The π-calculus as a theory in linear logic: Preliminary results *

Dale Miller

Computer Science Department
University of Pennsylvania
Philadelphia, PA 19104-6389 USA
dale@cis.upenn.edu

October 29, 1992

Abstract

The agent expressions of the π-calculus can be translated into a theory of linear logic in such a way that the reflective and transitive closure of π-calculus (unlabeled) reduction is identified with "entailed-by". Under this translation, parallel composition is mapped to the multiplicative disjunct ("par") and restriction is mapped to universal quantification. Prefixing, non-deterministic choice (+), replication (!), and the match guard are all represented using non-logical constants, which are specified using a simple form of axiom, called here a *process clause*. These process clauses resemble Horn clauses except that they may have multiple conclusions; that is, their heads may be the par of atomic formulas. Such multiple conclusion clauses are used to axiomatize communications among agents. Given this translation, it is nature to ask to what extent proof theory can be used to understand the meta-theory of the π-calculus. We present some preliminary results along this line for π_0, the "propositional" fragment of the π-calculus, which lacks restriction and value passing (π_0 is a subset of CCS). Using ideas from proof-theory, we introduce *co-agents* and show that they can specify some testing equivalences for π_0. If negation-as-failure-to-prove is permitted as a co-agent combinator, then testing equivalence based on co-agents yields observational equivalence for π_0. This latter result follows from observing that co-agents directly represent formulas in the Hennessy-Milner modal logic.

1 Introduction

In this paper we address the question "Can we view a given process calculus as a logic?" This is different (although certainly related) to the question "Can logic be used to characterize a given process calculus?" Such a question would view logic as an auxiliary language to that of the process calculus: for example, the Hennessy-Milner logic has such a relationship to CCS. Our approach here will be to use logic more immediately by trying to match combinators of the given process calculus directly to logical connectives and, if a combinator fails to match, trying to axiomatized it directly and uniformly in logic.

For our purposes here, we shall consider a formal system to be a logic if it has a sequent calculus presentation that admits a cut-elimination theorem. Of course, this definition of logic is not formal unless formal definitions of sequent calculi and cut-elimination are provided. We shall not attempt

*This paper appears in the Proceedings of the 1992 Workshop on Extensions to Logic Programming, edited by E. Lamma and P. Mello, Lecture Notes in Computer Science, Springer-Verlag.

formal definitions of these two terms here: we simply make use of a couple examples of sequent calculus systems. A constant of the formal system will be considered *logical* if it has left and right introduction rules. A *non-logical* constant is any other constant whose meaning is specified by axioms or theories: such constants do not, in general, participate in a cut-elimination theorem.

There seems to be two broad ways in which connections between concurrency and proof theory can be and are being developed: one uses proof reduction and the other proof search.

The functional programming approach. Functional programs can be viewed as natural deduction proofs and computation on them as the process of proof normalization. Using familiar correspondences between natural deduction proofs and normalization with sequent calculus and cut-elimination in intuitionistic logic [Pot77, Fel91], functional programs can be seen as sequent proofs and computation as cut-elimination. Traditionally, the sequents used are of the form $\Delta \longrightarrow G$, where Δ is a set of propositions (generally typing judgments) and G is a single proposition. Such sequents are called *single-conclusion* sequents.

Following ideas of Girard presented in [Gir87], Abramsky [Abr90, Abr91] has extended this interpretation of computation to multiple-conclusion sequents, that is, sequents of the form $\Delta \longrightarrow \Gamma$, where Δ and Γ are both sets (actually, multisets) of propositions. In this setting, cut-elimination specifies concurrent programming. In particular, Abramsky presents a method for "realizing" the computational content of multiple-conclusion proofs in linear logic that yields concurrent programs in CCS, CSP, and the π-calculus. In these realized programs, cut-elimination in proofs is modeled by communication.

The logic programming approach. In the logic programming setting, programs are theories (collections of formulas) describing the meaning of non-logical constants and computation is identified with the search for cut-free sequent proofs. Here, the sequent $\Sigma; \Delta \longrightarrow G$ is used to represent the state of an idealized logic programming interpreter in which the current set of non-logical constants is Σ, the current logic program (theory) about those constants is the set of formulas Δ and the formula to be established, called the query or goal, is G.

A logic and proof system will be consider a logic programming language if a simple kind of *goal-directed* search is complete. This kind of definition of logic programming was first given in [MNPS91] for single-conclusion sequents, where the technical notion of *uniform proof* provides an analysis of goal-directed search. A uniform proof is a cut-free, single-conclusion sequent proof where every sequent whose right-hand side is non-atomic is the conclusion of a right-introduction rule. In an interpreter attempting to find a uniform proof, the structure of the right-hand side (the goal) can be reflected directly into the proof being constructed. The given logic and proof system is called an *abstract logic programming language* if a sequent has a proof if and only if it has a uniform proof. First-order and higher-order variants of Horn clauses and the more expressive hereditary Harrop formulas can be used as the basis of abstract logic programming languages [MNPS91].

In abstract logic programming languages, the search semantics of a logical connective in the goal is independent from its context (the program): contexts are only considered to help in proving atomic formulas. For example, if our logical system is intuitionistic logic, an attempt to prove the sequent $\Sigma; \Delta \longrightarrow G_1 \vee G_2$ could be replaced by a proof of either $\Sigma; \Delta \longrightarrow G_1$ or $\Sigma; \Delta \longrightarrow G_2$, no matter what formulas are contained in Δ. This is not a complete strategy for full intuitionistic logic: while there is a proof of the sequent $\Sigma; p \vee q \longrightarrow q \vee p$, its last inference rule is not $\vee$-R, that is, there are no proofs of $\Sigma; p \vee q \longrightarrow q$ or of $\Sigma; p \vee q \longrightarrow p$. When the syntax of programs are restricted adequately, completeness of uniform proofs can be established. The resulting restriction then determines a logic programming language. Within this setting, cut-elimination plays the meta-theoretic role of guarantor of canonical models for logic programs (see, for example, [Mil92, HM92]).

Unfortunately, the definition of uniform proofs given here is restricted to only single-conclusion sequent proofs systems. Extending this notion of goal-directed search to multiple conclusion sequents runs into the following simple problem: if the right-hand side of a sequent contains two or

more non-atomic formulas, how should the logical connectives at the head of those formulas be introduced? There seems to be two choices. One approach simply requires that one of the possible introductions be done. This has the disadvantage that there might be an interdependency between right-introduction rules in that one may need to appear lower is a proof than another. In this case, logical connectives in the goal would not reflect directly and simply into the structure of the proof. A second approach requires that all right-hand rules should be done simultaneously. Although it is difficult to deal with simultaneous rule application in the sequent calculus, we can employ permutations of inference rules within the sequent calculus [Kle52]. That is, we can require that if two or more right-introduction rules can be use to derive a given sequent, then all possible orders of applying those right-introduction rules can be obtained from any other order simply by permuting right-introduction inferences. Using this second approach, we shall say that a cut-free sequent proof Ξ is *uniform* if for every subproof Ψ of Ξ and for every non-atomic formula occurrence B in the right-hand side of the endsequent of Ψ, there is a proof Ψ' that is equal to Ψ up to permutation of inference rules and is such that the last inference rule in Ψ' introduces the top-level logical connective occurring in B. It is easy to see that this definition of uniform proof generalizes the one given above for single-conclusion sequents.

As we shall see , the π-calculus can be viewed as a multiple-conclusion logic programming language in the sense that certain sequents are provable if and only if they have multiple-conclusion uniform proofs.

Our analytic tools are taken from the sequent calculus, especially the refinement of that subject found in linear logic [Gir87], and from logic programming, particularly the topics of goal-directed provability and negation-as-failure. We shall investigate to what extent the framework of introduction rules, λ-abstraction in terms and in proofs (also know as eigen-variables), and the central notion of cut-elimination helps in analyzing a process calculus. This work is preliminary: we shall only look at the π-calculus [MPW89a, MPW89b, Mil91, MPW91] as a particular example of a process calculus. This calculus is, of course, rich and presents several interesting challenges.

2 Translating π-calculus expressions into logic

Besides assuming some familiarity with sequent calculus, we shall also assume that the reader is familiar with the π-calculus as given in either [Mil90] or [MPW89a]. The principle mechanism of the π-calculus is the synchronization of two agents and the sending of a name from one agent to another. Synchronization is familiar from CCS; value passing is new to the π-calculus. The expression $\bar{x}z.P$ describes an agent that is willing to transmit the value z on the wire x (x and z are names). The expression $x(y).Q$ denotes an agent that is willing to receive a value on wire x and formally bind that value to y. The bound variable y in this expression is scoped over Q. The central computational step of the π-calculus is the reduction of the parallel composition $\bar{x}z.P \mid x(y).Q$ to the expression $P \mid Q[z/y]$. The agents P and $Q[z/y]$ are now able to continue their interactions with their environment independently.

The π-calculus differs from CCS also in that it has a notion of scope restriction: in the agent expression $(x)P$, x is bound and invisible to the outside. The scoped value x, however, can be communicated outside its scope, providing a phenomenon known as "scope extrusion." For example, $(z)(\bar{x}z.P \mid Q) \mid x(y).R$ is structurally equivalent to $(z)(\bar{x}z.P \mid Q \mid x(y).R)$, provided that z is not free in $x(y).R$. This scope restriction is always easy to accommodate since we shall assume that α-conversion is available for changing the name of bound variables. This expression can now be reduced to $(z)(P \mid Q \mid R[z/y])$, where the scope of the restriction (z) is larger since it contains the agent $R[z/y]$ in which z may be free. This mechanism of generating new names (using α-conversion) and sending them outside their scope is an important part of the computational power of the π-calculus.

