where the communication occurred, or else a multiplicity m, indicating that the communication occurred on a local channel with multiplicity m. Another difference from the standard reduction relation is that type annotations may be changed by reduction. For example, $(\nu x : \updownarrow^{1}[\,])\,(x![\,] \mid x?[\,].\,P)$ is reduced to $(\nu x : \emptyset^{1}[\,])\,P$. The change of type annotation for x captures the fact that the linear channel x can no longer be used for communication.

The most important reduction rule is the one for communication. The parallel composition of an output and an input on x can evolve to the body of the input-process with the arguments to the output substituted for the formal parameters. We record the channel x on the arrow.

$$x![\tilde{y}] \mid x?[\tilde{z}].\,P \xrightarrow{x} [\tilde{y}/\tilde{z}]P \qquad \text{(R-COM)}$$

Similarly, an output in parallel with a replicated input can reduce to the appropriate instance of the body of the input plus a fresh copy of the replicated input itself, ready to consume further outputs on x.

$$x![\tilde{y}] \mid x?^{*}[\tilde{z}].\,P \xrightarrow{x} [\tilde{y}/\tilde{z}]P \mid x?^{*}[\tilde{z}].\,P \qquad \text{(R-RCOM)}$$

If P can reduce to Q in isolation, then it can still do so when it is placed in parallel with some other process R:

$$\frac{P \xrightarrow{\alpha} Q}{P \mid R \xrightarrow{\alpha} Q \mid R} \qquad \text{(R-PAR)}$$

Similarly, if P reduces to Q, then it can still do so when placed under a ν. However, because of our extra annotations, we must divide this rule into three cases. If the ν-bound channel is x, the channel on which the communication in P has occurred, and x is a nonlinear channel, then we change the label x on the arrow to ω.

$$\frac{P \xrightarrow{x} Q}{(\nu x : \updownarrow^{\omega}[\tilde{T}])\,P \xrightarrow{\omega} (\nu x : \updownarrow^{\omega}[\tilde{T}])\,Q} \qquad \text{(R-NEW1)}$$

If the type of x is linear, we change the label x to 1. We also change the polarity of the ν-binding to $\emptyset$, since x cannot be used for further communications.

$$\frac{P \xrightarrow{x} Q}{(\nu x : \updownarrow^{1}[\tilde{T}])\,P \xrightarrow{1} (\nu x : \emptyset^{1}[\tilde{T}])\,Q} \qquad \text{(R-NEW2)}$$

If the communication in P occurs on some channel other than x, then we simply pass α on unaltered.

$$\frac{P \xrightarrow{\alpha} Q \qquad x \neq \alpha}{(\nu x : T)\, P \xrightarrow{\alpha} (\nu x : T)\, Q} \tag{R-New3}$$

A conditional expression reduces to either its then-branch or its else-branch, depending on its guard.

$$\text{if } \textit{true} \text{ then } P \text{ else } Q \xrightarrow{1} P \tag{R-IfT}$$

$$\text{if } \textit{false} \text{ then } P \text{ else } Q \xrightarrow{1} Q \tag{R-IfF}$$

Finally, if P can be rearranged to some P' using the structural congruence laws so that a communication is possible in P', then the same communication is possible in P.

$$\frac{P \equiv P' \qquad P' \xrightarrow{\alpha} Q' \qquad Q' \equiv Q}{P \xrightarrow{\alpha} Q} \tag{R-Cong}$$

Note that we sometimes abbreviate $P \xrightarrow{\alpha} Q$ to $P \longrightarrow Q$ in contexts where the label α is irrelevant. As usual, we write $P \longrightarrow^{*} Q$ when P reduces to Q in zero or more steps.

Technical Properties

LEMMA 4.2.1. *If* $P \xrightarrow{\alpha} Q$ *then one of the following statements is true (where* $R_1, \ldots, R_n$ *are guarded processes and the sequence* $\tilde{w}$ *in 2 and 3 may contain* x*):*

(1) $P \equiv (\nu\tilde{w}{:}\tilde{W})\,(\nu x : \updownarrow^{1}[\tilde{T}])\,(R_1 \mid \cdots \mid R_n \mid x![\tilde{z}] \mid x?[\tilde{y}].\, S)$ *and*
$Q \equiv (\nu\tilde{w}{:}\tilde{W})\,(\nu x : \emptyset^{1}[\tilde{T}])\,(R_1 \mid \cdots \mid R_n \mid [\tilde{z}/\tilde{y}]S)$

(2) $P \equiv (\nu\tilde{w}{:}\tilde{T})\,(R_1 \mid \cdots \mid R_n \mid x![\tilde{z}] \mid x?[\tilde{y}].\, S)$ *and*
$Q \equiv (\nu\tilde{w}{:}\tilde{T})\,(R_1 \mid \cdots \mid R_n \mid [\tilde{z}/\tilde{y}]S)$

(3) $P \equiv (\nu\tilde{w}{:}\tilde{T})\,(R_1 \mid \cdots \mid R_n \mid x![\tilde{z}] \mid x?^{*}[\tilde{y}].\, S)$ *and*
$Q \equiv (\nu\tilde{w}{:}\tilde{T})\,(R_1 \mid \cdots \mid R_n \mid [\tilde{z}/\tilde{y}]S \mid x?^{*}[\tilde{y}].\, S)$

(4) $P \equiv (\nu\tilde{x}{:}\tilde{T})\,(R_1 \mid \cdots \mid R_n \mid \textit{if true then } S_1 \textit{ else } S_2)$ *and*
$Q \equiv (\nu\tilde{x}{:}\tilde{T})\,(R_1 \mid \cdots \mid R_n \mid S_1)$

(5) $P \equiv (\nu\tilde{x}{:}\tilde{T})\,(R_1 \mid \cdots \mid R_n \mid \textit{if false then } S_1 \textit{ else } S_2)$ *and*
$Q \equiv (\nu\tilde{x}{:}\tilde{T})\,(R_1 \mid \cdots \mid R_n \mid S_2)$

PROOF. The result follows from the fact that R-CONG and the R-NEW rules can be permuted after the other rules and successive applications of R-CONG can be combined into one R-CONG. □

4.3 Type Soundness

The *sine qua non* of an operational semantics for a typed language is that well-typedness must be preserved under reduction.

We first introduce some notation to simplify our statement of type soundness: the expression $\Gamma - \alpha$ denotes the remaining capabilities in Γ after a communication on α:

$$\Gamma - \alpha = \begin{cases} \Delta, x : \emptyset^1[\tilde{T}] & \text{if } \alpha = x \text{ and } \Gamma = \Delta, x : \updownarrow^1[\tilde{T}] \\ \Delta, x : \updownarrow^\omega[\tilde{T}] & \text{if } \alpha = x \text{ and } \Gamma = \Delta, x : \updownarrow^\omega[\tilde{T}] \\ \Gamma & \text{if } \alpha = 1 \text{ or } \alpha = \omega \end{cases}$$

Note that $\Gamma - \alpha$ is undefined if $\alpha = x$ and the type of x in Γ does not have polarity $\updownarrow$. This give some additional force to our type soundness theorem, since it checks that channels which are used for communication allow both read and write access.

THEOREM 4.3.1 (TYPE SOUNDNESS). *If $\Gamma \vdash P$ and $P \xrightarrow{\alpha} Q$ then $\Gamma - \alpha \vdash Q$.*

PROOF. We use induction on the derivation of $P \xrightarrow{\alpha} Q$.

Case $x![\tilde{y}] \,|\, x?[\tilde{z}].\, P \xrightarrow{x} [\tilde{y}/\tilde{z}]P$. It must be the case that there exist Γ_1, Γ_2, m, and $\tilde{T}$ such that Γ_1 is unlimited, such that $\Gamma = (\Gamma_1 + x : !^m[\tilde{T}] + \tilde{y} : \tilde{T}) + (\Gamma_2 + x : ?^m[\tilde{T}])$, such that $\Gamma_1 + x : !^m[\tilde{T}] + \tilde{y} : \tilde{T} \vdash x![\tilde{y}]$, and such that $\Gamma_2, \tilde{z} : \tilde{T} \vdash P$. It is easy to check that the identity substitution extended by $[\tilde{y}/\tilde{z}]$ is a $(\Gamma_2, \tilde{z} : \tilde{T}) - (\Gamma_2 + \tilde{y} : \tilde{T})$ substitution; so we can use Lemma 3.6 to prove that $\Gamma_2 + \tilde{y} : \tilde{T} \vdash [\tilde{y}/\tilde{z}]P$.

In the case where $m = \omega$, the context $\Gamma_1 + x : \updownarrow^m[\tilde{T}]$ is unlimited, and we can use Lemma 3.2 to prove that $\Gamma_1 + x : \updownarrow^m[\tilde{T}] + \Gamma_2 + \tilde{y} : \tilde{T} \vdash [\tilde{y}/\tilde{z}]P$. The result follows, since $\Gamma - x = \Gamma = \Gamma_1 + x : \updownarrow^m[\tilde{T}] + \Gamma_2 + \tilde{y} : \tilde{T}$.

In the case where $m = 1$, the context $\Gamma_1 + x : \emptyset^m[\tilde{T}]$ is unlimited, and we can use Lemma 3.2 to prove that $\Gamma_1 + x : \emptyset^m[\tilde{T}] + \Gamma_2 + \tilde{y} : \tilde{T} \vdash [\tilde{y}/\tilde{z}]P$. The result follows, since $\Gamma - x = \Gamma_1 + x : \emptyset^m[\tilde{T}] + \Gamma_2 + \tilde{y} : \tilde{T}$.

Case $x![\tilde{y}] \,|\, x?^*[\tilde{z}].\, P \xrightarrow{x} [\tilde{y}/\tilde{z}]P \,|\, x?^*[\tilde{z}].\, P$. We can use exactly the same reasoning as above to prove that $\Gamma - x \vdash [\tilde{y}/\tilde{z}]P$. In addition, we also know that Γ_2 and the type of x are both unlimited, so we can prove that $\Gamma - x = (\Gamma - x) + \Gamma_2 + x : ?^\omega[\tilde{T}]$, and we can use E-RIN and E-PAR to prove that $\Gamma - x \vdash [\tilde{y}/\tilde{z}]P \,|\, x?^*[\tilde{z}].\, P$ as required.

Case $P \,|\, R \xrightarrow{\alpha} Q \,|\, R$ where $P \xrightarrow{\alpha} Q$. It must be the case that there exist Γ_1 and Γ_2 such that $\Gamma = \Gamma_1 + \Gamma_2$, $\Gamma_1 \vdash P$ and $\Gamma_2 \vdash R$. Using induction, we have that $\Gamma_1 - \alpha \vdash Q$. In the case where $\alpha = 1$, $\alpha = \omega$, or $\alpha = x$ for some nonlinear x, we have that $\Gamma_1 - \alpha = \Gamma_1$; the result follows immediately using the E-PAR rule, since $\Gamma - \alpha = \Gamma$ and $\Gamma = \Gamma_1 + \Gamma_2 = (\Gamma_1 - \alpha) + \Gamma_2$.

In the case where $\alpha = x$ for some linear x, it must be the case that $\Gamma_1 = \Delta, x : \updownarrow^1[\tilde{T}]$ for some Δ and $\tilde{T}$ such that $\Gamma_1 - \alpha = \Delta, x : \emptyset^1[\tilde{T}]$. Since Γ_1 contains both the input and output capabilities for x and $\Gamma_1 + \Gamma_2$ is well defined, it must be the case that either $x \notin \mathit{dom}(\Gamma_2)$ or $\Gamma_2(x) = \emptyset^1[\tilde{T}]$. This implies that $\Gamma - \alpha \vdash Q \,|\, R$, since $(\Gamma_1 - \alpha) + \Gamma_2$ is well defined and equal to $(\Gamma_1 + \Gamma_2) - \alpha$.

Case $(\nu x : \updownarrow^\omega[\tilde{T}])\, P \xrightarrow{\omega} (\nu x : \updownarrow^\omega[\tilde{T}])\, Q$ where $P \xrightarrow{x} Q$. It must be the case that $\Gamma, x : \updownarrow^\omega[\tilde{T}] \vdash P$. Using induction we have that $(\Gamma, x : \updownarrow^\omega[\tilde{T}]) - x \vdash Q$. However, since the type of x is nonlinear, we have that $(\Gamma, x : \updownarrow^\omega[\tilde{T}]) - x = \Gamma, x : \updownarrow^\omega[\tilde{T}]$. The result follows immediately using the E-NEW rule, since $\Gamma - \omega = \Gamma$.

Case $(\nu x : \updownarrow^1[\tilde{T}])\, P \xrightarrow{1} (\nu x : \emptyset^1[\tilde{T}])\, Q$ where $P \xrightarrow{x} Q$. It must be the case that

$\Gamma, x : \updownarrow^1[\tilde{T}] \vdash P$. Using induction we have that $(\Gamma, x : \updownarrow^1[\tilde{T}]) - x \vdash Q$. By definition, $(\Gamma, x : \updownarrow^1[\tilde{T}]) - x = \Gamma, x : \emptyset^1[\tilde{T}]$. The result follows immediately using the E-NEW rule, since $\Gamma - 1 = \Gamma$.