The silent transition τ is not discussed at all in this paper: although the techniques described below should be able to address τ, the appropriate methods for this have not yet been investigated.

Below we describe three translations of π-calculus agent expressions into logical expressions. The first two are simple duals of each other; the third is a simplification of the first.

The disjunctive translation. The first translation requires the logical constants $\oplus$ (additive disjunction), $\parr$ (par, multiplicative disjunction), ? (the exponential "why not"), $\forall$, and $\bot$ (the identity for $\parr$). Given its dependence on the additive and multiplicative disjunctions of linear logic, this translation is called the disjunctive translation. The following three simply typed, non-logical constants are also required (assuming that the type of logical expressions is o and that of names is i):

$$\mathtt{send} : i \to i \to o \to o, \quad \mathtt{get} : i \to (i \to o) \to o, \quad \mathtt{match} : i \to i \to o \to o.$$

As should be clear from these types, we shall freely make use of higher-order types and λ-calculus to smooth the treatment of bound variables and variable scoping. All those details will be pressed into a simple meta-level that contains the simply typed λ-calculus and quantification at higher-order types.

The disjunctive translation is given by the following induction on the structure of agent expressions.

$$\langle\!\langle P + Q \rangle\!\rangle = \langle\!\langle P \rangle\!\rangle \oplus \langle\!\langle Q \rangle\!\rangle \qquad \langle\!\langle P \mid Q \rangle\!\rangle = \langle\!\langle P \rangle\!\rangle \parr \langle\!\langle Q \rangle\!\rangle$$

$$\langle\!\langle (x)P \rangle\!\rangle = \forall x \langle\!\langle P \rangle\!\rangle \qquad \langle\!\langle ! P \rangle\!\rangle = ? \langle\!\langle P \rangle\!\rangle \qquad \langle\!\langle nil \rangle\!\rangle = \bot$$

$$\langle\!\langle \bar{x}y.P \rangle\!\rangle = \mathtt{send}\ x\ y\ \langle\!\langle P \rangle\!\rangle \qquad \langle\!\langle x(y).P \rangle\!\rangle = \mathtt{get}\ x\ \lambda y \langle\!\langle P \rangle\!\rangle$$

$$\langle\!\langle [x = y]P \rangle\!\rangle = \mathtt{match}\ x\ y\ \langle\!\langle P \rangle\!\rangle$$

To describe the meaning of the three non-logical constants, we have the following axioms.

$$\forall_i x \forall_i y \forall_o S \forall_{i \to o} R\ [Ry \parr S \multimap \mathtt{get}\ x\ R \parr \mathtt{send}\ x\ y\ S]$$
$$\forall_i x \forall_{i \to o} P\ [P \multimap \mathtt{match}\ x\ x\ P]$$

Notice that these axioms are higher-order in the sense that they allow quantification over predicate symbols. Such quantification is intended here to be purely syntactic: the type $i \to o$ denotes the set of closed, simply typed λ-terms of type $i \to o$ and not some abstract domain of functions. A similar treatment of higher-order type quantification for Horn clauses can be found in [NM90].

The conjunctive translation. It is trivial to dualize the disjunctive translation completely. That is, it is possible to map the "logical" combinators into the dual logical connectives.

$$\langle\!\langle P + Q \rangle\!\rangle = \langle\!\langle P \rangle\!\rangle \,\&\, \langle\!\langle Q \rangle\!\rangle \qquad \langle\!\langle P \mid Q \rangle\!\rangle = \langle\!\langle P \rangle\!\rangle \otimes \langle\!\langle Q \rangle\!\rangle$$

$$\langle\!\langle (x)P \rangle\!\rangle = \exists x \langle\!\langle P \rangle\!\rangle \qquad \langle\!\langle ! P \rangle\!\rangle = ! \langle\!\langle P \rangle\!\rangle \qquad \langle\!\langle nil \rangle\!\rangle = 1$$

In this case, the non-logical axioms would be axiomatized with the formulas

$$\forall_i x \forall_i y \forall_o S \forall_{i \to o} R\ [\mathtt{get}\ x\ R \otimes \mathtt{send}\ x\ y\ S \multimap Ry \otimes S]$$
$$\forall_i x \forall_o P\ [\mathtt{match}\ x\ x\ P \multimap P]$$

This translation is called *conjunctive* because it uses the multiplicative and additive conjunctions.

The formal analysis below is completely dualizable, so there appears to be no formal reason to pick one translation over the other. This seems to be the case because process calculus is fundamentally about reduction, while logic has made a commitment to both reduction (implies/implied-by) and to truth values. Truth values do not naturally map into processes. The disjunctive translation maps reduction to implied-by; the conjunctive translation to implies.

The following two extra-logical motivations can be offered for choosing the disjunctive translation over the conjunctive translation.

Goal reduction in logic programming and agent reduction in the π-calculus. In logic programming based on single-conclusion sequents, a uniform proof that results from the successful search for a proof of a sequent $\Sigma; \Delta \longrightarrow G$ records the goal reductions applied to G in the right-hand side of the proof's sequent when read from the bottom of the proof. If the disjunctive translation is used, a similar observation can be applied to the π-calculus: agent reduction is recorded in the right-hand sides of the sequents when read from the bottom. Andreoli and Pareschi [AP91] have made a similar choice in the representation of agent reduction using a kind of multiple-conclusion Horn clause. The conjunctive translation estranges this parallel since reductions would take place on the left-hand side.

Scope extrusion as a multiple-conclusion phenomenon. A natural notion of scoping occurs in logic programming based on single-conclusion sequents. For example, the search for a uniform proof of the sequent $\Sigma; \Delta \longrightarrow D \supset G$ reduces to the search for a uniform proof of the sequent $\Sigma; \Delta, D \longrightarrow G$. If Δ is considered to be the current program held by a logic programming interpreter, then D can be seen as a program unit that is added to the current program during a computation. A notion of modular programming for logic programming was developed in [Mil89] based on this simple observation. To enforce that this notion of modular programming obeys the correct notion of scoping, single conclusion sequent calculus is required. Consider, for example, searching for a uniform proof of the sequent $\Sigma; \Delta \longrightarrow G_1 \vee (D \supset G_2)$ using the usual intuitionistic introduction rules for $\vee$-R and $\supset$-R [Gen69]. This search would lead to the search for proofs of either the sequent $\Sigma; \Delta \longrightarrow G_1$ or $\Sigma; \Delta, D \longrightarrow G_2$. In particular, the formula D is only available to help prove the formula G_2: its scope does not include G_1. This formula is, however, classically equivalent to $(D \supset G_1) \vee G_2$ and $D \supset (G_1 \vee G_2)$. Thus the scope of D can move in ways not supported in intuitionistic logic. In particular, $p \vee (p \supset q)$ is not provable intuitionistically but it is classically. Gentzen's characterization of the differences between intuitionistic and classical logics as arising from differences in using single and multiple conclusion sequents provides an elegant analysis of scope extrusion. Consider the following sequent proof.

$$\dfrac{\dfrac{\overline{p \longrightarrow p, q}}{\longrightarrow p, p \supset q}}{\longrightarrow p \vee (p \supset q)}$$

The occurrence of p in the left of the initial sequent has as its scope all the formulas on the right: in the intuitionistic case, there can only be one such formula on the right and, hence, scope cannot be liberalized in this way.

If the disjunctive translation is used, scope extrusion in the π-calculus can be accounted for in an analogous fashion. In this case, however, scope extrusion arises between the interaction of the $\forall$-R rule and multiple conclusions. For a simple example, consider the sequent $\Sigma; p \longrightarrow (\forall x_i.q) \vee (\exists y_i.p)$, where we assume that Σ has no constant whose type contains i. This sequent is provable only if we admit multiple conclusion sequents in its proof. Below is a proof of this sequent.

$$\dfrac{\dfrac{\dfrac{\overline{\Sigma, x : i; p \longrightarrow q, p}}{\Sigma, x : i; p \longrightarrow q, \exists y_i.p}}{\Sigma; p \longrightarrow \forall x_i.q, \exists y_i.p}}{\Sigma; p \longrightarrow (\forall x_i.q) \vee (\exists y_i.p)}$$

Here, it is an eigen-variable that has its scope liberated. As we shall see, scope extrusion in the π-calculus will be explained by this use of eigen-variables. In the conjunctive translation, similar proofs are possible but the correspondence to scope extrusion in logic programming would disappear and the distinctions between single-conclusion and multiple-conclusion sequents would not then be relevant.

P	Q	$\Sigma; P \vdash Q$	$\Sigma; Q \vdash P$	$\Sigma; P \dashv\vdash Q$
$P \mid P$	P	no	no	no
$P + P$	P	yes	yes	yes
$(x)P$	$P[y/x]^{\S}$	yes	no	no
$(x)P^{\dagger}$	P	yes‡	yes	yes‡
$P \mid !\,P$	$!\,P$	yes	no	no
$!\,P$	$!\,P \mid !\,P$	yes	yes	yes
$!\,P$	nil	no	yes	no
$P_1 \mid (P_2 \mid P_3)$	$(P_1 \mid P_2) \mid P_3$	yes	yes	yes
$P_1 + (P_2 + P_3)$	$(P_1 + P_2) + P_3$	yes	yes	yes
$P \mid nil$	P	yes	yes	yes
$P \mid Q$	$Q \mid P$	yes	yes	yes
$P + Q$	$Q + P$	yes	yes	yes
$(x)(y)P$	$(y)(x)P$	yes	yes	yes
$(x)(P \mid Q)^{\dagger}$	$P \mid (x)Q$	yes	yes	yes
$(x)(P + Q)^{\dagger}$	$P + (x)Q$	no	yes	no
$(P_1 \mid P_2) + (P_1 \mid P_3)$	$P_1 \mid (P_2 + P_3)$	yes	no	no
$(P_1 + P_2) \mid (P_1 + P_3)$	$P_1 + (P_2 \mid P_3)$	no	no	no
$!\,P \mid !\,Q$	$!(P + Q)$	yes	yes	yes

($\dagger$) x is not free in P. ($\ddagger$) Σ is not empty. ($\S$) $y \in \Sigma$.

Figure 1: Some logical implications and equivalences assuming the disjunctive translation and assuming that + and ! are mapped to logical constants.

Structural equivalence. Before describing our final translation (a variant of the disjunctive translation), we present a simple method for determining structural equivalence between two agents. By $\Sigma; P \vdash Q$ we mean that the formula Q is provable from the formula P given the signature of constants Σ: a formal definition for this three-place predicate is given shortly. The notation $\Sigma; P \dashv\vdash Q$ simply means that $\Sigma; P \vdash Q$ and $\Sigma; Q \vdash P$. We shall extend the domain of $\vdash$ and $\dashv\vdash$ by allowing P and Q to be agent expressions: in this case, one of the above two translations is used to coerce an agent into a formula. Notice that the extension of $\Sigma; P \dashv\vdash Q$ is independent of which translation is used and if Σ is held fixed, the resulting binary relation is an equivalence. Also, since no axioms about communication or matching are used, only the logical identities are used to determine this equivalence. As a result, this equivalence to a good candidate for structural equivalence. Figure 1 provides some examples of $\Sigma; P \dashv\vdash Q$ and $\Sigma; P \vdash Q$ (for which we assume the use of the disjunctive translation described above).

A final translation. A much more serious aspect of the translation given above is the choice of which combinators should be genuine logical constants and which are axiomatized, non-logical constants. It seems an advantage to make as few of the combinators into logical connectives as possible as long as the remaining combinators can be described uniformly in terms of the logical ones. One reason for this advantage is that reduction steps map rather naturally into right introduction rules of the sequent calculus (this will be clear from the proof of Proposition 6), while the left introduction rules do not generally yield plausible reduction steps. For example, if + is mapped to the logical constant $\oplus$ and if we wish reduction to be identified with entailed-by, then we are forced to admit the reduction rule: if P reduces to Q_1 and to Q_2, then P reduces to $Q_1 + Q_2$: a dubious "reduction" rule. Fortunately, it is possible to axiomatize the reduction nature of + and ! by using the clauses

$$\frac{}{\Sigma; P \longrightarrow P}\ \text{initial} \qquad \frac{}{\Sigma; nil \longrightarrow}\ nil\text{-}L \qquad \frac{\Sigma; \Delta \longrightarrow \Gamma}{\Sigma; \Delta \longrightarrow nil, \Gamma}\ nil\text{-}R$$

$$\frac{\Sigma; P, \Delta_1 \longrightarrow \Gamma_1 \quad \Sigma; Q, \Delta_2 \longrightarrow \Gamma_2}{\Sigma; P \mid Q, \Delta_1, \Delta_2 \longrightarrow \Gamma_1, \Gamma_2}\ \mid L \qquad \frac{\Sigma; \Delta \longrightarrow P, Q, \Gamma}{\Sigma; \Delta \longrightarrow P \mid Q, \Gamma}\ \mid R$$

$$\frac{\Sigma; P[t/x], \Delta \longrightarrow \Gamma}{\Sigma; (x)P, \Delta \longrightarrow \Gamma}\ (-)L \qquad \frac{\Sigma \cup \{y\}; \Delta \longrightarrow P[y/x], \Gamma}{\Sigma; \Delta \longrightarrow (x)P, \Gamma}\ (-)R$$

Figure 2: Basic inference rules

$$\frac{\Sigma; \Delta_1 \longrightarrow P, \Gamma_1 \quad \Sigma; P, \Delta_2 \longrightarrow \Gamma_2}{\Sigma; \Delta_1, \Delta_2 \longrightarrow \Gamma_1, \Gamma_2} \qquad \frac{t \text{ a } \Sigma\text{-term} \quad \Sigma \cup \{x\}; \Delta \longrightarrow \Gamma}{\Sigma; [t/x]\Delta \longrightarrow [t/x]\Gamma}$$

Figure 3: Two forms of the cut-rule

$$\forall_o P \forall_o Q\ [P \multimap P + Q] \qquad \forall_o P \forall_o Q\ [Q \multimap P + Q]$$
$$\forall_o P\ [\bot \multimap !\, P] \qquad \forall_o P\ [!\, P \parr !\, P \multimap !\, P] \qquad \forall_o P\ [P \multimap !\, P]$$

These clauses encode right introduction rules without forcing us to accept the corresponding left introduction rules.

Instead of translating π-calculus expressions into the syntax of linear logic, we shall simply use the syntax of the π-calculus. We shall not make any distinction now between agents and formulas over the logical constants $\mid : o \to o \to o$, $(-) : (i \to o) \to o$, $nil : o$ and the non-logical constants $! : o \to o$, $+ : o \to o \to o$, plus the constructors for prefixing and matching, written $x(y).P$, $\bar{x}y.P$, and $[x = y]P$ of types $i \to (i \to o) \to o$, $i \to i \to o \to o$, and $i \to i \to o \to o$, respectively. We shall also assume that there is a denumerably infinite set of constants of type i.

Let Δ and Γ be finite, multisets of formulas. Let Σ be a *signature*, that is, a (possibly empty) set of typed constants. A term t is a Σ-*term* if t is closed and all constants in t are members of Σ. A sequent is a triple $\Sigma; \Delta \longrightarrow \Gamma$ where $\Delta \cup \Gamma$ contains formulas all of whose non-logical constants are from the set Σ. The notation $\Sigma; \Delta \vdash \Gamma$ means that the sequent $\Sigma; \Delta \longrightarrow \Gamma$ has a proof in linear logic (inference rules for the fragment of linear logic needed here are in Figures 2 and 3). The rule $(-)R$ has the proviso that $y \notin \Sigma$, and the rule $(-)L$ has the proviso that t is a Σ-term. Notice that the only inference rule with more than one premise is the left introduction rules for multiplicative disjunction. The structural rules of contraction and weakening are not present. The notation $\Sigma; \Delta \dashv\vdash \Gamma$ means $\Sigma; \Delta \vdash \Gamma$ and $\Sigma; \Gamma \vdash \Delta$. Again, the relation $\dashv\vdash$ will be used as structural equivalence. Since we have reduced the number of logical connectives, this equivalence is now weaker than is described in Figure 1. For example, $P + Q$ is no longer $\dashv\vdash$ related to $Q + P$.

The cut-elimination theorem for linear logic [Gir87] shows that the inference rules in Figure 3 are admissible with respect to the basic set of rules. Given the cut-elimination theorem, provability in this proof system is obviously decidable. Note that we have not given any status to the non-logical constants and their axioms in this proof system. We do this in the next section.

3 Process clauses and process theories

We now step back from the particular example of the π-calculus to consider some general considerations of the logical framework we have picked.

A *process clause* is a closed formula of the form

$$\forall \bar{x}\ [P \multimap Q_1 \mid \cdots \mid Q_m]$$

where $m \geq 1$, P is an agent expression, $Q_1, \ldots, Q_m$ are atomic (formulas with non-logical constants as their head symbols), and all free variables in P (called the *body* of the clause) are free in $Q_1|\cdots|Q_m$ (called the *head* of the clause). The quantified variables $\bar{x}$ may be of type i and o, as well as higher-order types, for example, $i \to o$. If $m = 1$, such a clauses is also called a *single-conclusion* clause; otherwise, it is called a *multiple-conclusion* clause. An instance of a process clause using Σ-terms (for a given Σ) is called a Σ-instance of that clause.

The propositional structure of process clauses is similar to the clauses studied by Andreoli and Pareschi [AP91] where & and $\top$ (erasure) are also permitted in the body of clauses: their formalism, however, permits neither universal quantification in the body of clauses nor quantification of higher-type variables.

A *process theory* is a finite, possibly empty, set H of process clauses. An H*-proof* is a proof built using the rules in Figure 2 and one inference of the form

$$\frac{\Sigma; \Delta \longrightarrow \Gamma, P}{\Sigma; \Delta \longrightarrow \Gamma, Q_1, \ldots, Q_m}$$

for every clause $\forall \bar{x}\ [P \multimap Q_1 \mid \cdots \mid Q_m]$ in H. When an H-clause is written as an inference rule in this way, that inference rule is called an H-inference rule. Let Σ be a signature that contains all the non-logical constants contained in clauses of H. We write $\Sigma; \Delta \vdash_H \Gamma$ to mean that the sequent $\Sigma; \Delta \longrightarrow \Gamma$ has an H-proof. The structure of H-proofs are particularly simple, as we shall now see.