Case $(\nu x : T)\, P \xrightarrow{\alpha} (\nu x : T)\, Q$ where $P \xrightarrow{\alpha} Q$ and $x \neq \alpha$. It must be the case that $\Gamma, x : T \vdash P$. Using induction we have that $(\Gamma, x : T) - \alpha \vdash Q$. Now, since $x \neq \alpha$, we have that $(\Gamma, x : T) - \alpha = (\Gamma - \alpha), x : T$. The result then follows immediately using the E-NEW rule.

Case $P \xrightarrow{\alpha} P'$ where $P \equiv Q$, $Q \xrightarrow{\alpha} Q'$ and $Q' \equiv P'$. Since structural congruence preserves typing (Lemma 4.1.1), we have that $\Gamma \vdash Q$. Using induction, we have $\Gamma - \alpha \vdash Q'$. A second use of Lemma 4.1.1 proves that $\Gamma - \alpha \vdash P'$ as required.

Case if *true* then P else $Q \xrightarrow{1} P$. It must be the case that there exists a Δ such that $\Delta \vdash P$ and $\Gamma = \Delta + \mathit{true} : \mathit{Bool}$. Since *Bool* is an unlimited type, we have using weakening (Lemma 3.2) that $\Gamma \vdash P$, where $\Gamma - 1 = \Gamma$ as required.

Case if *false* then P else $Q \xrightarrow{1} Q$. It must be the case that there exists a Δ such that $\Delta \vdash Q$ and $\Gamma = \Delta + \mathit{false} : \mathit{Bool}$. Since *Bool* is an unlimited type, we have using weakening (Lemma 3.2) that $\Gamma \vdash Q$, where $\Gamma - 1 = \Gamma$ as required. □

Finally, it is worth verifying that the type annotations and other labels in this semantics are used just for bookkeeping and do not play any role in determining the possible behaviors of well-typed process expressions. Let *Erase*(P) be the expression obtained from P by erasing all types.

THEOREM 4.3.2 (CORRESPONDENCE WITH UNTYPED SEMANTICS).

(1) *If* $P \xrightarrow{\alpha} Q$ *then* *Erase*$(P) \longrightarrow$ *Erase*(Q).

(2) *If* $\Gamma \vdash P$ *and* *Erase*$(P) \longrightarrow Q$ *then* $P \xrightarrow{\alpha} R$ *and* *Erase*$(R) = Q$, *for some* α *and* R.

PROOF. It is easy to check part (1), since our annotated semantics is a simple refinement of the untyped semantics. We prove part (2) using induction on the derivation of *Erase*$(P) \longrightarrow Q$. The only difficult case is the one for ν-bindings, which we show below:

Case $(\nu x)\, P \longrightarrow (\nu x)\, Q$ where $P \longrightarrow Q$. Using induction, we have that $P \xrightarrow{\alpha} R$ for some α and R such that *Erase*$(R) = Q$. If $\alpha \neq x$ then we can use the R-NEW3 rule to prove that $(\nu x : T)\, P \xrightarrow{\alpha} (\nu x : T)\, R$, and the result follows. If $\alpha = x$ then we need to consider the type of x in order to determine whether rules R-NEW1 or R-NEW2 apply. Since $\Gamma \vdash (\nu x : T)\, P$ it must be the case that T has the form $p^m[\tilde{T}]$.

If $m = \omega$ and $p = \updownarrow$ then we can use the R-NEW1 rule to prove that $(\nu x : \updownarrow^\omega[\tilde{T}])\, P \xrightarrow{\omega} (\nu x : \updownarrow^\omega[\tilde{T}])\, R$, and the result follows. If $m = 1$ and $p = \updownarrow$ then we can use the R-NEW2 rule to prove that $(\nu x : \updownarrow^1[\tilde{T}])\, P \xrightarrow{1} (\nu x : \emptyset^1[\tilde{T}])\, R$, and the result follows.

All other values of p can never occur. If they did then we could use type soundness (Theorem 4.3.1) to prove that $(\Gamma, x : p^m[\tilde{T}]) - x \vdash R$, which is impossible, since the binding for x must have polarity $\updownarrow$ for $(\Gamma, x : p^m[\tilde{T}]) - x$ to be well defined. □

4.4 Partial Confluence

We now come to the first results that show how linearity affects reasoning about program behavior. Our first theorem says that communication on a linear channel is deterministic:

THEOREM 4.4.1. *If* $\Gamma, x : \updownarrow^1[\tilde{T}] \vdash P$, $P \xrightarrow{x} Q$ *and* $P \xrightarrow{x} R$ *then* $Q \equiv R$.

PROOF. Using Lemma 4.2.1 twice we have that

$$\begin{aligned} P &\equiv (\nu\tilde{x}{:}\tilde{T})\,(R_1 \mid \cdots \mid R_n \mid x![\tilde{y}] \mid x?[\tilde{z}].\,S) \\ Q &\equiv (\nu\tilde{x}{:}\tilde{T})\,(R_1 \mid \cdots \mid R_n \mid [\tilde{y}/\tilde{z}]S) \end{aligned}$$

and

$$\begin{aligned} P &\equiv (\nu\tilde{x}'{:}\tilde{T}')\,(R_1' \mid \cdots \mid R_n' \mid x![\tilde{y}'] \mid x?[\tilde{z}'].\,S') \\ R &\equiv (\nu\tilde{x}'{:}\tilde{T}')\,(R_1' \mid \cdots \mid R_n' \mid [\tilde{y}'/\tilde{z}']S'). \end{aligned}$$

The type of x is linear, so $x![\tilde{y}]$ and $x?[\tilde{z}].\,S$ are the only processes in P which can use x for input or output. Thus, since normal forms are unique up to structural congruence (Lemma 4.1.4), we have that $x![\tilde{y}] \equiv x![\tilde{y}']$, $x?[\tilde{z}].\,S \equiv x?[\tilde{z}'].\,S'$, and the result follows. ☐

Our second theorem says: If P is a process that contains two possible communications, one of which is on a linear channel, then they may be performed in either order to reach the same result. In other words, linear reduction is *confluent* with respect to all other reductions. Partial confluence is particularly useful for simplifying the exploration of a process' state space. In fact, we will later show that reduction on a local linear channel preserves the congruence class of a process, enabling us to focus on reduction of unlimited channels in order to enumerate the state space of a process. (A similar property was observed by Honda and Yoshida [1994] for a syntactically restricted form of reduction on process terms called β-reduction. They went on to show, as we do in Theorem 5.11, that their restricted form of reduction preserves the observational equivalence class of a process.)

Definition 4.4.2. We let $\Gamma(\alpha)$ denote the multiplicity of α:

$$\Gamma(\alpha) = \begin{cases} 1 & \text{if } \alpha = 1 \\ \omega & \text{if } \alpha = \omega \\ m & \text{if } \alpha = x \text{ and } \Gamma = \Delta, x : p^m[\tilde{T}] \end{cases}$$

THEOREM 4.4.3 (PARTIAL CONFLUENCE). *If* $\Gamma \vdash P$, $P \xrightarrow{\alpha} Q$ *and* $\Gamma(\alpha) = 1$ *then, for any* R *and* β *such that* $P \xrightarrow{\beta} R$, *either* $R \equiv Q$ *or there is some* S *such that* $R \xrightarrow{\alpha} S$ *and* $Q \xrightarrow{\beta} S$.

PROOF. We can use Lemma 4.2.1 to analyze the two reductions $P \xrightarrow{\alpha} Q$ and $P \xrightarrow{\beta} R$. If the two communications occur on nonlocal channels x and x' respectively, then we have that

$$\begin{aligned} P &\equiv (\nu\tilde{x}{:}\tilde{T})\,(R_1 \mid \cdots \mid R_n \mid x![\tilde{y}] \mid x?[\tilde{z}].\,S) \\ Q &\equiv (\nu\tilde{x}{:}\tilde{T})\,(R_1 \mid \cdots \mid R_n \mid [\tilde{y}/\tilde{z}]S) \end{aligned}$$

and

$$
\begin{aligned}
P &\equiv (\nu\tilde{x}'{:}\tilde{T}')\,(R_1' \mid \cdots \mid R_n' \mid x'![\tilde{y}'] \mid x'?[\tilde{z}'].\,S') \\
R &\equiv (\nu\tilde{x}'{:}\tilde{T}')\,(R_1' \mid \cdots \mid R_n' \mid [\tilde{y}'/\tilde{z}']S').
\end{aligned}
$$

If $x = x'$ then we can use the same reasoning as we used in the proof of Theorem 4.4.1 to prove that $R \equiv Q$. Otherwise, since $\Gamma(\alpha) = 1$, it must be the case that x is a linear channel. It is therefore easy to check that $x![\tilde{y}] \not\equiv x'![\tilde{y}']$ and $x?[\tilde{z}].\,S \not\equiv x'?[\tilde{z}'].\,S'$. This, in addition to the fact that normal forms are unique up to structural congruence (Lemma 4.1.4), implies that $\tilde{x}{:}\tilde{T}$ is a permutation of $\tilde{x}'{:}\tilde{T}'$ and that there exist $S_1, \dots, S_n$ such that

$$
\begin{aligned}
(R_1 \mid \cdots \mid R_n) &\equiv (x'![\tilde{y}'] \mid x'?[\tilde{z}'].\,S' \mid S_1 \mid \cdots \mid S_n) \\
(R_1' \mid \cdots \mid R_n') &\equiv (x![\tilde{y}] \mid x?[\tilde{z}].\,S \mid S_1 \mid \cdots \mid S_n).
\end{aligned}
$$

The result then follows easily, since we can reduce both Q and R to the process

$$(\nu\tilde{x}{:}\tilde{T})\,(S_1 \mid \cdots \mid S_n \mid [\tilde{y}/\tilde{z}]S \mid [\tilde{y}'/\tilde{z}']S').$$

The cases for reductions on local channels, replicated inputs, and conditional expressions can be proved in a similar manner. □

5. BISIMULATION

Our final task is to formalize process equivalence in the presence of linear types, so that we can check that equivalences like those claimed in the introduction really hold. We adapt Milner and Sangiorgi's notion of *barbed bisimulation* [Milner and Sangiorgi 1992; Sangiorgi 1992], which is suited to the job for several reasons. First, it is naturally a relation on *typed* processes, which is crucial here. Second, it is a fairly "modular" definition, in the sense that barbed bisimilarities for many different calculi can be formulated almost identically; this facilitates comparison and helps give confidence that the definition is natural. And finally, like all varieties of bisimilarity, it comes with a coinductive proof technique.

We begin by defining what "tests" can be made of a process by an external observer, relative to a certain type environment Γ.

Definition 5.1. We say that P *exhibits the barb a under type environment* Γ, written $P \Downarrow_a^\Gamma$, if P has the immediate possibility of performing an input or output action on the channel a and if the type of a in Γ is such that the observer can supply the other side of the communication—i.e., if it is not the case that a is linear and both its input and output capabilities are used by P. Formally, $P \Downarrow_a^{\Gamma, a:T}$ iff one of the following holds:

(1) $P \equiv (\nu\tilde{x}{:}\tilde{V})\,(Q_1 \mid \cdots \mid Q_n \mid a![\tilde{y}])$ and either $T = p^\omega[\tilde{U}]$ or $T = !^1[\tilde{U}]$
(2) $P \equiv (\nu\tilde{x}{:}\tilde{V})\,(Q_1 \mid \cdots \mid Q_n \mid a?[\tilde{y}].\,R)$ and either $T = p^\omega[\tilde{U}]$ or $T = ?^1[\tilde{U}]$
(3) $P \equiv (\nu\tilde{x}{:}\tilde{U})\,(Q_1 \mid \cdots \mid Q_n \mid a?^*[\tilde{y}].\,R)$

In each case, our assumption that all bound variables have distinct names ensures that the channel a cannot be one of the $\tilde{x}$. We write $\mathit{Barbs}_\Gamma(P)$ for the set $\{a \mid P \Downarrow_a^\Gamma\}$.

Experts should note that this definition of barbs agrees with the standard one for the full pi-calculus. Since we are working with an asynchronous fragment here, with output atoms instead of output prefixes, we do not actually need to allow observation of inputs: we could drop clauses (2) and (3) of the definition and obtain a notion of observation closer to "what processes can see about each other" (cf. [Amadio et al. 1996; Honda and Tokoro 1991]). We conjecture that the theory developed in the rest of the article would not be affected by this change.

One slight complication is that we cannot require that bisimilar processes be well typed in exactly the same type environment, since each linear communication removes capabilities from the type environment and we want to allow the possibility that two processes are bisimilar even though they may use up their linear resources in different orders. We deal with this by allowing bisimilar processes to be typed in different environments, provided that both environments can be obtained by removing linear capabilities from some common environment.

Definition 5.2. The set of *remnants* of a type environment Γ, written $\mathit{Rems}(\Gamma)$, comprises all the type environments that can be obtained from Γ by replacing some subset $x_1 : \updownarrow^1[\tilde{T}_1], \ldots, x_n : \updownarrow^1[\tilde{T}_n]$ of its bindings by their remnants $x_1 : \emptyset^1[\tilde{T}_1], \ldots, x_n : \emptyset^1[\tilde{T}_n]$.

Definition 5.3 (Γ-Relations). A relation $\mathcal{R}$ is a Γ-*relation* if

$$\mathcal{R} \subseteq \{ (P, Q) \mid \Delta, \Lambda \in \mathit{Rems}(\Gamma) \ \wedge\ \Delta \vdash P \ \wedge\ \Lambda \vdash Q \}.$$

We say that two processes are bisimilar if they have the same sets of observable actions and if, for each step that one can make, the other can make a step and reach a state that is again bisimilar.