The *site* of an instance of an inference rule is a multiset of occurrence of formulas in the concluding sequent defined using the following cases: if the inference rule is the initial rule proving the sequent $\Sigma; P \longrightarrow P$, then the site is the multiset containing both occurrences of P; the site for an introduction rule is the singleton multiset containing the formula occurrence containing the introduced logical constant; and the site for an H-inference rule based on the clause $\forall \bar{x}\ [P \multimap Q_1 | \cdots | Q_m]$ is the multiset containing the occurrences of the instances of the formulas $Q_1, \ldots, Q_m$. Two inference rules *permute* if whenever instances of these two rules have a common sequent as a conclusion and the sites of these two inference rule instances are disjoint, then those inference rules can be composed in either order to yield identical premises to their composition. When doing a bottom-up search for proofs, the order in which permuting inference rules are applied is not important. For example, the following two proof fragments demonstrate that $(-)R$ and $|R$ permute over each other.

$$\dfrac{\dfrac{\Sigma, y : i; \Delta \longrightarrow P, Q, [y/x]R, \Gamma}{\Sigma; \Delta \longrightarrow P, Q, (x)R, \Gamma}}{\Sigma; \Delta \longrightarrow P \mid Q, (x)R, \Gamma} \qquad \dfrac{\dfrac{\Sigma, y : i; \Delta \longrightarrow P, Q, [y/x]R, \Gamma}{\Sigma, y : i; \Delta \longrightarrow P \mid Q, [y/x]R, \Gamma}}{\Sigma; \Delta \longrightarrow P \mid Q, (x)R, \Gamma}$$

We assume here that $y \notin \Sigma$.

Proposition 1 *All pairs of right rules (nil-R, $|R$, and $(-)R$) and H-inference rules permute over each other.*

Proof. This proposition follows from simply checking all cases. The case where a $(-)R$ inference rule is below an H-inference rule requires the assumption about process clauses that all free variables in the body of clauses are also free in their head. □

Consider the process theory that contains the single process clause

$$\forall P \forall Q \forall x [P \mid Q \multimap x.P \mid \bar{x}.Q].$$

Here, prefixing is represented by two non-logical constants, both of type $i \to o \to o$. The order of the two H-rules in the proof fragment

$$\dfrac{\dfrac{\Sigma; P \mid Q \mid R \longrightarrow P, Q, R}{\Sigma; P \mid Q \mid R \longrightarrow b.P, Q, \bar{b}.R}}{\Sigma; P \mid Q \mid R \longrightarrow a.b.P, \bar{a}.Q, \bar{b}.R}$$

cannot be switched: the site of the lower rule contains a subformula that is in the site of the upper rule. Notice also that a sequent with right-hand side $a.P, \bar{a}.Q, \bar{a}.R$ can be the result of an H-inference rule in two ways: the choice of one of these precludes the other choice.

Proposition 2 *If $\Sigma; \Delta \longrightarrow \Gamma$ has an H-proof, it has an H-proof Ξ such that Ξ has an occurrence of a sequent $\Sigma'; \Delta \longrightarrow \Gamma'$ where all inference rules above this sequent occurrence are left introduction rules and instances of the initial inference rule and all inference rules below this sequent occurrence are right introduction rules or H-inference rules.*

The sequent $\Sigma'; \Delta \longrightarrow \Gamma'$ is called the *crossover sequent* for the proof Ξ.

Proof. Let Ψ be an H-proof of $\Sigma; \Delta \longrightarrow \Gamma$. If there is no pair of inference rules such that the lower one is a left introduction rule and the upper one is a right introduction rule or an H-inference rule, then Ψ has the structure described for the Ξ in the proof. Otherwise, assume that such a pair of inference rules exists. It is then possible to permute the order of these two rules and still have a proof of the same endsequent. The fact that a |-L below a $(-)$-R rule can be permuted requires observing the general fact that if $\Sigma; \Delta \longrightarrow \Gamma$ has an H-proof then $\Sigma'; \Delta \longrightarrow \Gamma$ has an H-proof whenever Σ' is a signature that contains Σ. A simple inductive argument then shows that by doing enough permutations, all such pairs of inference rules can be removed. □

Corollary 3 *If $\Sigma; \Delta \longrightarrow \Gamma$ has an H-proof then Δ must be a singleton multiset.*

Proof. Assume not and let $\Sigma; \Delta \longrightarrow \Gamma$ be a sequent in which Δ contains more than one member and which has an H-proof Ξ of minimal height. Clearly, $\Sigma; \Delta \longrightarrow \Gamma$ is not an initial sequent. But it is simple to check that no matter what its last inference rule is, Ξ must contain a proper subproof of a sequent containing more than one formula on its left. This contradicts the choice of Ξ. □

Proposition 4 *The two cut rules of Figure 3 are admissible in H-proofs.*

Proof.

Let Ξ_1 be an H-proof for $\Sigma; Q \longrightarrow P, \Gamma_1$ and let Ξ_2 be an H-proof for $\Sigma; P \longrightarrow \Gamma_2$. We must show that there is an H-proof for $\Sigma; Q \longrightarrow \Gamma_1, \Gamma_2$. First, we can assume that the last inference rule in Ξ_2 is a left introduction rule or the initial sequent rule since cut permutes up through all the right introduction rules and H-inference rules for proofs of this premise. Similarly, we can assume that the last inference rule in Ξ_1 is either a right introduction rule or an H-inference rule in which the occurrence of P is in the site. Now, if P has a top-level logical constant, then Ξ_1 ends in a right introduction of that constant and Ξ_2 ends in a left introduction of that constant. The usual movement of cut upwards in a proof will work in this case. Finally, if P is atomic, then Γ_2 is the multiset that consists of just P, so we can simply use Ξ_1 as the proof of $\Sigma; Q \longrightarrow \Gamma_1, \Gamma_2$.

The proof that the other cut rule involving substitution is admissible is simpler and more direct. □

The following proposition demonstrates that process theories can be viewed as multiple conclusion logic programming languages.

Proposition 5 *The sequent has an H-proof if and only if it has a uniform H-proof.*

Proof. Assume that Ξ is a cut-free, atomically closed sequent proof. An *atomically closed* proof is a proof in which all initial sequents contain only atomic formulas: it is easy to show that a sequent has a proof if and only if it has an atomically closed proof. Using the definition of uniform proofs for multiple conclusion sequents given in the introduction, we now show that Ξ is, in fact, a uniform proof. Let Ψ be a subproof of Ξ that proves the sequent $\Sigma; \Delta \longrightarrow \Gamma$ and let B be a non-atomic formula occurrence in Γ. Since the sites of H-rules contain only atoms and since initial sequent rules involve only atoms, the top-level logical connective of B must be introduced somewhere is Ψ. Given that all left rules can be permuted upward through a proof and that all right-rules and H-inference rules also permute over each other, a series of permutations can carry the proof Ψ into a proof Ψ' in

$$\frac{}{\Sigma; S \Longrightarrow R}\ \mathrm{H}$$

Figure 4: Theory reduction rules: provided that $R \multimap S$ is a Σ-instance of a clause in H.

$$\frac{\Sigma \cup \{x\}; P \Longrightarrow P'}{\Sigma; (x)P \Longrightarrow P'}\ \mathrm{INS} \qquad \frac{t \text{ is a } \Sigma\text{-term} \quad \Sigma; Q \Longrightarrow P[t/x]}{\Sigma; Q \Longrightarrow (x)P}\ \mathrm{GEN}$$

$$\frac{\Sigma; P \Longrightarrow P'}{\Sigma; P \mid Q \Longrightarrow P' \mid Q}\ \mathrm{PAR}$$

Figure 5: Descent reduction rules: In INS, x is not free in P'.

$$\frac{\Sigma; P \dashv\vdash Q}{\Sigma; P \Longrightarrow Q}\ \mathrm{REF} \qquad \frac{\Sigma; P \Longrightarrow Q \quad \Sigma; Q \Longrightarrow R}{\Sigma; P \Longrightarrow R}\ \mathrm{TRANS}$$

Figure 6: Structural reduction rules.

which the last inference rule introduces the top-level logical connective of B. Thus, Ξ is a uniform proof. $\Box$

Process calculi are generally described using a notion of reduction. We will focus on unlabeled reduction, such as is found in [Mil90]; an example of labeled reductions is used in Section 5. Figures 4, 5, and 6 present a proof system for a formulation of reduction determined by a process theory H. The following proposition shows the close relation between $\vdash_H$ and such reduction.

Proposition 6 *Let H be a process theory and let reduction be defined with respect to it. Then $\Sigma; Q \Longrightarrow P$ has a proof if and only if $\Sigma; P \vdash_H Q$.*

Proof. First, assume that $\Sigma; Q \Longrightarrow P$ has a reduction proof. Proceed by induction on the structure of that proof. If the proof is of height 1, then it is an instance of either a theory reduction rule or the REF rule. In each case, it follows immediately that $\Sigma; P \vdash_H Q$.

To handle the remaining structural rule, assume that the proof ends in an instance of the TRANS rule. By induction, $\Sigma; R \vdash_H Q$ and $\Sigma; Q \vdash P$, and thus $\Sigma; R \vdash_H P$ by cut.

Assume that the last inference rule was a descent reduction rule. If that rule is INS, then induction guarantees that $\Sigma \cup \{x\}; P' \longrightarrow P$ has a proof. Adding the $(-)R$ rules yields a proof for $\Sigma; P' \longrightarrow (x)P$. Similarly, if that rule is GEN then induction provides a proof of $\Sigma; P[t/x] \longrightarrow Q$ where t is a Σ-term. Adding the $(-)L$ rule yields a proof of $\Sigma; (x)P \longrightarrow Q$. If that rule is PAR, then induction provides a proof of $\Sigma; P' \longrightarrow P$. Using the initial sequent $\Sigma; Q \longrightarrow Q$ and the $|L$ and $|R$ rules yields a proof of $\Sigma; P' \mid Q \longrightarrow P \mid Q$.

Now consider the converse of this proposition. That is, assume that $\Sigma; P \longrightarrow Q_1, \ldots, Q_n$ has an H-proof. We prove by induction on the structure of a (cut-free) H-proof that $\Sigma; Q_1 \mid \cdots \mid Q_n \Longrightarrow P$ has a reduction proof. (If $n = 0$ then $Q_1 \mid \cdots \mid Q_n$ is simply nil.)

Case initial: If the proof is an instance of the initial rule, then $n = 1$ and Q_1 is P. The reduction proof is simply an instance of the REF rule.

Case nil-R: The final sequent is $\Sigma; P \longrightarrow nil, Q_1, \ldots, Q_n$ and induction provides a proof of $\Sigma; Q_1 \mid \cdots \mid Q_n \Longrightarrow P$. Noticing that $\Sigma; nil \mid R \dashv\vdash R$ for any R, use the REF and TRANS rules to provide a reduction proof for $\Sigma; nil \mid Q_1 \mid \cdots \mid Q_n \Longrightarrow P$.

Case $|R$: The final sequent is $\Sigma; P \longrightarrow P \mid Q, Q_1, \ldots, Q_n$ and induction provides a proof of $\Sigma; (P \mid Q) \mid Q_1 \mid \cdots \mid Q_n \Longrightarrow P$. If this is not the desired reduction sequent already, simply use REF and TRANS to associate the $|$'s differently.

Case $(-)R$: The final sequent is $\Sigma; P \longrightarrow (x)Q, Q_1, \ldots, Q_n$ and induction provides a proof of $\Sigma \cup \{y\}; Q[y/x] \mid Q_1 \mid \cdots \mid Q_n \Longrightarrow P$. Adding the INS proof rule yields a proof of $\Sigma; (y)(Q[y/x] \mid Q_1 \mid$

$\cdots \mid Q_n) \Longrightarrow P$. Since $\Sigma; (y)(Q[y/x] \mid Q_1 \mid \cdots \mid Q_n) \dashv\vdash (x)Q \mid Q_1 \mid \cdots \mid Q_n$, a use of REF and TRANS yields a proof of $\Sigma; (x)Q \mid Q_1 \mid \cdots \mid Q_n \Longrightarrow P$.

Case H-inference rule: The final sequent is $\Sigma; P \longrightarrow Q_1, \ldots, Q_i, \ldots, Q_n$ and $R \multimap Q_1 \mid \cdots \mid Q_i$ is a Σ-instance of a rule in H. Thus, induction provides a proof of $\Sigma; R \mid Q_{i+1} \mid \cdots \mid Q_n \Longrightarrow P$. But by the H-rule (Figure 4), we have $\Sigma; Q_1 \mid \cdots \mid Q_i \Longrightarrow R$. By $n - i$ applications of PAR, we have $\Sigma; Q_1 \mid \cdots \mid Q_n \Longrightarrow R \mid Q_{i+1} \mid \cdots \mid Q_n$. A use of TRANS and we are finished.

Case nil-L: The final sequent is $\Sigma; nil \longrightarrow$. But $\Sigma; nil \Longrightarrow nil$ follows from REF.

Case $|L$: The final sequent is $\Sigma; P_1 \mid P_2 \longrightarrow Q_1, \ldots, Q_n$ and induction provides proofs of $\Sigma; Q_1 \mid \cdots \mid Q_i \Longrightarrow P_1$ and $\Sigma; Q_{i+1} \mid \cdots \mid Q_n \Longrightarrow P_2$, so $i = 1, \ldots, n$. (Of course, it is permitted to permute the Q's prior to splitting them.) Using $n - i$ applications of PAR provides a reduction proof of $\Sigma; Q_1 \mid \cdots \mid Q_n \Longrightarrow P_1 \mid Q_{i+1} \mid \cdots \mid Q_n$. One application of PAR yields $\Sigma; P_1 \mid Q_{i+1} \mid \cdots \mid Q_n \Longrightarrow P_1 \mid P_2$. Thus, one use of TRANS provides a reduction proof of $\Sigma; Q_1 \mid \cdots \mid Q_n \Longrightarrow P_1 \mid P_2$.

Case $(-)L$: The final sequent is $\Sigma; (x)P \longrightarrow Q_1, \ldots, Q_n$ and induction provides a proof of $\Sigma; Q_1 \mid \cdots \mid Q_n \Longrightarrow P[t/x]$ for t a Σ-term. The GEN rule immediately yields $\Sigma; Q_1 \mid \cdots \mid Q_n \Longrightarrow (x)P$. □

4 Reduction in the π-calculus

We should like to identify reduction in the π-calculus with the reduction relation defined in the previous section using the signature

$$\begin{aligned}\Sigma_\pi = \{&\mathtt{send} : i \to i \to o \to o,\ \mathtt{get} : i \to (i \to o) \to o,\\ &\mathtt{match} : i \to i \to o \to o,\ ! : o \to o,\ + : o \to o \to o\}\end{aligned}$$

and the following theory, which we shall call the π-theory.

$$\begin{gathered}\forall_i x \forall_i y \forall_o S \forall_{i \to o} R\ [Ry \parr S \multimap \mathtt{get}\ x\ R \parr \mathtt{send}\ x\ y\ S]\\ \forall_i x \forall_{i \to o} P\ [P \multimap \mathtt{match}\ x\ x\ P]\\ \forall_o P \forall_o Q\ [P \multimap P + Q] \qquad \forall_o P \forall_o Q\ [Q \multimap P + Q]\\ \forall_o P\ [\bot \multimap !\, P] \qquad \forall_o P\ [!\, P \parr !\, P \multimap !\, P] \qquad \forall_o P\ [P \multimap !\, P]\end{gathered}$$

A π-proof is an H-proof, where H is the set of axioms displayed above. We write $\Sigma; \Delta \vdash_\pi \Gamma$ if the sequent $\Sigma_\pi \cup \Sigma; \Delta \longrightarrow \Gamma$ has a π-proof. Notice that since Σ_π contains no constructors for type i, the signature Σ can be restricted to being composed only of tokens of type i. Thus, t is a Σ-term if and only if $t \in \Sigma$: the only values used during computations (search for proofs) are names and not general terms.

Notice that it is very easy to accommodate definition of agents using process clauses: the definition $C(\bar{x}) = P$, where the free variables of P are contained in the list $\bar{x}$, can be translated to the process clauses $\forall \bar{x}[P \multimap C(\bar{x})]$. Such clauses can be added to the base π-theory.

Since the notion of reduction is central to the definition of a process calculus, we must be very careful in making any claim to having captured the π-calculus as it is described in, say, [MPW89a, MPW89b]. There seems to be at least the following significant differences with the description given in those reports.

1. Signatures are made explicit and reductions depend on them.

2. The + and ! combinators are treated only via computation rules: there are no rules for explicitly descending through them. Thus several reduction steps defined here may be needed to account for a single reduction step of the π-calculus.

3. The GEN and INS rules do not correspond to any rules of [MPW89a, MPW89b]. As a reduction rule, GEN does seem odd since it does not seem to be making anything simpler. Its main purpose seems to be that it allows the result of a reduction to discharge its dependence on any part of the surrounding signature. Notice that a version of the reduction rule for restriction

in the π-calculus can be proved here: if $\Sigma \cup \{x\}; P \Longrightarrow Q$ can be proved then by one instance each of GEN and INS, we have a proof of $\Sigma; (x)P \Longrightarrow (x)Q$.

Given our plan to use proof theory to organize the syntax of process calculi, these differences seem forced. Probably only additional results will tell us if what is defined here is significantly different from the π-calculus of [MPW89a]. Of course, the process calculi defined here may be of their own interest.

5 An analysis of the propositional fragment

Because the π-calculus communicates values of type i only, we shall think of the π-calculus as a first-order theory. In this section we analysis the "propositional" fragment of the π-calculus. In particular, we shall only be interested in synchronization and not with value passing, binding, or restriction, or with match. Thus, agent expressions in the propositional calculus are defined via the grammar

$$P ::= nil \quad | \quad P_1 \,|\, P_2 \quad | \quad P_1 + P_2 \quad | \quad !\,P \quad | \quad a.P \quad | \quad \bar{a}.P,$$

where, a ranges over some fixed, finite set of names Σ_0. We refer to this propositional theory as the π_0-calculus. It is determined by the signature

$$\Sigma_{\pi_0} = \{\mathtt{send} : i \to o \to o,\ \mathtt{get} : i \to o \to o,\ ! : o \to o,\ + : o \to o \to o\}$$

and the following set of process clauses.

$$\forall_i x \forall_o S \forall_o R\, [R \parr S \multimap \mathtt{get}\ x\ R \parr \mathtt{send}\ x\ S]$$
$$\forall_o P \forall_o Q\, [P \multimap P + Q] \qquad \forall_o P \forall_o Q\, [Q \multimap P + Q]$$
$$\forall_o P\, [\bot \multimap !\,P] \qquad \forall_o P\, [!\,P \parr !\,P \multimap !\,P] \qquad \forall_o P\, [P \multimap !\,P]$$

We shall, of course, identify $\mathtt{get}\ a\ P$ and $\mathtt{send}\ a\ P$ with $a.P$ and $\bar{a}.P$, respectively, and identify $\bar{\bar{a}}$ with a. A π_0-proof is an H-proof, where H is the set of axioms displayed above. We write $\Delta \vdash_{\pi_0} \Gamma$ if the sequent $\Sigma_{\pi_0} \cup \Sigma_0; \Delta \longrightarrow \Gamma$ has a π_0-proof. Since agents of the π_0-calculus do not contain universal quantification, all occurrences of signatures in any π_0-proof are equal and, therefore, we shall choose not to display signatures. The π_0-calculus is essentially a subset of CCS.