Definition 5.4 (Strong Barbed Bisimulation). A Γ-relation $\mathcal{R}$ is a strong Γ-*bisimulation* if, for each $(P, Q) \in \mathcal{R}$,

(1) $\mathit{Barbs}_\Gamma(P) = \mathit{Barbs}_\Gamma(Q)$,
(2) $P \longrightarrow P'$ implies $Q \longrightarrow Q'$ and $(P', Q') \in \mathcal{R}$, for some Q', and
(3) $Q \longrightarrow Q'$ implies $P \longrightarrow P'$ and $(P', Q') \in \mathcal{R}$, for some P'.

We say that P and Q are *strongly* Γ-*bisimilar*, written $P \overset{\bullet}{\sim}_\Gamma Q$, if $(P, Q) \in \mathcal{R}$ for some strong Γ-bisimulation $\mathcal{R}$.

As usual, strong barbed bisimulation has a more useful, but somewhat more complex, counterpart, called *weak bisimulation*, where we drop the requirement that the two processes being compared must proceed in lock-step and instead allow each to take zero or many steps to match a single step of the other. We write $\mathit{WBarbs}_\Gamma(P)$ for the set $\{a \mid P \longrightarrow^* P' \ \wedge\ P' \Downarrow_a^\Gamma\}$.

Definition 5.5 (Weak Barbed Bisimulation). A Γ-relation $\mathcal{R}$ is a weak Γ-*bisimulation* if, for each $(P, Q) \in \mathcal{R}$,

(1) $\mathit{WBarbs}_\Gamma(P) = \mathit{WBarbs}_\Gamma(Q)$,
(2) $P \longrightarrow P'$ implies $Q \longrightarrow^* Q'$ and $(P', Q') \in \mathcal{R}$, for some Q', and
(3) $Q \longrightarrow Q'$ implies $P \longrightarrow^* P'$ and $(P', Q') \in \mathcal{R}$, for some P'.

We say that P and Q are *weakly* Γ-*bisimilar*, written $P \overset{\bullet}{\approx}_\Gamma Q$, if $(P, Q) \in \mathcal{R}$ for some weak Γ-bisimulation $\mathcal{R}$.

By itself, barbed bisimulation is a rather weak relation. In order to make it interesting, we need to close it under substitution and parallel composition. The resulting relation, called barbed congruence, will later be seen to be a full congruence for all of the other process constructors (Theorem 5.25).

Definition 5.6 (Barbed Congruence). P and Q are Γ-*congruent*, written $P \approx_\Gamma Q$, if $\Gamma \vdash P$, $\Gamma \vdash Q$, and for any Γ-Δ substitution σ and $\Theta \vdash R$ such that $\Theta + \Delta$ is well defined, $R \,|\, \sigma P \stackrel{\bullet}{\approx}_{\Theta+\Delta} R \,|\, \sigma Q$.

Proving that two processes are weakly barbed bisimilar is often difficult, since we need to consider all possible reduction sequences. The following theorem allows us to compare strong barbs instead of weak barbs, which substantially simplifies proofs.

Theorem 5.7. *A Γ-relation $\mathcal{R}$ is a weak Γ-bisimulation if, for each $(P, Q) \in \mathcal{R}$,*

(1) $Barbs_\Gamma(P) \subseteq WBarbs_\Gamma(Q)$ and $Barbs_\Gamma(Q) \subseteq WBarbs_\Gamma(P)$,

(2) $P \longrightarrow P'$ implies $Q \longrightarrow^* Q'$ and $(P', Q') \in \mathcal{R}$, for some Q', and

(3) $Q \longrightarrow Q'$ implies $P \longrightarrow^* P'$ and $(P', Q') \in \mathcal{R}$, for some P'.

Proof. We prove that if $(P, Q) \in \mathcal{R}$ then $WBarbs_\Gamma(P) = WBarbs_\Gamma(Q)$. Suppose that $a \in WBarbs_\Gamma(P)$. Then there is some P' such that $P \longrightarrow^* P'$ and $a \in Barbs_\Gamma(P')$. By condition (2), there is a Q' such that $Q \longrightarrow^* Q'$ with $(P', Q') \in \mathcal{R}$. By condition (1), $a \in WBarbs_\Gamma(Q')$. Since $WBarbs_\Gamma(Q') \subseteq WBarbs_\Gamma(Q)$, we know that a is in $WBarbs_\Gamma(Q)$, so $WBarbs_\Gamma(P) \subseteq WBarbs_\Gamma(Q)$. Similarly, $WBarbs_\Gamma(Q) \subseteq WBarbs_\Gamma(P)$. $\square$

We also need a form of *weak bisimulation up to* $\stackrel{\bullet}{\approx}_\Delta$ [Milner 1989; Sangiorgi 1994] (note that the standard construction of weak bisimulation up to $\stackrel{\bullet}{\approx}_\Delta$ is not valid [Sangiorgi and Milner 1992], so we use a weaker version). In the following, we use juxtaposition to indicate composition of relations. For example, $P\, \mathcal{R}_1\mathcal{R}_2\mathcal{R}_3\, Q$ iff there exist P' and Q' such that $P\mathcal{R}_1 P'$, $P'\mathcal{R}_2 Q'$ and $Q'\mathcal{R}_3 Q$.

Definition 5.8 (Weak Bisimulation Up to $\stackrel{\bullet}{\approx}_\Gamma$). A Γ-relation $\mathcal{R}$ is a weak Γ-bisimulation up to $\stackrel{\bullet}{\approx}_\Gamma$ if, for each $(P, Q) \in \mathcal{R}$,

(1) $Barbs_\Gamma(P) \subseteq WBarbs_\Gamma(Q)$ and $Barbs_\Gamma(Q) \subseteq WBarbs_\Gamma(P)$,

(2) $P \longrightarrow P'$ implies $Q \longrightarrow^* Q'$ and $(P', Q') \in \stackrel{\bullet}{\sim}_\Gamma \mathcal{R} \stackrel{\bullet}{\approx}_\Gamma$, for some Q', and

(3) $Q \longrightarrow Q'$ implies $P \longrightarrow^* P'$ and $(P', Q') \in \stackrel{\bullet}{\approx}_\Gamma \mathcal{R} \stackrel{\bullet}{\sim}_\Gamma$, for some P'.

We sometimes use a even simpler variant of weak Γ-bisimulation up to $\stackrel{\bullet}{\approx}_\Gamma$, where we replace $\stackrel{\bullet}{\sim}_\Gamma \mathcal{R} \stackrel{\bullet}{\approx}_\Gamma$ and $\stackrel{\bullet}{\approx}_\Gamma \mathcal{R} \stackrel{\bullet}{\sim}_\Gamma$ with $\equiv\mathcal{R}\equiv$. We call such a relation a weak Γ-bisimulation up to $\equiv$. The properties of weak Γ-bisimulation up to $\equiv$ can be proved trivially from the properties of weak Γ-bisimulation up to $\stackrel{\bullet}{\approx}_\Gamma$, since $\Gamma \vdash P$ and $P \equiv Q$ implies $P \stackrel{\bullet}{\approx}_\Gamma Q$.

Example 5.9. As a simple example, it is easy to check that processes $x![] \,|\, y![] \,|\, x?[].\, y?[].\, \mathbf{0}$ and $x![] \,|\, y![] \,|\, y?[].\, x?[].\, \mathbf{0}$ are congruent under the environment $\Delta = x : \updownarrow^1[], y : \updownarrow^1[]$, and that they are *not* congruent under the environment $\Delta' = x : \updownarrow^\omega[], y : \updownarrow^\omega[]$.

PROOF SKETCH. Let $P_1 = x![]\,|\,y![]\,|\,x?[].\,y?[].\,\mathbf{0}$ and $P_2 = x![]\,|\,y![]\,|\,y?[].\,x?[].\,\mathbf{0}$. Since $\Delta \vdash P_1$, $\Delta \vdash P_2$ and substitutions can only rename x and/or y with (distinct) fresh names, it is enough to check that for any $\Gamma \vdash Q$ such that $\Gamma + \Delta$ is well defined, $Q\,|\,P_1 \overset{\bullet}{\approx}_{\Gamma+\Delta} Q\,|\,P_2$. This follows from the fact that

$$\{(Q'\,|\,P_1, Q'\,|\,P_2), (Q'\,|\,y![]\,|\,y?[].\,\mathbf{0}, Q'\,|\,x![]\,|\,x?[].\,\mathbf{0}), (Q'\,|\,\mathbf{0}, Q'\,|\,\mathbf{0}) \;\mid\; Q \longrightarrow^* Q'\}$$

is a weak $(\Gamma + \Delta)$-bisimulation up to $\equiv$. □

We can now express the partial confluence of linear reduction (Theorem 4.4.3) in a more useful way: we prove that reduction on a local linear channel does not change the congruence class of a process.

LEMMA 5.10. *If $\Delta \vdash P$, $P \xrightarrow{\alpha} Q$ and $\Delta(\alpha) = 1$ then $P \overset{\bullet}{\approx}_\Delta Q$.*

PROOF. Let $\mathcal{R}$ be the relation $\{(P', Q') \mid P \longrightarrow^* P' \;\wedge\; P' \xrightarrow{\alpha} Q'\}$. We use Theorem 5.7 to prove that $\mathcal{R} \cup \overset{\bullet}{\approx}_\Delta$ is a weak Δ-bisimulation. Since $\overset{\bullet}{\approx}_\Delta$ is itself a weak bisimulation, we only need to consider pairs coming from R.

We first check the conditions on barbs. The fact that $\mathit{Barbs}_\Delta(Q') \subseteq \mathit{WBarbs}_\Delta(P')$ follows immediately from the definition of weak barbs. The reduction $P' \xrightarrow{\alpha} Q'$ can only remove the barb α, but since $\Delta(\alpha) = 1$ and α has even polarity we have, by Definition 5.1, that $\alpha \notin \mathit{Barbs}_\Delta(P')$. Thus, $\mathit{Barbs}_\Delta(P') \subseteq \mathit{WBarbs}_\Delta(Q')$ as required.

We now just need to consider the possible reductions of pairs in $\mathcal{R}$. Consider any reduction $P' \xrightarrow{\beta} P''$. If $P'' \equiv Q'$, then it is matched by $Q' \longrightarrow^* Q'$. Otherwise, by Theorem 4.4.3, there is some Q'' such that $P'' \xrightarrow{\alpha} Q''$ and $Q' \xrightarrow{\beta} Q''$. Therefore, the reduction is matched by $Q' \xrightarrow{\beta} Q''$ and $(P'', Q'') \in \mathcal{R}$.

If $Q' \longrightarrow Q''$ then $P' \xrightarrow{\alpha} Q' \longrightarrow Q''$ and $Q'' \overset{\bullet}{\approx}_\Delta Q''$ as required. □

THEOREM 5.11. *If $\Delta \vdash P$ and $P \xrightarrow{1} Q$ then $P \approx_\Delta Q$.*

PROOF. Suppose that σ is a Δ-Θ substitution, that $\Lambda \vdash R$, and that $\Lambda + \Theta$ is well defined. We must show $R\,|\,\sigma P \overset{\bullet}{\approx}_{\Delta+\Theta} R\,|\,\sigma Q$. By using Lemma 3.6 and rule E-PAR, we have $\Lambda + \Theta \vdash R\,|\,\sigma P$. $P \xrightarrow{1} Q$ implies $R\,|\,\sigma P \xrightarrow{1} R\,|\,\sigma Q$. So the result follows by Lemma 5.10. □

Technical Properties

Our goal in the rest of this section is to show that barbed congruence $\approx_\Delta$ is indeed a congruence relation. After checking the validity of the proof technique using "weak bisimulation up to" (in Lemma 5.15), we use it to show that $\approx_\Delta$ is closed under each process constructor in Lemmas 5.18–5.24.