Given this proof-theoretic setting, a natural way to attribute meaning $[\![P]\!]$ to an agent P is via the definition

$$[\![P]\!] = \{W \mid \vdash_{\pi_0} P \,|\, W, \text{ where } W \text{ is an agent}\}.$$

The goal would then be to say that two agents, P and Q, are equivalent, in some sense, if $[\![P]\!] = [\![Q]\!]$. Unfortunately, using this definition, all agents are equivalent since $[\![P]\!]$ is always empty: there is no notion of a "true" agent. The only notion we have so far is that of one agent reducing to (implied-by) another.

Since we are inside a logic containing many more logical constants than we are using so far, it is possible to extend the notion of agents to *co-agents*, one of which will be "truth." Given some notion of co-agents, we shall define the meaning of agents using

$$[\![P]\!] = \{W \mid \vdash_{\pi_0} P \,|\, W, \text{ where } W \text{ is a co-agent}\}.$$

Thus, co-agents will be used to probe the behavior of agents. It is important to make the following observation: no matter what we choose for co-agents, if $[\![P]\!] \subseteq [\![Q]\!]$ then $[\![P + Q]\!] = [\![Q]\!]$. Thus, if $[\![P]\!] \subseteq [\![Q]\!]$ is ever strictly true, we have not captured deadlock within our theory of equivalence.

In analyzing the π_0-calculus, we shall first introduce two co-agents, identified as the two (linear) logical connectives $\top$ (erasure) and & (additive conjunction) for which their right introduction rules are given in Figure 7. Assume for now that we define a co-agent to be any expression that contains at least one occurrence of either $\top$ or &. We can make the following observations regarding occurrences of $\top$ in $[\![P]\!]$.

$$\frac{}{\Sigma;\Delta \longrightarrow \top, \Gamma}\ \top\text{-R} \qquad \frac{\Sigma;\Delta \longrightarrow \Gamma, W_1 \qquad \Sigma;\Delta \longrightarrow \Gamma, W_2}{\Sigma;\Delta \longrightarrow \Gamma, W_1 \,\&\, W_2}\ \&\text{-R}$$

Figure 7: Proof rules for the two co-agent connectives $\top$ and $\&$.

- It is always the case that $\top \in [\![P]\!]$.
- The agent P has an a-transition if and only if $\bar{a}.\top \in [\![P]\!]$.
- The agent P has an a-transition followed by a b-transition if and only if $\bar{a}.\bar{b}.\top \in [\![P]\!]$.

Thus, P has a trace $a_1, \ldots, a_n$ if and only if $\bar{a}_1.\cdots.\bar{a}_n.\top \in [\![P]\!]$. If $\top$ were the only co-agent, then the equivalence described by $[\![P]\!] = [\![Q]\!]$ would be that of trace equivalence.

By allowing $\&$ as a co-agent expression, we can make more distinctions between the behaviors of agents. For example, let P be $a.b.nil + a.(c.nil + d.nil)$ and let Q be $a.(b.nil + c.nil) + a.d.nil$. While these have the same traces, the co-agent $\bar{a}.(\bar{c}.\top \,\&\, \bar{d}.\top)$ is a member of $[\![P]\!]$ but not of $[\![Q]\!]$. Notice, however, that since $[\![a.b.nil]\!] \subseteq [\![a.(b.nil+c.nil)]\!]$, it follows that $[\![a.(b.nil+c.nil)]\!] = [\![a.b.nil+ a.(b.nil + c.nil)]\!]$.

Clearly, co-agents are acting as testers. The logical constant $\top$ behaves very much as the w tester in [Hen88]. The logical constant $\&$ specifies two tests that a process must satisfy simultaneous: in a sense, the process must be copied and the two copies must be able to satisfy two separate tests. Thus co-agents treat agents extensionally, that is, as black boxes whose internal structure is not examined directly. Consider what would happen if $\otimes$ (the multiplicative conjunction) were permitted to also be a co-agent connective. The co-agent $W_1 \otimes W_2$ would require that the agent being tested be divided into two pieces, one of which must pass W_1 and the other W_2. While such a tensor tester may have its uses, we do not consider it any further here.

It will be important for a subsequent result (regarding bisimilarity) that we allow possibly infinite conjunctions. Let I be a denumerable set (possibly empty). The right introduction rule for $\&_{i\in I}$ is given by the inference figure

$$\frac{\longrightarrow \Gamma, W_{i_1} \cdots \longrightarrow \Gamma, W_{i_j} \cdots}{\longrightarrow \Gamma, \&_{i\in I} W_i},$$

where $I = \{i_1, \ldots, i_j, \ldots\}$. If the index set is empty, then $\&_{i\in I}$ is the same as $\top$ and if the index set has two element, then $\&_{i\in I}$ is the same as $\&$. The term *co-agent* now refers to any agent expression containing at least one occurrence of $\&_{i\in I}$, where I is not a singleton.

The following proposition shows that if co-agents are only used to define testing equivalence, they only need to be built up out of prefixing and the co-agent combinator $\&_{i\in I}$.

Proposition 7 *Define $[\![\Gamma]\!]_1$ to be the set of all multisets of agents and co-agents Δ such that $\vdash_{\pi_0} \Gamma, \Delta$. Define $[\![\Gamma]\!]_2$ to be the set of co-agents W built exclusively from occurrences of the indexed $\&$ and prefixing so that $\vdash_{\pi_0} \Gamma, W$. For multisets of agents Γ and Ψ, $[\![\Gamma]\!]_1 = [\![\Psi]\!]_1$ if and only if $[\![\Gamma]\!]_2 = [\![\Psi]\!]_2$.*

Proof. The proof that $[\![\Gamma]\!]_1 = [\![\Psi]\!]_1$ implies $[\![\Gamma]\!]_2 = [\![\Psi]\!]_2$ is immediate. Thus, assume that $[\![\Gamma]\!]_2 = [\![\Psi]\!]_2$ and that Δ is a multiset of agents and co-agents such that $\vdash_{\pi_0} \Gamma, \Delta$. Consider the proof system given Figure 8. A π_0-proof of $\longrightarrow \Gamma, \Delta$ can then be extended to a proof in Figure 8 of $\longrightarrow \Gamma \boxed{R} \Delta$, for some R built exclusively from occurrences of the indexed $\&$ and prefixing. Thus, $R \in [\![\Gamma]\!]_2$ and $R \in [\![\Psi]\!]_2$. Now given a π_0-proof of $\longrightarrow \Psi, R$ and the proof of $\longrightarrow \Gamma \boxed{R} \Delta$, it is an easy to construct a proof of $\longrightarrow \Psi \boxed{R} \Delta$. Thus, $\vdash_{\pi_0} \Psi, \Delta$ and $\Delta \in [\![\Psi]\!]_1$. The converse inclusion is similar. $\Box$

The following proposition describes the fact that in the bottom-up search for proofs involving co-agents, the top-level logical structure of co-agents can be addressed first. We shall strive to preserve this property when we add one more connective to the structure of co-agents.

$$\frac{\longrightarrow \Gamma \boxed{R} \Delta}{\longrightarrow nil, \Gamma \boxed{R} \Delta} \qquad \frac{\longrightarrow P, Q, \Gamma \boxed{R} \Delta}{\longrightarrow P \mid Q, \Gamma \boxed{R} \Delta} \qquad \frac{\longrightarrow \Gamma, Q \boxed{R} \Delta}{\longrightarrow \Gamma, P \boxed{R} \Delta}\ \dagger$$

$$\frac{\longrightarrow \Gamma \boxed{R} \Delta}{\longrightarrow \Gamma \boxed{R} nil, \Delta} \qquad \frac{\longrightarrow \Gamma \boxed{R} P, Q, \Delta}{\longrightarrow \Gamma \boxed{R} P \mid Q, \Delta} \qquad \frac{\longrightarrow \Gamma \boxed{R} Q, \Delta}{\longrightarrow \Gamma \boxed{R} P, \Delta}\ \dagger$$

$$\frac{\longrightarrow P, Q, \Gamma \boxed{R} \Delta}{\longrightarrow a.P, \bar{a}.Q, \Gamma \boxed{R} \Delta} \qquad \frac{\longrightarrow P, \Gamma \boxed{R} Q, \Delta}{\longrightarrow a.P, \Gamma \boxed{\bar{a}.R} \bar{a}.Q, \Delta} \qquad \frac{\longrightarrow \Gamma \boxed{R} P, Q, \Delta}{\longrightarrow \Gamma \boxed{R} a.P, \bar{a}.Q, \Delta}$$

$$\frac{\text{for all } i \in I \quad \longrightarrow \Gamma \boxed{R_i} W_i, \Delta}{\longrightarrow \Gamma \boxed{\&_{i\in I} R_i} \&_{i\in I} W_i, \Delta}$$

Figure 8: A proof system used for "interpolating" between agents and co-agents. The † proviso: $Q \multimap P$ is a instance of a single conclusion π_0-axiom.

Proposition 8 *If $\longrightarrow \Gamma, W$ has a proof, where Γ is a multiset of agents and W is a co-agent built exclusively from occurrences of the indexed & and prefixing, then this sequent has a proof Ξ such that for every occurrence of a sequent in Ξ, if the co-agent expression in that sequent is a top-level $\&_{i\in I}$ then that sequent occurrence is the conclusion of a $\&_{i\in I}$-R introduction rule.*

Proof. This can be proved by a observing that if there is an inference rule in a proof of $\longrightarrow \Gamma, W$ immediately below an instance of a $\&_{i\in I}$-R introduction rule, then the $\&_{i\in I}$-R introduction rule can be permuted lower. □

So far co-agents are extracting only positive information. The equivalence of processes, $[\![P]\!] = [\![Q]\!]$, does not come close to the notion of bisimulation since it is not possible to test for what a process cannot do. For this, it appears necessary to leave the usual logical connectives and their introduction rules and develop a notion of negation as "failing to pass a test" or of "negation-as-failure," as it is often called in the logic programming literature.