LEMMA 5.12. *If $\Gamma \vdash P$ and $P \equiv Q$ then $P \overset{\bullet}{\sim}_\Gamma Q$.*

PROOF. Follows from the fact that types, barbs, and reductions are preserved under $\equiv$. □

LEMMA 5.13. *If $P \overset{\bullet}{\sim}_\Delta Q$ then $P \overset{\bullet}{\approx}_\Delta Q$.*

PROOF. Straightforward using the definitions of $\overset{\bullet}{\sim}_\Delta$ and $\overset{\bullet}{\approx}_\Delta$. □

Lemma 5.14. *Suppose that $\mathcal{R}$ is a weak Γ-bisimulation up to $\dot{\approx}_\Gamma$. If $P\mathcal{R}Q$ and $P \longrightarrow^* P'$, then $P' \dot{\sim}_\Gamma \mathcal{R} \dot{\approx}_\Gamma Q'$ and $Q \longrightarrow^* Q'$ for some Q'.*

Proof. We proceed by induction on the number of reduction steps in $P \longrightarrow^* P'$. If $P = P'$ then we can choose $Q' = Q$, and the result follows, since $(P, Q) \in \mathcal{R}$ and $\mathcal{R} \subseteq \dot{\sim}_\Gamma \mathcal{R} \dot{\approx}_\Gamma$. Otherwise, suppose $P \longrightarrow^* P'' \longrightarrow P'$. Using the induction hypothesis, we have $Q \longrightarrow^* Q''$ and $P'' \dot{\sim}_\Gamma R \mathcal{R} S \dot{\approx}_\Gamma Q''$ for some R, S, and Q''. Since $P'' \longrightarrow P'$ and $P'' \dot{\sim}_\Gamma R$, there exists an R' such that $R \longrightarrow R'$ and $P' \dot{\sim}_\Gamma R'$. Since $R \mathcal{R} S$ and $R \longrightarrow R'$, we have that there exists an S' such that $R' \dot{\sim}_\Gamma \mathcal{R} \dot{\approx}_\Gamma S'$ and $S \longrightarrow^* S'$. This implies that there exists a Q' such that, $Q'' \longrightarrow^* Q'$ and $S' \dot{\approx}_\Gamma Q'$. We therefore have that $Q \longrightarrow^* Q'' \longrightarrow^* Q'$ and $P' \dot{\sim}_\Gamma R' \dot{\sim}_\Gamma \mathcal{R} \dot{\approx}_\Gamma S' \dot{\approx}_\Gamma Q'$, which implies that $P' \dot{\sim}_\Gamma \mathcal{R} \dot{\approx}_\Gamma Q'$ as required. □

Lemma 5.15. *If $\mathcal{R}$ is a weak Γ-bisimulation up to $\dot{\approx}_\Gamma$, then $\mathcal{R} \subseteq \dot{\approx}_\Gamma$.*

Proof. Since $\mathcal{R} \subseteq \dot{\approx}_\Gamma \mathcal{R} \dot{\approx}_\Gamma$ it is sufficient to prove that $\dot{\approx}_\Gamma \mathcal{R} \dot{\approx}_\Gamma$ is a weak Γ-bisimulation. We use Theorem 5.7. Suppose that $(P, Q) \in \dot{\approx}_\Gamma \mathcal{R} \dot{\approx}_\Gamma$. It is easy to check the appropriate conditions on barbs: $\mathit{Barbs}_\Gamma(P) \subseteq \mathit{WBarbs}_\Gamma(Q)$ and $\mathit{Barbs}_\Gamma(Q) \subseteq \mathit{WBarbs}_\Gamma(P)$. The proofs relating reductions in P and Q are similar, so we just show the proof for P. Suppose that $P \longrightarrow P'$. We have, by assumption, that $P \dot{\approx}_\Gamma R \mathcal{R} S \dot{\approx}_\Gamma Q$, for some R and S. Since $P \dot{\approx}_\Gamma R$, we have that $R \longrightarrow^* R'$ and $P' \dot{\approx}_\Gamma R'$ for some R'. Using Lemma 5.14, there exists an S' such that $S \longrightarrow^* S'$ and $(R', S') \in \dot{\sim}_\Gamma \mathcal{R} \dot{\approx}_\Gamma$. Finally, since $S \dot{\approx}_\Gamma Q$, there exists a Q' such that $S' \dot{\approx}_\Gamma Q'$ and $Q \longrightarrow^* Q'$. So, $(P', Q') \in \dot{\approx}_\Gamma \dot{\sim}_\Gamma \mathcal{R} \dot{\approx}_\Gamma \dot{\approx}_\Gamma$, which implies that $(P', Q') \in \dot{\approx}_\Gamma \mathcal{R} \dot{\approx}_\Gamma$ as required. □

Lemma 5.16 (Closure of Congruence under Substitution). *If $P \approx_\Delta Q$ then $\sigma P \approx_\Gamma \sigma Q$ for any Δ-Γ substitution σ.*

Proof. Follows from our substitution lemma (3.6) and the definition of congruence, since composing a Δ-Γ substitution with a Γ-Θ substitution yields a Δ-Θ substitution. □

Lemma 5.17 (Closure of Bisimulation under Addition of Capabilities). *If $P \dot{\approx}_\Delta Q$ and $\Delta \in \mathit{Rems}(\Gamma)$ then $P \dot{\approx}_\Gamma Q$.*

Proof. Suppose $\mathcal{R}$ is a Δ-bisimulation containing (P, Q). It is easy to check that $\mathcal{R}$ is also a Γ-bisimulation, since $\mathit{Rems}(\Delta) \subseteq \mathit{Rems}(\Gamma)$ and $R \Downarrow_a^\Gamma$ iff $R \Downarrow_a^\Delta$. (Δ only differs from Γ in that some linear bindings of the form $x : \updownarrow^1[\tilde{T}]$ have been replaced by $x : \emptyset^1[\tilde{T}]$. This can never affect the actions visible to the observer.) □

Lemma 5.18 (Congruence for $|$). *If $P_1 \approx_\Delta P_2$ then, for each Γ and Q such that $\Gamma \vdash Q$ and such that $\Gamma + \Delta$ is well defined, $Q \,|\, P_1 \approx_{\Gamma+\Delta} Q \,|\, P_2$.*

Proof. Since $\Delta \vdash P_1$ and $\Delta \vdash P_2$, $(\Gamma + \Delta) \vdash Q \,|\, P_1$ and $(\Gamma + \Delta) \vdash Q \,|\, P_2$. Suppose that σ is a $(\Gamma + \Delta)$-Θ substitution, that $\Lambda \vdash R$, and that $\Lambda + \Theta$ is well defined. By Lemma 3.4, there are Θ_1, Θ_2 such that $\Theta_1 + \Theta_2 = \Theta$ and such that σ is both a Γ-Θ_1 and a Δ-Θ_2 substitution. Moreover, by Lemma 3.6, $\Theta_1 \vdash \sigma Q$, $\Theta_2 \vdash \sigma P_1$ and $\Theta_2 \vdash \sigma P_2$. Since $\Lambda + (\Theta_1 + \Theta_2)$ is well defined, so is $\Lambda + \Theta_1$. Therefore, we have $\Lambda + \Theta_1 \vdash R \,|\, \sigma Q$. The assumption $P_1 \approx_\Delta P_2$ now yields

$$(R \,|\, \sigma Q) \,|\, \sigma P_1 \stackrel{\bullet}{\approx}_{(\Lambda+\Theta_1)+\Theta_2} (R \,|\, \sigma Q) \,|\, \sigma P_2.$$

Since $(\Lambda + \Theta_1) + \Theta_2 = \Lambda + \Theta$, $R \,|\, \sigma(Q \,|\, P_1) \equiv (R \,|\, \sigma Q) \,|\, \sigma P_1$ and $R \,|\, \sigma(Q \,|\, P_2) \equiv (R \,|\, \sigma Q) \,|\, \sigma P_2$, we obtain

$$R \,|\, \sigma(Q \,|\, P_1) \stackrel{\bullet}{\approx}_{\Lambda+\Theta} R \,|\, \sigma(Q \,|\, P_2)$$

by Lemma 5.12, as required. □

Lemma 5.19. *Suppose Δ is a type environment and Δ_P and Δ_Q are remnants of Δ. If $\Delta_P, x : p^m[\tilde{T}] \vdash P$ and $\Delta_Q, x : p^m[\tilde{T}] \vdash Q$ where $P \stackrel{\bullet}{\approx}_{\Delta, x:p^m[\tilde{T}]} Q$ then*

$$(\nu x : p^m[\tilde{T}])\, P \stackrel{\bullet}{\approx}_\Delta (\nu x : p^m[\tilde{T}])\, Q.$$

Proof. Suppose $\mathcal{R}$ is a weak $(\Delta, x : p^m[\tilde{T}])$-bisimulation containing (P, Q). It must be the case that, for each $(P', Q') \in \mathcal{R}$, there exist $\Delta_{P'}$, $T_{P'}$, $\Delta_{Q'}$ and $T_{Q'}$ such that $\Delta_{P'}, x : T_{P'} \vdash P'$, $\Delta_{Q'}, x : T_{Q'} \vdash Q'$, and both $\Delta_{P'}, x : T_{P'}$ and $\Delta_{Q'}, x : T_{Q'}$ are remnants of $\Delta, x : p^m[\tilde{T}]$. Let $\mathcal{R}'$ be the relation $\{((\nu x : T_{P'})\, P', (\nu x : T_{Q'})\, Q') \mid (P', Q') \in \mathcal{R}\}$. We now show that $\mathcal{R}'$ is a weak Δ-bisimulation.

Firstly, we have that

$$\begin{array}{rl} & \mathit{WBarbs}_\Delta((\nu x : T_{P'})\, P') = \mathit{WBarbs}_{\Delta, x:T_{P'}}(P') \backslash \{x\} \\ = & \mathit{WBarbs}_{\Delta, x:T_{Q'}}(Q') \backslash \{x\} = \mathit{WBarbs}_\Delta((\nu x : T_{Q'})\, Q'). \end{array}$$

Secondly, consider any reduction $(\nu x : T_{P'})\, P' \longrightarrow R$. Using Lemma 4.2.1, we have that $R \equiv (\nu x : T_{P''})\, P''$ for some $T_{P''}$ and P'', where $P' \xrightarrow{\alpha} P''$. Since $\mathcal{R}$ is a $(\Delta, x : p^m[\tilde{T}])$-bisimulation containing (P', Q'), there are $T_{Q''}$ and Q'' such that $Q' \longrightarrow^* Q''$ and $(P'', Q'') \in \mathcal{R}$ from which we obtain $(\nu x : T_{Q'})\, Q' \longrightarrow^* (\nu x : T_{Q''})\, Q''$ where $((\nu x : T_{P''})\, P'', (\nu x : T_{Q''})\, Q'') \in \mathcal{R}'$ as required. If $(\nu x : T_{Q'})\, Q' \longrightarrow R$ then we can prove the result in a similar manner. □

Lemma 5.20 (Congruence for ν). *If $P_1 \approx_{\Delta, x:T} P_2$ and $T = p^m[\tilde{U}]$ then $(\nu x : T)\, P_1 \approx_\Delta (\nu x : T)\, P_2$.*

Proof. It is easy to check that $\Delta \vdash (\nu x : T)\, P_1$ and $\Delta \vdash (\nu x : T)\, P_2$. Suppose σ is a Δ-Γ substitution, that $\Theta \vdash Q$, and that $\Theta + \Gamma$ is well defined. We must show that $Q \,|\, \sigma((\nu x : T)\, P_1) \stackrel{\bullet}{\approx}_{\Theta+\Gamma} Q \,|\, \sigma((\nu x : T)\, P_2)$.

Let σ' be the $(\Delta, x : T)$-$(\Gamma, x : T)$ substitution $\sigma[x/x]$. Then $P_1 \approx_{\Delta, x:T} P_2$ implies $Q \,|\, \sigma' P_1 \stackrel{\bullet}{\approx}_{\Theta+(\Gamma, x:T)} Q \,|\, \sigma' P_2$. By Lemma 5.19, $(\nu x : T)\, (Q \,|\, \sigma' P_1) \stackrel{\bullet}{\approx}_{\Theta+\Gamma} (\nu x : T)\, (Q \,|\, \sigma' P_2)$. However,

$$\begin{array}{rcl} (\nu x : T)\, (Q \,|\, \sigma' P_1) & \equiv & Q \,|\, \sigma((\nu x : T)\, P_1) \\ (\nu x : T)\, (Q \,|\, \sigma' P_2) & \equiv & Q \,|\, \sigma((\nu x : T)\, P_2), \end{array}$$

so the result follows by Lemma 5.12. □

Lemma 5.21 (Congruence for If). *If $P_1 \approx_\Delta P_2$, $\Delta(b) = \mathit{Bool}$, and $\Delta \vdash Q$, then if b then P_1 else $Q \approx_\Delta$ if b then P_2 else Q, and if b then Q else $P_1 \approx_\Delta$ if b then Q else P_2.*

PROOF. We check just the first case; the other is similar. It is easy to check that $\Delta \vdash \text{if } b \text{ then } P_1 \text{ else } Q$ and $\Delta \vdash \text{if } b \text{ then } P_2 \text{ else } Q$. Consider any Δ-Γ substitution σ and $\Theta \vdash R$ such that $\Theta + \Gamma$ is well defined. We must show that

$$R \,|\, \text{if } \sigma b \text{ then } \sigma P_1 \text{ else } \sigma Q \;\dot{\approx}_{\Theta+\Gamma}\; R \,|\, \text{if } \sigma b \text{ then } \sigma P_2 \text{ else } \sigma Q.$$

Suppose $\mathcal{R}$ is the relation below:

$$\{(R' \,|\, \text{if } \sigma b \text{ then } \sigma P_1 \text{ else } \sigma Q, R' \,|\, \text{if } \sigma b \text{ then } \sigma P_2 \text{ else } \sigma Q) \mid R \longrightarrow^* R'\}$$

Theorem 5.7 and Lemma 5.15 prove that it suffices to show that the relation $\mathcal{R} \cup \dot{\approx}_{\Theta+\Gamma}$ is a weak $(\Theta + \Gamma)$-bisimulation up to $\equiv$. Since $\dot{\approx}_{\Theta+\Gamma}$ is itself a bisimulation, we only need to consider pairs coming from $\mathcal{R}$. It is easy to check the conditions on barbs, since $\mathit{Barbs}_{\Theta+\Gamma}(\text{if } \sigma b \text{ then } \sigma P_i \text{ else } \sigma Q) = \emptyset$:

$$\begin{array}{rcl} \mathit{Barbs}_{\Theta+\Gamma}(R' \,|\, \text{if } \sigma b \text{ then } \sigma P_1 \text{ else } \sigma Q) & \subseteq & \mathit{WBarbs}_{\Theta+\Gamma}(R' \,|\, \text{if } \sigma b \text{ then } \sigma P_2 \text{ else } \sigma Q) \\ \mathit{Barbs}_{\Theta+\Gamma}(R' \,|\, \text{if } \sigma b \text{ then } \sigma P_2 \text{ else } \sigma Q) & \subseteq & \mathit{WBarbs}_{\Theta+\Gamma}(R' \,|\, \text{if } \sigma b \text{ then } \sigma P_1 \text{ else } \sigma Q) \end{array}$$

We now just need to check the possible reductions of pairs in $\mathcal{R}$.