A notion of negation-as-failure cannot be achieved by simply adding introduction rules. Instead we shall use a hierarchy of proof systems $\{S_n \mid n = 0, 1, 2, \ldots\}$ such that S_n can handle a nesting of at most n occurrences of negation and where non-provable sequents in the S_n proof system yield initial sequents (axioms) for negation in the S_{n+1} proof system (S_0 is identified with the π_0 proof system). Even given this hierarchy of proof systems, negation can still cause us one serious problem. Notice that the sequent $\longrightarrow a.nil, \bar{b}.\top$ has no π_0-proof. Thus, in the S_1 proof system, we shall accept the sequent $\longrightarrow a.nil, \neg\bar{b}.\top$ as initial. If we do not add any further restrictions, there will also be an S_1 proof of $\longrightarrow a.nil + b.nil, \neg\bar{b}.\top$. This conclusion is not acceptable since there is a S_0-proof (and, hence, S_1 proof) of $\longrightarrow a.nil + b.nil, \bar{b}.\top$. Thus, it would be possible for an agent ($a.nil + b.nil$) to pass a test ($\bar{b}.\top$) and its complement. A suitable solution to this problem is to insist that proofs in S_n introduce $\neg$ as early as possible; that is, require that any S_{n+1} proof of a sequent $\longrightarrow \Gamma, \neg W$ be, in fact, an initial sequent of S_{n+1}. Notice that this condition is essentially equivalent to the one which can be verified for $\&_{i\in I}$ (Proposition 8). With negation, however, this condition cannot be inferred so we must enforce it.

Given this motivation, we can now make the following definitions. Let C_n for $n \geq 0$ be sets of expressions defined by the following recursion on the structure of formulas.

- If I is a denumerable (possibly empty) set and for all $i \in I$, $W_i \in C_n$ then $\&_{i\in I} W_i \in C_n$.
- If $P \in C_n$ then $a.P \in C_n$ and $\bar{a}.P \in C_n$.
- If $P \in C_n$ then $\neg P \in C_{n+1}$.

We now introduce, for each $n \geq 0$, a proof system S_n. Sequents in S_n-proofs will be of the form $\longrightarrow \Gamma, W$ where Γ is a multiset of agents and $W \in C_n$. For each $n \geq 0$, S_n contains the right introduction rules for *nil*, $|$, and $\&_{i \in I}$ as well as the inference rules for all the π_0-theory axioms. In particular, the only initial sequents in S_0 are given by the $\&_{i \in I}$-R rule when I is empty. The systems S_{n+1} have as additional initial sequents $\longrightarrow \Gamma, \neg W$, where the sequent $\longrightarrow \Gamma, W$ does not have an S_n-proof. Such initial sequents are called *negative initial sequents*. An S_n-proof is a tree structure arrangement of such initial sequents and inferences with the following proviso: if the sequent $\longrightarrow \Gamma, \neg W$ has an occurrence in the tree, then that occurrence is an initial sequent. For $n \geq 0$, we write $\vdash^n \Gamma, W$ to mean that there is an S_n-proof of $\longrightarrow \Gamma, W$. Notice that $\vdash_{\pi_0} \Gamma, W$ if and only if $\vdash^0 \Gamma, W$. Let $C_\omega = \bigcup_{n \geq 0} C_n$.

Notice that it is easy to extend Proposition 8 to S_n for $n \geq 0$. That is, S_n-proofs can be assumed to be such that whenever a sequent occurrence has an occurrence of a $\&_{i \in I}$ co-agent, that sequent occurrence is the conclusion of the $\&_{i \in I}$ introduction rule.

Proposition 9 *Let $W \in C_n$, let $m \geq n$, and let Γ be a multiset of agent expressions. Then $\vdash^m \Gamma, W$ if and only if $\vdash^{m+1} \Gamma, W$.*

Proof. By induction on m. If $m = 0$ then $n = 0$. Since W has no occurrences of negations, the result is immediate. Assume that $m > 0$. Let Ξ be an S_m-proof of $\longrightarrow \Gamma, W$. If Ξ has no negative initial sequents, then Ξ is both an S_0 and S_{m+1}-proof. Thus, assume that Ξ contains the initial sequent $\longrightarrow \Delta, \neg W'$. Since $\neg W'$ is a subformula of W, $W' \in C_{n-1}$. Also, there is no S_{m-1}-proof of $\longrightarrow \Delta, W'$. By the inductive hypothesis, there is no S_m-proof of $\longrightarrow \Delta, W'$ so S_{m+1} contains the initial sequent $\longrightarrow \Delta, \neg W'$. Since every initial sequent of Ξ is initial in S_{m+1}, Ξ is an S_{m+1} proof of $\longrightarrow \Gamma, W$. Conversely, let Ξ be an S_{m+1}-proof of $\longrightarrow \Gamma, W$. Again, let $\longrightarrow \Delta, \neg W'$ be a negative initial sequent in Ξ. Thus, $W' \in C_{n-1}$ and there is no S_m-proof of $\longrightarrow \Delta, W'$. By the inductive hypothesis, there is no S_{m-1} proof of $\longrightarrow \Delta, W'$ so $\longrightarrow \Delta, W'$ is an initial sequent in S_m. Thus, Ξ is also an S_m proof. □

Let Γ be a multiset of agent expressions and let $W \in C_\omega$. We write $\vdash^\omega \Gamma, W$ if there is some $n \geq 0$ such that $\vdash^n \Gamma, R$. Notice that if $W \in C_n$ for some $n \geq 0$, Proposition 9 implies that $\vdash^\omega \Gamma, W$ if and only if $\vdash^n \Gamma, W$.

Proposition 10 *Let Γ be a multiset of agent expressions and let $W \in C_\omega$.*

(i) Either $\vdash^\omega \Gamma, W$ or $\vdash^\omega \Gamma, \neg W$.

(ii) It is not the case that $\vdash^\omega \Gamma, W$ and $\vdash^\omega \Gamma, \neg W$.

(iii) $\nvdash^\omega \Gamma, W$ if and only if $\vdash^\omega \Gamma, \neg W$.

(iv) $\vdash^\omega \Gamma, W$ if and only if $\vdash^\omega \Gamma, \neg\neg W$.

Proof. To prove (i), let n be such that $W \in C_n$. Then $\longrightarrow \Gamma, W$ is either provable or not provable in S_n. In the first case, $\vdash^\omega \Gamma, W$. In the second case, $\vdash^{n+1} \Gamma, \neg W$ and therefore $\vdash^\omega \Gamma, \neg W$.

To prove (ii), let n be such that $W \in C_n$ and assume that $\vdash^\omega \Gamma, W$ and $\vdash^\omega \Gamma, \neg W$. By Proposition 9, $\vdash^n \Gamma, W$ and $\vdash^{n+1} \Gamma, \neg W$. Given the restriction on proofs involving negations in proofs, $\longrightarrow \Gamma, \neg W$ is an initial sequent of S_{n+1} and thus there is no S_n-proof of $\longrightarrow \Gamma, W$, which is a contradiction.

To prove (iii), let n be such that $W \in C_n$. Thus by Proposition 9 $\nvdash^\omega \Gamma, W$ if and only if $\nvdash^n \Gamma, W$. But this is equivalent to $\vdash^{n+1} \Gamma, \neg W$. By Proposition 9 again, this is equivalent to $\vdash^\omega \Gamma, \neg W$.

To prove (iv), notice that by (iii), $\vdash^\omega \Gamma, W$ is equivalent to $\nvdash^\omega \Gamma, \neg W$, which (by (iii) again) is equivalent to $\vdash^\omega \Gamma, \neg\neg W$. □

For Γ a multiset of agent expressions, define $[\![\Gamma]\!] = \{W \in C_\omega \mid \vdash^\omega \Gamma, W\}$.

Proposition 11 *Let P and Q be two agent expressions. Then $[\![P]\!] \subseteq [\![Q]\!]$ if and only if $[\![P]\!] = [\![Q]\!]$.*

Proof. Assume that $[\![P]\!] \subseteq [\![Q]\!]$ and that $[\![P]\!] \neq [\![Q]\!]$. Thus, there is a $W \in C_\omega$ such that $\vdash^\omega Q, W$ but $\not\vdash^\omega P, W$. By Proposition 10 (*iii*), $\vdash^\omega P, \neg W$. But this implies that $\neg W \in [\![P]\!]$ and $\neg W \in [\![Q]\!]$ which contradicts Proposition 10 (*ii*). □

Before connecting the equivalence given by $[\![P]\!] = [\![Q]\!]$ to known equivalences, we need to define the notion of labeled transition. Let a be an action (that is, a constant of type i). The three place relation $P \stackrel{a}{\Longrightarrow} P'$ is defined to hold if $P' \mid 1 \vdash_{\pi_0} P \mid \bar{a}.1$. Here, the constant 1 is some anonymous symbol for which no inference rule or axiom is provided. (It is possible to identify 1 with the constant of the same name used in linear logic [Gir87] since the inference rules given for 1 there cannot be used in any cut-free π_0-proof of the sequent $P' \mid 1 \longrightarrow P \mid \bar{a}.1$.) Given this definition, it follows that if $P \stackrel{a}{\Longrightarrow} P'$ and Q is an agent expression then $P \mid \bar{a}.Q \Longrightarrow P' \mid Q$ (simply replace 1 with Q).