(1) $R' \,|\, (\text{if } \sigma b \text{ then } \sigma P_1 \text{ else } \sigma Q) \;\longrightarrow\equiv\; R'' \,|\, (\text{if } \sigma b \text{ then } \sigma P_1 \text{ else } \sigma Q)$ where $R' \longrightarrow R''$. This can be matched by

$$R' \,|\, (\text{if } \sigma b \text{ then } \sigma P_2 \text{ else } \sigma Q) \longrightarrow R'' \,|\, (\text{if } \sigma b \text{ then } \sigma P_2 \text{ else } \sigma Q),$$

and the result follows, since

$$(R'' \,|\, \text{if } \sigma b \text{ then } \sigma P_1 \text{ else } \sigma Q, R'' \,|\, \text{if } \sigma b \text{ then } \sigma P_2 \text{ else } \sigma Q) \in \mathcal{R}.$$

(2) $R' \,|\, (\text{if } \sigma b \text{ then } \sigma P_1 \text{ else } \sigma Q) \;\longrightarrow\equiv\; R' \,|\, \sigma P_1$. This can be matched by $R' \,|\, (\text{if } \sigma b \text{ then } \sigma P_2 \text{ else } \sigma Q) \longrightarrow R' \,|\, \sigma P_2$. Since $P_1 \approx_\Delta P_2$ and σ is a Δ-Γ substitution, $R' \,|\, \sigma P_1 \dot{\approx}_{\Theta+\Gamma} R' \,|\, \sigma P_2$ as required.

(3) $R' \,|\, (\text{if } \sigma b \text{ then } \sigma P_1 \text{ else } \sigma Q) \;\longrightarrow\equiv\; R' \,|\, \sigma Q$. This can be matched by $R' \,|\, (\text{if } \sigma b \text{ then } \sigma P_2 \text{ else } \sigma Q) \longrightarrow R' \,|\, \sigma Q$. We trivially have that $R' \,|\, \sigma Q \dot{\approx}_{\Theta+\Gamma} R' \,|\, \sigma Q$ as required.

(The cases involving the reductions of $R' \,|\, (\text{if } \sigma x \text{ then } \sigma P_2 \text{ else } \sigma Q)$ are symmetric to those above.) ☐

Definition 5.22. For each type T, the type $\mathit{Rem}(T)$ is defined as follows:

$$\begin{array}{rcl} \mathit{Rem}(\mathit{Bool}) & = & \mathit{Bool} \\ \mathit{Rem}(p^\omega[\tilde{T}]) & = & p^\omega[\tilde{T}] \\ \mathit{Rem}(p^1[\tilde{T}]) & = & \emptyset^1[\tilde{T}] \end{array}$$

Note that $\mathit{Rem}(T)$ is always unlimited.

LEMMA 5.23 (CONGRUENCE FOR ?). *If $P_1 \approx_{\Delta,\tilde{y}:\tilde{T}} P_2$ and $\Delta + x : ?^m[\tilde{T}]$ is well defined, then $x?[\tilde{y}].\, P_1 \approx_{\Delta+x:?^m[\tilde{T}]} x?[\tilde{y}].\, P_2$.*

PROOF. It is easy to check that $\Delta + x : ?^m[\tilde{T}] \vdash x?[\tilde{y}].\, P_1$ and $\Delta + x : ?^m[\tilde{T}] \vdash x?[\tilde{y}].\, P_2$. Consider any $(\Delta + x : ?^m[\tilde{T}])$-$\Gamma$ substitution σ and $\Theta \vdash R$ such that $\Theta + \Gamma$ is well defined. We must show that $R \,|\, \sigma(x?[\tilde{y}].\, P_1) \dot{\approx}_{\Theta+\Gamma} R \,|\, \sigma(x?[\tilde{y}].\, P_2)$. Note that since $\tilde{y}$ are bound variables, we may assume without loss of generality that $\tilde{y} \cap \mathit{dom}(\sigma) = \emptyset$ and $\tilde{y} \cap \mathit{codom}(\sigma) = \emptyset$.

Let $\mathcal{R}$ be the relation $\{(R' \mid \sigma x?[\tilde{y}].\,\sigma P_1, R' \mid \sigma x?[\tilde{y}].\,\sigma P_2) \mid R \longrightarrow^* R'\}$. We will show that $\mathcal{R} \cup \stackrel{\bullet}{\approx}_{\Theta+\Gamma}$ is a weak $(\Theta + \Gamma)$-bisimulation up to $\equiv$. Since $\stackrel{\bullet}{\approx}_{\Theta+\Gamma}$ is itself a bisimulation, we only need to consider pairs coming from $\mathcal{R}$. It is easy to check the conditions on barbs:

$$\begin{array}{rcl} \mathit{Barbs}_{\Theta+\Gamma}(R' \mid \sigma x?[\tilde{y}].\,\sigma P_1) & \subseteq & \mathit{WBarbs}_{\Theta+\Gamma}(R' \mid \sigma x?[\tilde{y}].\,\sigma P_2) \\ \mathit{Barbs}_{\Theta+\Gamma}(R' \mid \sigma x?[\tilde{y}].\,\sigma P_2) & \subseteq & \mathit{WBarbs}_{\Theta+\Gamma}(R' \mid \sigma x?[\tilde{y}].\,\sigma P_1) \end{array}$$

We now just need to consider the possible reductions of pairs in $\mathcal{R}$.

(1) $R' \mid \sigma x?[\tilde{y}].\,\sigma P_1 \longrightarrow\equiv R'' \mid \sigma x?[\tilde{y}].\,\sigma P_1$ where $R' \longrightarrow R''$. This can be matched by $R' \mid \sigma x?[\tilde{y}].\,\sigma P_2 \longrightarrow\equiv R'' \mid \sigma x?[\tilde{y}].\,\sigma P_2$, and the result follows, since $(R'' \mid \sigma x?[\tilde{y}].\,\sigma P_1, R'' \mid \sigma x?[\tilde{y}].\,\sigma P_2) \in \mathcal{R}$.

(2) $R' \mid \sigma x?[\tilde{y}].\,\sigma P_1 \longrightarrow\equiv (\nu\tilde{w}{:}\tilde{W})\,(R_1 \mid \cdots \mid R_n \mid [\tilde{z}/\tilde{y}]\sigma P_1)$. This can be matched by

$$R' \mid \sigma x?[\tilde{y}].\,\sigma P_2 \longrightarrow\equiv (\nu\tilde{w}{:}\tilde{W})\,(R_1 \mid \cdots \mid R_n \mid [\tilde{z}/\tilde{y}]\sigma P_2).$$

The result follows if we can prove that

$$(\nu\tilde{w}{:}\tilde{W})\,(R_1 \mid \cdots \mid R_n \mid [\tilde{z}/\tilde{y}]\sigma P_1) \stackrel{\bullet}{\approx}_{\Theta+\Gamma} (\nu\tilde{w}{:}\tilde{W})\,(R_1 \mid \cdots \mid R_n \mid [\tilde{z}/\tilde{y}]\sigma P_2).$$

Lemma 5.19 proves that it suffices to show that

$$R_1 \mid \cdots \mid R_n \mid [\tilde{z}/\tilde{y}]\sigma P_1 \stackrel{\bullet}{\approx}_{\Theta+\Gamma,\tilde{w}:\tilde{W}} R_1 \mid \cdots \mid R_n \mid [\tilde{z}/\tilde{y}]\sigma P_2.$$

Using Lemma 3.5, we have that there exists a Φ such that $\Gamma = \Phi + \sigma x : ?^m[\tilde{T}]$ and that σ is a $\Delta - \Phi$ substitution. Since $R \longrightarrow^* R' \equiv (\nu\tilde{w}{:}\tilde{W})\,(R_1 \mid \cdots \mid R_n \mid \sigma x![\tilde{z}])$, it must be the case that there exists a Λ such that $\Lambda + \tilde{z}{:}\tilde{T} + \sigma x{:}!^m[\tilde{T}]$ is a remnant of $\Theta, \tilde{w}{:}\tilde{W}$, and $\Lambda + x{:}\mathit{Rem}(\updownarrow^m[\tilde{T}]) \vdash R_1 \mid \cdots \mid R_n$. Using $\Gamma = \Phi + \sigma x{:}?^m[\tilde{T}]$, we know that $(\Lambda + x{:}\mathit{Rem}(\updownarrow^m[\tilde{T}])) + (\Phi + \tilde{z}{:}\tilde{T})$ is a remnant of $\Theta + \Gamma, \tilde{w}{:}\tilde{W}$. We also know that the composition of $[\tilde{z}/\tilde{y}]$ and σ is a $(\Delta, \tilde{y}{:}\tilde{T}) - (\Phi + \tilde{z}{:}\tilde{T})$ substitution. The result then follows from $P_1 \approx_{\Delta,\tilde{y}:\tilde{T}} P_2$ and Lemma 5.17.

(The cases involving reductions of $R' \mid \sigma x?[\tilde{y}].\,\sigma P_2$ are symmetric to those above.) $\square$

Lemma 5.24 (Congruence for $?^*$). *If $P_1 \approx_{\Delta,\tilde{y}:\tilde{T}} P_2$, Δ is unlimited and $\Delta + x : ?^\omega[\tilde{T}]$ is well defined, then $x?^*[\tilde{y}].\,P_1 \approx_{\Delta+x:?^\omega[\tilde{T}]} x?^*[\tilde{y}].\,P_2$.*

Proof. It is easy to check that $\Delta + x : ?^\omega[\tilde{T}] \vdash x?^*[\tilde{y}].\,P_1$ and $\Delta + x : ?^\omega[\tilde{T}] \vdash x?^*[\tilde{y}].\,P_2$. Consider any $(\Delta + x : ?^\omega[\tilde{T}])$-$\Gamma$ substitution σ and $\Theta \vdash R$ such that $\Theta + \Gamma$ is well defined. We must show that $R \mid \sigma(x?^*[\tilde{y}].\,P_1) \stackrel{\bullet}{\approx}_{\Theta+\Gamma} R \mid \sigma(x?^*[\tilde{y}].\,P_2)$. Note that since $\tilde{y}$ are bound variables, we may assume without loss of generality that $\tilde{y} \cap \mathit{dom}(\sigma) = \emptyset$ and $\tilde{y} \cap \mathit{codom}(\sigma) = \emptyset$.

Let $\mathcal{R}$ be the relation

$$\{(R' \mid \sigma x?^*[\tilde{y}].\,\sigma P_1, R' \mid \sigma x?^*[\tilde{y}].\,\sigma P_2) \mid R \mid \sigma x?^*[\tilde{y}].\,\sigma P_1 \longrightarrow^* R' \mid \sigma x?^*[\tilde{y}].\,\sigma P_1\}.$$

We first prove that $\mathcal{R}$ is a $(\Theta + \Gamma)$-relation. Using Theorem 4.3.1, we know that there exist Φ_1 and Φ_2 such that $\Phi_1 + \Phi_2 \vdash R' \mid \sigma x?^*[\tilde{y}].\,\sigma P_1$ where $\Phi_1 \vdash R'$, $\Phi_2 \vdash \sigma x?^*[\tilde{y}].\,\sigma P_1$, and $\Phi_1 + \Phi_2$ is a remnant of $\Theta + \Gamma$. Moreover, rule E-RIn ensures that Φ_2 is an unlimited type environment, so we can use Lemma 3.2 to prove that

$\Phi_1 + \Phi_2 \vdash R'$. Now, since Γ is unlimited and $\Phi_1 + \Phi_2$ is a remnant of $\Theta + \Gamma$, we have that $\Phi_1 + \Phi_2 + \Gamma$ is well formed. Therefore, using rule E-PAR and the fact $\Gamma \vdash \sigma x?^*[\tilde{y}].\,\sigma P_2$, we have that $\Phi_1 + \Phi_2 + \Gamma \vdash R' \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_2$, where $\Phi_1 + \Phi_2 + \Gamma$ is a remnant of $\Theta + \Gamma$ as required.

We will show that $\mathcal{R} \cup \dot{\approx}_{\Theta+\Gamma}$ is a weak $(\Theta + \Gamma)$-bisimulation up to $\dot{\approx}_{\Theta+\Gamma}$. Since $\dot{\approx}_{\Theta+\Gamma}$ is itself a bisimulation, we only need to consider pairs coming from $\mathcal{R}$. It is easy to check the conditions on barbs:

$$\begin{array}{rcl} \mathit{Barbs}_{\Theta+\Gamma}(R' \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_1) & \subseteq & \mathit{WBarbs}_{\Theta+\Gamma}(R' \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_2) \\ \mathit{Barbs}_{\Theta+\Gamma}(R' \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_2) & \subseteq & \mathit{WBarbs}_{\Theta+\Gamma}(R' \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_1) \end{array}$$

We now just need to consider the possible reductions of pairs in $\mathcal{R}$.

(1) $R' \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_1 \longrightarrow\equiv R'' \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_1$ where $R' \longrightarrow R''$. This can be matched by $R' \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_2 \longrightarrow\equiv R'' \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_2$, and the result follows since $(R'' \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_1, R'' \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_2) \in \mathcal{R}$.

(2) $R' \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_1 \longrightarrow\equiv (\nu\tilde{w}{:}\tilde{W})\,(R_1 \,|\, \cdots \,|\, R_n \,|\, [\tilde{z}/\tilde{y}]\sigma P_1) \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_1$. This reduction can be matched by

$$R' \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_2 \longrightarrow\equiv (\nu\tilde{w}{:}\tilde{W})\,(R_1 \,|\, \cdots \,|\, R_n \,|\, [\tilde{z}/\tilde{y}]\sigma P_2) \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_2$$

if it is the case that

$$\begin{array}{rl} & (\nu\tilde{w}{:}\tilde{W})\,(R_1 \,|\, \cdots \,|\, R_n \,|\, [\tilde{z}/\tilde{y}]\sigma P_1) \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_1 \\ \mathcal{R}\,\dot{\approx}_{\Theta+\Gamma} & (\nu\tilde{w}{:}\tilde{W})\,(R_1 \,|\, \cdots \,|\, R_n \,|\, [\tilde{z}/\tilde{y}]\sigma P_2) \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_2. \end{array}$$

This follows if we can prove that

$$\begin{array}{rl} & (\nu\tilde{w}{:}\tilde{W})\,(R_1 \,|\, \cdots \,|\, R_n \,|\, [\tilde{z}/\tilde{y}]\sigma P_1) \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_2 \\ \dot{\approx}_{\Theta+\Gamma} & (\nu\tilde{w}{:}\tilde{W})\,(R_1 \,|\, \cdots \,|\, R_n \,|\, [\tilde{z}/\tilde{y}]\sigma P_2) \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_2, \end{array}$$

since the pair

$$\begin{array}{l} ((\nu\tilde{w}{:}\tilde{W})\,(R_1 \,|\, \cdots \,|\, R_n \,|\, [\tilde{z}/\tilde{y}]\sigma P_1) \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_1, \\ \;(\nu\tilde{w}{:}\tilde{W})\,(R_1 \,|\, \cdots \,|\, R_n \,|\, [\tilde{z}/\tilde{y}]\sigma P_1) \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_2) \end{array}$$

is an element of $\mathcal{R}$. Lemma 5.19 proves that it suffices to show that

$$\begin{array}{rl} & R_1 \,|\, \cdots \,|\, R_n \,|\, [\tilde{z}/\tilde{y}]\sigma P_1 \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_2 \\ \dot{\approx}_{\Theta+\Gamma,\,\tilde{w}:\tilde{W}} & R_1 \,|\, \cdots \,|\, R_n \,|\, [\tilde{z}/\tilde{y}]\sigma P_2 \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_2. \end{array}$$

Using Lemma 3.5, we have that there exists a Φ such that $\Gamma = \Phi + \sigma x{:}?^\omega[\tilde{T}]$ and such that σ is a $\Delta - \Phi$ substitution. Since $R' \,|\, \sigma x?^*[\tilde{y}].\sigma P_2$ is well typed under a remnant of $\Theta + \Gamma$, it must be the case that there exists a Λ such that $\Lambda + \tilde{z}{:}\tilde{T} + \sigma x{:}!^\omega[\tilde{T}]$ is a remnant of $\Theta + \Gamma, \tilde{w}{:}\tilde{W}$ and $\Lambda + \sigma x{:}\updownarrow^\omega[\tilde{T}] \vdash R_1 \,|\, \cdots \,|\, R_n \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_2$. Since $\Gamma = \Phi + \sigma x{:}?^\omega[\tilde{T}]$ is unlimited, $(\Lambda + \sigma x{:}\updownarrow^\omega[\tilde{T}]) + (\Phi + \tilde{z}{:}\tilde{T})$ is also a remnant of $\Theta + \Gamma, \tilde{w}{:}\tilde{W}$. We also know that the composition of $[\tilde{z}/\tilde{y}]$ and σ is a $(\Delta, \tilde{y}{:}\tilde{T}) - (\Phi + \tilde{z}{:}\tilde{T})$ substitution. The result then follows from $P_1 \approx_{\Delta,\tilde{y}:\tilde{T}} P_2$ and Lemma 5.17.

(The cases involving reductions of $R' \,|\, \sigma x?^*[\tilde{y}].\,\sigma P_2$ are symmetric.) $\square$

THEOREM 5.25 (CONGRUENCE). *The relation $\approx_\Delta$ is a congruence relation.*

PROOF. Follows from Lemmas 5.18, 5.20, 5.21, 5.23, and 5.24. $\square$

6. EXAMPLE

We now apply the foregoing theory to show that the two versions of the *plustwo* process presented in the introduction are congruent. These processes are nontrivial, and the proof of their equivalence requires some care; it is obtained as a corollary of a general theorem (6.3), which states that a substitution of a linear channel can be simulated by running a forwarder process in parallel. We begin by introducing two technical lemmas to help structure the argument.

LEMMA 6.1. *If* $\Gamma = \Delta, x : p^1[\tilde{T}]$ *and* $\Gamma \vdash (\nu y : \emptyset^1[\tilde{T}])\, P$ *then* $[x/y]P \stackrel{\bullet}{\approx}_\Gamma (\nu y : \emptyset^1[\tilde{T}])\, P$.

PROOF. Let $\mathcal{R}$ be the Γ-relation $\{([x/y]P', (\nu y : \emptyset^1[\tilde{T}])\, P') \mid P \longrightarrow^* P'\}$. Because the pair $([x/y]P, (\nu y : \emptyset^1[\tilde{T}])\, P)$ is an element of $\mathcal{R}$, the result follows if we can prove that $\mathcal{R}$ is a weak Γ-bisimulation. By Lemmas 5.12 and 5.15, it suffices to show that $\mathcal{R}$ is a weak Γ-bisimulation up to $\equiv$. We first check the conditions on barbs: since $\Gamma, y : \emptyset^1[\tilde{T}] \vdash P$ and $P \longrightarrow^* P'$, P' cannot have barb y, so $\mathit{Barbs}_\Gamma([x/y]P') = \mathit{Barbs}_{\Gamma, y:\emptyset^1[\tilde{T}]}(P') = \mathit{Barbs}_\Gamma((\nu y : \emptyset^1[\tilde{T}])\, P')$.

We now just need to consider the possible reductions of pairs in $\mathcal{R}$. If $[x/y]P' \longrightarrow R$ then there exists an R' such that $P' \longrightarrow R'$ and $R = [x/y]R'$, since the type of y in P' is $\emptyset^1[\tilde{T}]$. This implies that $(\nu y : \emptyset^1[\tilde{T}])\, P' \longrightarrow (\nu y : \emptyset^1[\tilde{T}])\, R'$, where $([x/y]R', (\nu y : \emptyset^1[\tilde{T}])\, R') \in \mathcal{R}$ as required.

If $(\nu y : \emptyset^1[\tilde{T}])\, P' \longrightarrow\equiv (\nu y : \emptyset^1[\tilde{T}])\, R'$ then it must be the case that $P' \longrightarrow R'$, so we have that $[x/y]P' \longrightarrow [x/y]R'$ where $([x/y]R', (\nu y : \emptyset^1[\tilde{T}])\, R') \in \mathcal{R}$ as required. $\square$

LEMMA 6.2. *If* $\Delta, y : !^1[\tilde{T}] \vdash P$ *and* $\Delta + x : !^1[\tilde{T}]$ *is well defined, then*

$$[x/y]P \stackrel{\bullet}{\approx}_{\Delta + x:!^1[\tilde{T}]} (\nu y : \updownarrow^1[\tilde{T}])\, (P \mid y?[\tilde{z}].\, x![\tilde{z}]).$$

PROOF. Let $\Gamma = \Delta + x : !^1[\tilde{T}]$ and $\mathcal{R}$ be the Γ-relation

$$\{([x/y]P', (\nu y : \updownarrow^1[\tilde{T}])\, (P' \mid y?[\tilde{z}].\, x![\tilde{z}])) \mid P \longrightarrow^* P'\}.$$

We prove the result by showing that $\mathcal{R} \cup \stackrel{\bullet}{\approx}_\Gamma$ is a weak Γ-bisimulation up to $\equiv$. Since $\stackrel{\bullet}{\approx}_\Gamma$ is itself a weak bisimulation, we only need to consider pairs coming from $\mathcal{R}$. We first check the conditions on barbs (where $Q = (\nu y : \updownarrow^1[\tilde{T}])\, (P' \mid y?[\tilde{z}].\, x![\tilde{z}])$):

(1) $a \in \mathit{Barbs}_\Gamma(Q)$ implies $a \in \mathit{WBarbs}_\Gamma([x/y]P')$

(2) $a \in \mathit{Barbs}_\Gamma([x/y]P')$ implies $a \in \mathit{WBarbs}_\Gamma(Q)$

It is easy to check the result for any $a \neq x$. For case (1) where $a = x$, we find that $x \notin \mathit{Barbs}_\Gamma(Q)$ (the output $x![\tilde{v}]$ is guarded by an input on y, and there cannot be any other outputs on x, since x is a linear channel; if there is an input, $\Gamma(x) = \updownarrow^1[\tilde{T}]$, so $x \notin \mathit{Barbs}_\Gamma(Q)$). For case (2), suppose that $x \in \mathit{Barbs}_\Gamma([x/y]P')$. Since the type of y in P' is $!^1[\tilde{T}]$ it must be the case that $P' \equiv (\nu \tilde{w}{:}\tilde{U})\, (y![\tilde{v}] \mid R_1 \mid \cdots \mid R_n)$. We therefore have that Q reduces to a process which has barb x as required:

$$\begin{aligned} Q &\equiv (\nu \tilde{w}{:}\tilde{U})\, (\nu y : \updownarrow^1[\tilde{T}])\, (y![\tilde{v}] \mid R_1 \mid \cdots \mid R_n \mid y?[\tilde{z}].\, x![\tilde{z}]) \\ &\longrightarrow (\nu \tilde{w}{:}\tilde{U})\, (\nu y : \emptyset^1[\tilde{T}])\, (R_1 \mid \cdots \mid R_n \mid x![\tilde{v}]) \end{aligned}$$

We now just need to consider the possible reductions of pairs in $\mathcal{R}$.

(1) $[x/y]P' \xrightarrow{\alpha} R$ where $\alpha \neq x$. It must be the case that there exists an R' such that $P' \longrightarrow R'$ and $R = [x/y]R'$. This implies that $Q \longrightarrow (\nu y : \updownarrow^1[\tilde{T}])\,(R' \,|\, y?[\tilde{z}].\,x![\tilde{z}])$, as required.

(2) $[x/y]P' \xrightarrow{x} R$. Since P' is well typed under a remnant of $\Delta, y : !^1[\tilde{T}]$, it must be the case that

$$\begin{aligned} P' &\equiv (\nu\tilde{w}{:}\tilde{W})\,(R_1 \,|\, \cdots \,|\, R_n \,|\, y![\tilde{u}] \,|\, x?[\tilde{z}].\,O) \\ R &\equiv [x/y](\nu\tilde{w}{:}\tilde{W})\,(R_1 \,|\, \cdots \,|\, R_n \,|\, [\tilde{u}/\tilde{z}]O). \end{aligned}$$

The reduction is matched by

$$\begin{aligned} Q \;&\equiv\; (\nu y : \updownarrow^1[\tilde{T}])\,((\nu\tilde{w}{:}\tilde{U})\,(R_1 \,|\, \cdots \,|\, R_n \,|\, y![\tilde{u}] \,|\, x?[\tilde{z}].\,O) \,|\, y?[\tilde{z}].\,x![\tilde{z}]) \\ &\longrightarrow\; (\nu y : \emptyset^1[\tilde{T}])\,(\nu\tilde{w}{:}\tilde{U})\,(R_1 \,|\, \cdots \,|\, R_n \,|\, x![\tilde{u}] \,|\, x?[\tilde{z}].\,O) \\ &\longrightarrow\; (\nu y : \emptyset^1[\tilde{T}])\,(\nu\tilde{w}{:}\tilde{U})\,(R_1 \,|\, \cdots \,|\, R_n \,|\, [\tilde{u}/\tilde{z}]O), \end{aligned}$$

if we can prove that

$$\begin{aligned} &[x/y](\nu\tilde{w}{:}\tilde{U})\,(R_1 \,|\, \cdots \,|\, R_n \,|\, [\tilde{u}/\tilde{z}]O) \\ \stackrel{\bullet}{\approx}_\Gamma\;& (\nu y : \emptyset^1[\tilde{T}])\,(\nu\tilde{w}{:}\tilde{U})\,(R_1 \,|\, \cdots \,|\, R_n \,|\, [\tilde{u}/\tilde{z}]O). \end{aligned}$$

Let $Q' = (\nu\tilde{w}{:}\tilde{U})\,(R_1 \,|\, \cdots \,|\, R_n \,|\, [\tilde{u}/\tilde{z}]O)$. Since $([x/y]P', Q) \in \mathcal{R}$, we can use Theorem 4.3.1 and the fact that $Q \longrightarrow^* (\nu y : \emptyset^1[\tilde{T}])\,Q'$ to prove that there exists a Φ such that $\Phi \vdash (\nu y : \emptyset^1[\tilde{T}])\,Q'$ and such that Φ is a remnant of Γ. By using Lemma 6.1, we obtain $[x/y]Q' \stackrel{\bullet}{\approx}_\Phi (\nu y : \emptyset^1[\tilde{T}])\,Q'$. Lemma 5.17 proves that $[x/y]Q' \stackrel{\bullet}{\approx}_\Gamma (\nu y : \emptyset^1[\tilde{T}])\,Q'$ as required.