If Γ is a multiset of formulas then $|\Gamma$ denotes the parallel composition (using $|$) of all the formulas Γ in some fixed but arbitrary order.

Proposition 12 *Let Γ be a multiset of agents, let a be an action, and let $W \in C_\omega$. Then $\vdash^\omega \Gamma, \bar{a}.W$ if and only if there is a multiset of agents Ψ such that $|\Gamma \stackrel{a}{\Longrightarrow} |\Psi$ and $\vdash^\omega \Psi, W$.*

Proof. First assume that $\vdash^\omega \Gamma, \bar{a}.W$. A proof of $\longrightarrow \Gamma, \bar{a}.W$ must contain a subproof where the last inference rule is

$$\frac{\longrightarrow \Gamma', R, W}{\longrightarrow \Gamma', a.R, \bar{a}.W},$$

for some multiset of agents Γ' and some agent R. Set Ψ equal to the multiset $\Gamma' \cup \{R\}$. Now the sequent $(|\Psi) \mid 1 \longrightarrow \Gamma', R, 1$ clearly has a π_0-proof (involving only $|$L rules and initial sequents). If we now add to this sequent all the right rules that were applied to build the proof of $\longrightarrow \Gamma, a.W$ from $\longrightarrow \Gamma', R, W$, we can construct a proof of $(|\Psi) \mid 1 \longrightarrow \Gamma, \bar{a}.1$ and of $(|\Psi) \mid 1 \longrightarrow (|\Gamma) \mid \bar{a}.1$. Thus, $|\Gamma \stackrel{a}{\Longrightarrow} |\Psi$.

For the converse, assume that Ψ is such that $|\Gamma \stackrel{a}{\Longrightarrow} |\Psi$ and $\vdash^\omega \Psi, W$. The crossover sequent of the proof of $(|\Psi) \mid 1 \longrightarrow \Gamma, \bar{a}.1$ must be $(|\Psi) \mid 1 \longrightarrow \Gamma', 1$ for some multiset of agents Γ' (see Proposition 2). Since $(|\Psi)$ and $(|\Gamma')$ are equal agent expressions up to associativity and commutativity of $|$, $\vdash^\omega \Gamma', W$. Now applying to the sequent $\longrightarrow \Gamma', W$ all those right rules that were used to prove $(|\Psi) \mid 1 \longrightarrow (|\Gamma) \mid \bar{a}.1$ from $(|\Psi) \mid 1 \longrightarrow \Gamma', 1$ yields a proof of $\longrightarrow \Gamma, \bar{a}.W$. □

We can now show that co-agents act the same as formulas of the Hennessy-Milner modal logic. *Assertion formulas* are formulas containing the indexed conjunction $\wedge_{i \in I}$ (for a denumerable index set I), the possibility modal $\langle a \rangle$ (for a an action), and the negation $\neg$. The logical constant *true* is defined to be $\wedge_{i \in I} A_i$ for the empty index set I. The satisfaction of an assertion A by a process expression P, written as $P \models A$, is defined by the following induction on the structure of assertions.

- $P \models \wedge_{i \in I} A_i$ if $P \models A_i$ for every $i \in I$.
- $P \models \langle a \rangle A$ if there is an agent P' such that $P \stackrel{a}{\Longrightarrow} P'$ and $P' \models A$.
- $P \models \neg A$ if it is not the case that $P \models A$.

Define the following bijection of C_ω into assertion formulas: $(\&_{i \in I} W_i)^o = \wedge_{i \in I} W_i^o$, $(a.W)^o = \langle \bar{a} \rangle W^o$, and $(\neg W)^o = \neg W^o$.

Proposition 13 *Let $W \in C_\omega$ and let P be an agent expression. Then $\vdash^\omega P, W$ if and only if $P \models W^o$.*

Proof. This proof is by induction on the structure of co-agents in C_ω. The cases for $\&_{i \in I}$ and $\neg$ are immediate. Let W be $a.W'$. If $P \models \langle \bar{a} \rangle (W')^o$ then there is a P' such that $P \stackrel{\bar{a}}{\Longrightarrow} P'$ and $P' \models (W')^o$. By the inductive hypothesis, $\vdash^\omega P', W'$ and by Proposition 12, $\vdash^\omega P, W$. Conversely, if $\vdash^\omega P, a.W'$ then, by Proposition 12, there is a P' such that $P \stackrel{\bar{a}}{\Longrightarrow} P'$ and $\vdash^\omega P', W$. By the inductive hypothesis, $P' \models W^o$ and by the definition of $\models$, $P \models \langle \bar{a} \rangle (W')^o$. □

The following proposition is now immediate.

Proposition 14 *Let P and Q be agents. Then $[\![P]\!] = [\![Q]\!]$ if and only if for every assertion A, $P \models A$ if and only if $Q \models A$.*

Since the Hennessy-Milner logic characterizes observational equivalence, P and Q are observational equivalence if and only if $[\![P]\!] = [\![Q]\!]$. It is possible to show this result directly without making use of the Hennessy-Milner logic but the proof would be essentially identical to the proof using this modal logic.

This derivation of the Hennessy-Milner logic via co-agents is rather satisfactory for at least two reasons. First, it is possible to understand agents and assertion formulas as part of the same logical system, here a theory in a fragment of linear logic. Second, the intensional prefixing operator gives rise to the intensional modal operator: the latter does not need to be added separately. Representing prefixing as two non-logical constants of higher-order type might be considered one of the more controversial aspects of this representation. The fact that this choice also explains the modal operator provides us with more confidence in this choice.

In fact, the parsimony of our presentation of co-agents can be improved even further: negation $\neg$ is the only co-agent combinator that is necessary. For example, the expression $\neg nil$ can be used for $\top$ ($\neg P$ for any agent P will also do) and the expression $\neg(\neg W_1 + \neg W_2)$ can be used for $W_1 \,\&\, W_2$. Thus, if we admit indexed sums $\sum_{i \in I}$ into π_0, the only co-agent combinator we need to introduce is the negation-as-failure-to-prove operator.

Of course, there is a great deal of work left to be done in getting a full picture of the relationship between multiple-conclusion sequent calculus and process calculi. For example, Abramsky's work on bisimulation as testing equivalence [Abr87] should be related to the material just presented. A very good test of our approach would be to see if the modal logics recently described for the π-calculus in [MPW91] can be motivated using the notion of co-agents.

6 Conclusions

In this paper we have attempted to show one way that proof theory can be used to represent and organize the details of the π-calculus. This approach seems successful on more than one level: not only can the reflective and transitive closure of reduction be identified with entailed-by but also proof-theoretic notions of semantics provide a natural link to well studied semantics for concurrency. The derivation of the Hennessy-Milner modal logic via the notion of co-agents speaks strongly for the directness of this approach.

Acknowledgements. The anonymous reviewers of an earlier draft of this paper have suggested several improvements to presentation of this paper. The author has been funded in part by ONR N00014-88-K-0633, NSF CCR-91-02753, and DARPA N00014-85-K-0018.

References

[Abr87] Samson Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53:225–241, 1987.

[Abr90] Samson Abramsky. Computational interpretations of linear logic. Technical Report Research Report DOC 90/20, Imperial College, October 1990.

[Abr91] Samson Abramsky. Proofs as processes. Copy of transparencies, 1991.

[AP91] J.M. Andreoli and R. Pareschi. Linear objects: Logical processes with built-in inheritance. *New Generation Computing*, 9:3-4, 1991. (Special issue of papers selected from ICLP'90).

[Fel91] Amy Felty. A logic program for transforming sequent proofs to natural deduction proofs. In Peter Schroeder-Heister, editor, *Extensions of Logic Programming: International Workshop, Tübingen FRG, December 1989*, volume 475 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1991.

[Gen69] Gerhard Gentzen. Investigations into logical deductions, 1935. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland Publishing Co., Amsterdam, 1969.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Hen88] Matthew Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

[HM92] Joshua Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Journal of Information and Computation*, 1992. Invited to a special issue of papers from the 1991 LICS conference.

[Kle52] Stephen Cole Kleene. Permutabilities of inferences in gentzen's calculi LK and LJ. *Memoirs of the American Mathematical Society*, 10, 1952.

[Mil89] Dale Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, 6:79 – 108, 1989.

[Mil90] Robin Milner. Functions as processes. Research Report 1154, INRIA, 1990.

[Mil91] Robin Milner. The polyadic π-calculus: A tutorial. LFCS Report Series ECS-LFCS-91-180, University of Edinburgh, October 1991.

[Mil92] Dale Miller. Abstract syntax and logic programming. In *Logic Programming: Proceedings of the First and Second Russian Conferences on Logic Programming*, number 592 in Lecture Notes in Artificial Intelligence, pages 322–337. Springer-Verlag, 1992. Also available as technical report MS-CIS-91-72, UPenn.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[MPW89a] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, Part I. LFCS Report Series ECS-LFCS-89-85, University of Edinburgh, June 1989.

[MPW89b] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, Part II. LFCS Report Series ECS-LFCS-89-86, University of Edinburgh, June 1989.

[MPW91] Robin Milner, Joachim Parrow, and David Walker. Modal logics for mobile processes. LFCS Report Series ECS-LFCS-91-136, University of Edinburgh, April 1991.

[NM90] Gopalan Nadathur and Dale Miller. Higher-order Horn clauses. *Journal of the ACM*, 37(4):777 – 814, October 1990.

[Pot77] Garrel Pottinger. Normalization as a homomorphic image of cut-elimination. *Annals of Mathematical Logic*, 12(3):223–357, 1977.