(3) $Q \longrightarrow (\nu y : \updownarrow^1[\tilde{T}])\,(P'' \,|\, y?[\tilde{z}].\,x![\tilde{z}])$ where $P' \longrightarrow P''$. This is matched by $[x/y]P' \longrightarrow [x/y]P''$, as required.

(4) $Q \longrightarrow (\nu y : \emptyset^1[\tilde{T}])\,(\nu\tilde{w}{:}\tilde{W})\,(R \,|\, x![\tilde{u}])$. It must be the case that $P' \equiv (\nu\tilde{w}{:}\tilde{W})\,(R \,|\, y![\tilde{u}])$ and $\Theta \vdash (\nu y : \emptyset^1[\tilde{T}])\,(\nu\tilde{w}{:}\tilde{W})\,(R \,|\, x![\tilde{u}])$ for some $\Theta \in \mathit{Rems}(\Gamma)$. This implies that the reduction of Q can be matched by zero reductions in $[x/y]P'$, since Lemma 6.1 proves that

$$\begin{aligned} (\nu y : \emptyset^1[\tilde{T}])\,(\nu\tilde{w}{:}\tilde{W})\,(R \,|\, x![\tilde{u}]) \;&\stackrel{\bullet}{\approx}_\Theta\; [x/y](\nu\tilde{w}{:}\tilde{W})\,(R \,|\, x![\tilde{u}]) \\ &=\; [x/y]P'. \end{aligned}$$

Lemma 5.17 lets us conclude that

$$(\nu y : \emptyset^1[\tilde{T}])\,(\nu\tilde{w}{:}\tilde{W})\,(R \,|\, x![\tilde{u}]) \stackrel{\bullet}{\approx}_\Gamma [x/y]P'$$

as required.

We therefore have that $\mathcal{R}$ is a weak Γ-bisimulation up to $\equiv$ as required. □

THEOREM 6.3. *If $\Delta, y : !^1[\tilde{T}] \vdash P$ and $\Delta + x : !^1[\tilde{T}]$ is well defined then*

$$[x/y]P \approx_{\Delta + x:!^1[\tilde{T}]} (\nu y : \updownarrow^1[\tilde{T}])\,(P \,|\, y?[\tilde{z}].\,x![\tilde{z}]).$$

PROOF. Suppose that σ is a $(\Delta + x : !^1[\tilde{T}])$-$\Gamma$ substitution, $\Theta \vdash R$, and that $\Theta + \Gamma$ is well defined. We must show

$$R \,|\, \sigma[x/y]P \stackrel{\bullet}{\approx}_{\Theta+\Gamma} R \,|\, \sigma(\nu y : \updownarrow^1[\tilde{T}])\,(P \,|\, y?[\tilde{z}].\,x![\tilde{z}]).$$

By Lemma 3.5, we know that there exists a Γ' such that σ is also a Δ-Γ' substitution and $\Gamma = \Gamma' + \sigma x{:}!^1[\tilde{T}]$. Let $\sigma' = \sigma[y/y]$. Then σ' is a $(\Delta, y : !^1[\tilde{T}])$-$(\Gamma', y : !^1[\tilde{T}])$

substitution. By using Lemma 3.6 and rule E-PAR, we have $\Theta + \Gamma', y : !^1[\tilde{T}] \vdash R \,|\, \sigma' P$. By using Lemma 6.2, we obtain

$$[\sigma x/y](R \,|\, \sigma' P) \stackrel{\bullet}{\approx}_{\Theta+\Gamma} (\nu y : \updownarrow^1[\tilde{T}])\, (R \,|\, \sigma' P \,|\, y?[\tilde{z}].\, \sigma x![\tilde{z}]).$$

The required result follows from

$$\begin{array}{l} [\sigma x/y](R \,|\, \sigma' P) = R \,|\, \sigma[x/y]P \\ (\nu y : \updownarrow^1[\tilde{T}])\, (R \,|\, \sigma' P \,|\, y?[\tilde{z}].\, \sigma x![\tilde{z}]) = R \,|\, \sigma(\nu y : \updownarrow^1[\tilde{T}])\, (P \,|\, y?[\tilde{z}].\, x![\tilde{z}]). \quad \Box \end{array}$$

Example 6.4. If $\Gamma = \mathit{plustwo} : ?^\omega[\mathit{Num}, !^1[\mathit{Num}]], \mathit{plusone} : !^\omega[\mathit{Num}, !^1[\mathit{Num}]]$ then

$$\begin{array}{ll} & \mathit{plustwo}?^*[j,s].\, (\nu r_1, r_2 : \updownarrow^1[\mathit{Num}]) \\ & \quad (\mathit{plusone}![j, r_1] \,|\, r_1?[k].\, (\mathit{plusone}![k, r_2] \,|\, r_2?[l].\, s![l])) \\ \approx_\Gamma & \mathit{plustwo}?^*[j,s].\, (\nu r_1 : \updownarrow^1[\mathit{Num}])\, (\mathit{plusone}![j, r_1] \,|\, r_1?[k].\, \mathit{plusone}![k, s]). \end{array}$$

PROOF. Let $\Delta = \mathit{plustwo} : ?^\omega[\mathit{Num}, !^1[\mathit{Num}]], \mathit{plusone} : !^\omega[\mathit{Num}, !^1[\mathit{Num}]], k : \mathit{Num}$ and $P = \mathit{plusone}![k, r_2]$. Since $\Delta, r_2 : !^1[\mathit{Num}] \vdash P$, we obtain

$$(\nu r_2 : \updownarrow^1[\mathit{Num}])\, (\mathit{plusone}![k, r_2] \,|\, r_2?[l].\, s![l]) \approx_{\Delta, s:!^1[\mathit{Num}]} \mathit{plusone}![k, s]$$

by using Theorem 6.3. The result follows, since $\approx_{\Delta, s:!^1[\mathit{Num}]}$ is a congruence relation (Theorem 5.25). $\Box$

7. APPLICATIONS

In concurrent programming languages such as Pict [Pierce and Turner 1995; 1997], where channel communications are the basis of all computation, it is important to have an efficient implementation of channel-based communication. In Pict, communications on linear channels are particularly important, since they account for up to 50% of communications in a typical program (channels in Pict programs are predominantly used to implement function or method calls, and in both cases the channel used to carry the result value can be assigned a linear type).

There are two ways that we can exploit linear type information during compilation. Firstly, linear types enable tail-call optimization (Theorem 6.3 proves that tail-call optimization is valid for linear channels, and Example 6.4 illustrates a simple instance of tail-call optimization). Tail-call optimization is important for languages like Pict, since it allows loop-like control constructs (such as `while` or `for` loops) to be efficiently implemented in terms of recursively defined processes. Secondly, linear type information can be used to enable specialized implementations of channel-based communication, as we explain below.

7.1 Implementing Channels

Ordinary pi-calculus channels can be represented using a pair of values: a counter (we use positive numbers to count blocked readers and negative numbers to count blocked writers) and a FIFO queue (which contains a queue of either blocked readers or blocked writers). A process attempting to write a value to a channel proceeds as follows (reading from a channel is similar):

(1) If the channel counter is less than or equal to zero, then there are no blocked readers available to communicate with, so we simply place the current process

in the FIFO queue and decrement the channel counter to indicate that it now contains one more blocked writer.

(2) Otherwise, since the channel counter is strictly greater than zero, there is at least one blocked reader available for communication. We remove the first reader from the FIFO queue, place it on the run queue, and decrement the channel counter. The current process then continues executing.

One thing that is immediately apparent about the above implementation is that the code is quite complex. Even though we can determine the current state of a channel quite efficiently (by examining the channel's counter), we nevertheless have quite a lot of work to do to add and remove processes from the channel's FIFO queue. It would certainly not be practical to include a copy of the above code (including the FIFO code) at every communication point in the program.

There are ways in which the above implementation can be improved, but all share the problem that they need to implement a data structure to hold blocked processes in FIFO order. A FIFO ordering is necessary in order to ensure fair progress of processes competing to use a shared channel, where by "fair progress" we mean that if some process is waiting to write (resp., read) on a channel from which infinitely many processes try to read (write), then eventually the writer will succeed in communicating with some reader. (This simple definition of fairness suffices in our present choice-free setting. In the presence of choice, stronger forms of fairness may be desired, but these are much more expensive to implement.)

7.2 Linear Channels

The primary advantage of linear channels is that they can never contain more than one blocked process, which means that neither a FIFO queue nor a counter is required. We can implement linear channels using just a reference cell (the cell contains a `NULL` value, if the channel is empty, or a pointer to a single blocked reader or writer process). The procedure for writing to a linear channel is therefore much simpler than before (reading from a linear channel is similar):

(1) If the channel contains `NULL`, then there is no blocked reader available to communicate with, so we simply update the channel to contain a pointer to the current process.

(2) Otherwise, the channel must contain a pointer to a blocked reader (since linear channels can never have more than one writer). We remove the reader from the channel and place it on the run queue. The current process then continues executing.

The above code is sufficiently compact for us to include it in-line at each linear communication point. In addition, the cost of allocating a new linear channel is significantly lower, since linear channels are less than half the size of ordinary channels (the exact proportion depends on the cost of allocating an empty FIFO queue).

7.3 Replicated Channels

Replicated channels are an important variant of linear channels, which can be used arbitrarily often for output but which only allow a single, replicated, input. Communication on replicated channels enjoys the same partial confluence property as linear communication. To handle replicated channels formally, we introduce a new multiplicity $*$ (either replacing or adding to the linear multiplicity 1). We define the operator $+$ for the new multiplicity as follows

$$p^*[\tilde{T}] + q^*[\tilde{T}] = (p \cup q)^*[\tilde{T}] \quad \text{if } i \notin p \cap q$$

and modify the side conditions of the rules E-In and E-RIn as follows:

$$\frac{\Gamma, \tilde{z}{:}\tilde{T} \vdash P \qquad m \neq *}{\Gamma + x : ?^m[\tilde{T}] \vdash x?[\tilde{z}].\, P} \tag{E-In}$$

$$\frac{\Gamma, \tilde{z}{:}\tilde{T} \vdash P \qquad \Gamma \text{ unlimited} \qquad m = * \text{ or } \omega}{\Gamma + x : ?^m[\tilde{T}] \vdash x?^*[\tilde{z}].\, P} \tag{E-RIn}$$

The idea of replicated channels has been applied by Sangiorgi [1999] in a proof of correctness of a useful program transformation in a concurrent object-oriented setting. A similar idea is discussed by Honda [1996].

A variant of replicated channels has been implemented in the Pict language, where the input capabilities associated with replicated channels are checked using a simple, syntactic version of the above typing rules. The simplified typing rules are less permissive than the rules described above (and therefore require some channels to be assigned multiplicity ω, whereas the above rules could assign the channel a multiplicity $*$). However, they ensure a useful extra invariant: every output on a replicated channel will find a replicated input expression available in the channel.

Replicated channels are important in Pict because they are used extensively in the encoding of function calls. Since the status of a replicated channel can be determined at compilation time, replicated channels can be compiled even more efficiently than linear channels (there is no need for a conditional expression to test the current status of a replicated channel).

8. RELATED WORK

A line of work by Honda [1993; 1996; 1998] adopts a type-theoretic perspective related to ours. Indeed, the type system we have studied here can be viewed as a particular example of his notion of *typed Milner algebra.*

Our type system also shares a good deal with linear type systems for lambda-calculi and related programming languages [Abramsky 1993; Baker 1992; Mackie 1994; Turner et al. 1995; Wadler 1991]. In particular, linear typing has recently been proposed as a framework for *syntactic control of interference* in sequential programming languages (cf. [O'Hearn et al. 1995]).

In the world of process calculi, there have been several articles on analyzing channel usage using *effect systems.* Nielson and Nielson [1995] proposed a method for inferring the maximum usages of channels by applying the effect system for CML [Nielson and Nielson 1994], while Kobayashi et al. [1995] proposed a method for approximating the number of receivers that can ever try to read from a channel simultaneously, which in particular can be used to detect channels whose maximum queue length is 1. Our type-based analysis and these effect-based analyses both seem to have advantages and disadvantages. In approaches based on effect analysis, channel usage is analyzed with respect to "regions," which may be aliased to an infinite number of channels, degrading the accuracy of the analysis. Although Kobayashi et al. tried to overcome this defect to some degree, the resulting analysis is difficult, and some algorithmic aspects (especially time complexity) are left open.

Abramsky et al. [1994] and Gay and Nagarajan [1995] describe a typed calculus of synchronous processes. Although their calculus ensures deadlock-freedom, its expressive power is much more limited than ours in the sense that there must be exactly one sender and receiver for each channel, and mobility (passing channels as data along channels) cannot be expressed.

A type system reminiscent of ours—though not explicitly incorporating the notion of linearity—has been developed by Steffen and Nestmann [1995] for the purpose of analyzing confluence in pi-calculus processes arising in the semantics of concurrent object-oriented programs. (The same problem has been tackled by Liu and Walker using purely semantic techniques [Liu and Walker 1995a].) Our analysis is also related to the one used by Niehren [1996] to guarantee "uniform confluence" in a pi-calculus fragment with only replicated inputs.

Effects similar to our linear typing have also been achieved by more syntactic means, as in Honda and Yoshida's notion of "beta-reduction" for concurrent combinators [Honda and Yoshida 1994]. A more recent article by Takeuchi et al. [1994] describes a modified pi-calculus whose syntax guarantees that certain channels are shared between just two processes, achieving a restricted form of linearity by much more elementary means.

In the present framework, this kind of interaction can be viewed as a generalization from linear to *linearized* channels, which can be used multiple times but only in a sequential manner: an output $x![\tilde{y}].P$ can reuse x again for output in P, while an input $x?[\tilde{y}].P$ can reuse x for input in P. (Linearized channel types can be viewed as a variant of a fragment of Honda's "types for dyadic interaction" [Honda 1993].) Combining the intuitions behind replicated and linearized channel types yields another important usage analysis, which Milner calls "unique handling" [Milner 1991]: a channel is uniquely handled if it has a single replicated receiver and, at any given moment, at most one sender. That is, the capability to send must be explicitly passed from one "client" to another, so there can never be any contention between clients for access to the service provided by the receiver.

Kobayashi's type system for partial deadlock-freedom [Kobayashi 1998] is based on the linear type system presented here: he added dependency among communications on linear (and other special) channels to type information and showed that certain communications do not cause deadlock. The typed process equivalence theory developed here should be applicable to his type system.

Igarashi and Kobayashi [1997] have recently developed a type inference algorithm for an affine variant of our type system which can recover linear type information from unannotated process terms. Work is also underway on evaluating a concurrent linear logic programming language *HACL* [Kobayashi and Yonezawa 1995a], whose core language is close to the pi-calculus.

ACKNOWLEDGEMENTS

We profited from lively discussions with Robin Milner, Phil Wadler, and the Monday "Interruption Club" at Cambridge. Cédric Fournet gave us useful comments on an earlier draft. Kohei Honda and Nobuko Yoshida assisted us with some points of history. The careful readings of the anonymous referees led to many improvements in both presentation and technicalities.

REFERENCES

ABRAMSKY, S. 1993. Computational interpretations of linear logic. *Theor. Comput. Sci. 111,* 1–2 (Apr. 12), 3–57.

ABRAMSKY, S., GAY, S. J., AND NAGARAJAN, R. 1994. Interaction categories and the foundations of typed concurrent programming. In *Deductive Program Design: Proceedings of the 1994 Marktoberdorf Summer School, NATO ASI Series F.* Springer-Verlag, Berlin.

AGHA, G. A. 1986. *Actors: a Model of Concurrent Computation in Distributed Systems.* The MIT Press, Cambridge, MA.

AMADIO, R. M., CASTELLANI, I., AND SANGIORGI, D. 1996. On bisimulations for the asynchronous pi-calculus. In *Seventh International Conference on Concurrency Theory (CONCUR '96).* Lecture Notes in Computer Science, vol. 1119. Springer-Verlag, Berlin.

BAKER, H. G. 1992. Lively linear lisp – look ma, no garbage! *ACM Sigplan Notices 27,* 8, 89–98.

CARDELLI, L. 1986. Amber. In *Combinators and Functional Programming Languages,* G. Cousineau, P.-L. Curien, and B. Robinet, Eds. Lecture Notes in Computer Science, vol. 242. Springer-Verlag, Berlin, 21–47.

FOURNET, C. AND GONTHIER, G. 1996. The reflexive chemical abstract machine and the join-calculus. In *Conference Record of POPL '96: the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* 372–385.

FOURNET, C., GONTHIER, G., LÉVY, J.-J., MARANGET, L., AND RÉMY, D. 1996. A calculus of mobile agents. In *7th International Conference on Concurrency Theory (CONCUR'96).* Lecture Notes in Computer Science, vol. 1119. Springer-Verlag, Berlin, 406–421.

GAY, S. AND NAGARAJAN, R. 1995. A typed calculus of synchronous processes. In *Proceedings of IEEE Symposium on Logic in Computer Science.* IEEE, New York, 210–220.

GAY, S. J. 1993. A sort inference algorithm for the polyadic π-calculus. In *Proceedings of the 20th ACM Symposium on Principles of Programming Languages.* 429–438.

GIACALONE, A., MISHRA, P., AND PRASAD, S. 1989. Facile: A Symmetric Integration of Concurrent and Functional Programming. *Int. J. Parallel Program. 18,* 2, 121–160.

GIRARD, J.-Y. 1987. Linear logic. *Theor. Comput. Sci. 50*, 1–102.

GIRARD, J.-Y., LAFONT, Y., AND TAYLOR, P. 1989. *Proofs and Types.* Cambridge Tracts in Theoretical Computer Science, vol. 7. Cambridge University Press, Cambridge.

HEWITT, C. 1977. Viewing control structures as patterns of passing messages. *Artificial Intelligence 8*, 323–364.

HONDA, K. 1993. Types for dyadic interaction. In *CONCUR'93.* Lecture Notes in Computer Science, vol. 715. Springer-Verlag, Berlin, 509–523.

HONDA, K. 1996. Composing processes. In *Conference Record of POPL '96: the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* 344–357.

HONDA, K. 1998. A theory of types for pi-calculus. Manuscript; available through `http://www.dcs.qmw.ac.uk/~kohei`.

HONDA, K. AND TOKORO, M. 1991. An object calculus for asynchronous communication. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, P. America, Ed. Lecture Notes in Computer Science, vol. 512. Springer-Verlag, Berlin, 133–147.

HONDA, K. AND YOSHIDA, N. 1994. Combinatory representation of mobile processes. In *Proceedings of POPL '94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon.* 348–360.

IGARASHI, A. AND KOBAYASHI, N. 1997. Type-based analysis of usage of communication channels for concurrent programming languages. In *Proceedings of International Static Analysis Symposium (SAS'97)*. Lecture Notes in Computer Science, vol. 1302. Springer-Verlag, Berlin, 187–201.

JONES, C. B. 1993. A pi-calculus semantics for an object-based design notation. In *Proceedings of CONCUR'93*, E. Best, Ed. Lecture Notes in Computer Science, vol. 715. Springer-Verlag, Berlin, 158–172.

KOBAYASHI, N. 1998. A partially deadlock-free typed process calculus. *ACM Trans. Program. Lang. Syst. 20,* 2, 436–482.

KOBAYASHI, N., NAKADE, M., AND YONEZAWA, A. 1995. Static analysis of communication for asynchronous concurrent programming languages. In *Second International Static Analysis Symposium (SAS'95)*. Lecture Notes in Computer Science, vol. 983. Springer-Verlag, Berlin, 225–242.

KOBAYASHI, N. AND YONEZAWA, A. 1994. Type-theoretic foundations for concurrent object-oriented programming. In *Proceedings of ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'94)*. 31–45.

KOBAYASHI, N. AND YONEZAWA, A. 1995a. Higher-order concurrent linear logic programming. In *Theory and Practice of Parallel Programming*. Lecture Notes in Computer Science, vol. 907. Springer-Verlag, Berlin, 137–166.

KOBAYASHI, N. AND YONEZAWA, A. 1995b. Towards foundations for concurrent object-oriented programming—types and language design. *Theory Pract. Object Syst. 1,* 4, 243–268.

LIU, X. AND WALKER, D. 1995a. Confluence of processes and systems of objects. In *Proceedings of TAPSOFT'95*. Lecture Notes in Computer Science, vol. 915. Springer-Verlag, Berlin, 217–231.

LIU, X. AND WALKER, D. 1995b. A polymorphic type system for the polyadic π-calculus. In *CONCUR'95: Concurrency Theory*. Lecture Notes in Computer Science, vol. 962. Springer-Verlag, Berlin, 103–116.

MACKIE, I. 1994. Lilac: A functional programming language based on linear logic. *J. Funct. Program. 4,* 4 (Oct.), 395–433.

MILNER, R. 1989. *Communication and Concurrency*. Prentice Hall.

MILNER, R. 1990. Functions as processes. Research Report 1154, INRIA, Sofia Antipolis. Final version in *Math. Struct. Comput. Sci.* 2(2):119–141, 1992.

MILNER, R. 1991. The polyadic π-calculus: a tutorial. Tech. Rep. ECS–LFCS–91–180, Laboratory for Foundations of Computer Science, Dept. of Computer Science, Univ. of Edinburgh, UK. Oct. Appeared in *Proceedings of the International Summer School on Logic and Algebra of Specification*, Marktoberdorf, August 1991. Reprinted in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer, and H. Schwichtenberg, Springer-Verlag, 1993.

MILNER, R., PARROW, J., AND WALKER, D. 1992. A calculus of mobile processes (Parts I and II). *Inf. Comput. 100*, 1–77.

MILNER, R. AND SANGIORGI, D. 1992. Barbed bisimulation. In *19th ICALP*, W. Kuich, Ed. Lecture Notes in Computer Science, vol. 623. Springer-Verlag, Berlin, 685–695.

NIEHREN, J. 1996. Functional computation as concurrent computation. In *Conference Record of POPL '96: the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 333–343.

NIELSON, H. R. AND NIELSON, F. 1994. Higher-order concurrent programs with finite communication topology. In *Proceedings of POPL '94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon*. 84–97.

NIELSON, H. R. AND NIELSON, F. 1995. Static and dynamic processor allocation for higher-order concurrent languages. In *TAPSOFT'95: Theory and Practice of Software Development*. Lecture Notes in Computer Science, vol. 915. Springer-Verlag, Berlin, 590–604.

ODERSKY, M. 1995. Polarized name passing. In *Proc. FST & TCS*. Lecture Notes in Computer Science, vol. 1026. Springer-Verlag, Berlin, 324–337.

O'HEARN, P. W., TAKEYAMA, M., POWER, A. J., AND TENNENT, R. D. 1995. Syntactic control of interference revisited. In *MFPS XI, conference on Mathematical Foundations of Program Se-*

mantics. Electronic Notes in Theoretical Computer Science, vol. 1. Elsevier Science Publishers, Amsterdam.

Pierce, B. and Sangiorgi, D. 1993. Typing and subtyping for mobile processes. In *Logic in Computer Science.* 376–385. Full version in *Math. Struct. Comput. Sci.*, Vol. 6, No. 5, 1996.

Pierce, B. C. and Turner, D. N. 1995. Concurrent objects in a process calculus. In *Theory and Practice of Parallel Programming (TPPP), Sendai, Japan (Nov. 1994)*, T. Ito and A. Yonezawa, Eds. Number 907 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, 187–215.

Pierce, B. C. and Turner, D. N. 1997. Pict: A programming language based on the pi-calculus. Tech. Rep. CSCI 476, Computer Science Department, Indiana Univ. To appear in *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, MIT Press, 1998.

Reppy, J. 1991. CML: A higher-order concurrent language. In *Programming Language Design and Implementation.* SIGPLAN, ACM, New York, 293–259.

Sangiorgi, D. 1992. Expressing mobility in process algebras: First-order and higher-order paradigms. Ph.D. thesis, Dept. of Computer Science, Univ. of Edinburgh.

Sangiorgi, D. 1993. An investigation into functions as processes. In *Proc. Ninth International Conference on the Mathematical Foundations of Programming Semantics (MFPS'93).* Lecture Notes in Computer Science, vol. 802. Springer-Verlag, Berlin, 143–159.

Sangiorgi, D. 1994. The lazy lambda calculus in a concurrency scenario. *Inf. Comput. 111,* 1, 120–153.

Sangiorgi, D. 1999. Typed π-calculus at work: A correctness proof of Jones's parallelisation transformation on concurrent objects. *Theory and Practice of Object Systems 5,* 1, 25–33. Earlier version in proceedings of *Foundations of Object Oriented Languages, 1997.*

Sangiorgi, D. and Milner, R. 1992. The problem of "Weak Bisimulation up to". In *Proceedings of CONCUR '92*, W. Cleveland, Ed. Vol. 630. Springer-Verlag, Berlin, 32–46.

Steffen, M. and Nestmann, U. 1995. Typing confluence. Interner Bericht IMMD7-xx/95, Informatik VII, Universität Erlangen-Nürnberg.

Takeuchi, K., Honda, K., and Kubo, M. 1994. An interaction-based language and its typing system. In *Proceedings of PARLE'94.* Lecture Notes in Computer Science, vol. 817. Springer-Verlag, Berlin, 398–413.

Turner, D. N. 1995. The polymorphic pi-calulus: Theory and implementation. Ph.D. thesis, Univ. of Edinburgh.

Turner, D. N., Wadler, P., and Mossin, C. 1995. Once upon a type. In *Functional Programming Languages and Computer Architecture.* San Diego, California, 1–11.

Vasconcelos, V. T. 1994. Typed concurrent objects. In *Proceedings of the 8th European Conference on Object-Oriented Programming (ECOOP).* Lecture Notes in Computer Science, vol. 821. Springer-Verlag, Berlin, 100–117.

Vasconcelos, V. T. and Honda, K. 1993. Principal typing schemes in a polyadic pi-calculus. In *Proceedings of CONCUR '93.* Lecture Notes in Computer Science, vol. 715. Springer-Verlag, Berlin, 524–538. Also available as Keio Univ. Report CS-92-004.

Wadler, P. 1991. Is there a use for linear logic? In *Proceedings of ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation.* ACM, New York, 255–273.

Walker, D. 1995. Objects in the π-calculus. *Inf. Comput. 116*, 253–271.

Received May 1998; revised December 1998; accepted April 1999