

# Proof Search Specifications of Bisimulation and Modal Logics for the $\pi$ -calculus

Alwen Tiu

Logic and Computation Group

College of Engineering and Computer Science

The Australian National University

and

Dale Miller

INRIA-Saclay & LIX, École polytechnique

---

We specify the operational semantics and bisimulation relations for the finite  $\pi$ -calculus within a logic that contains the  $\nabla$  quantifier for encoding *generic judgments* and definitions for encoding fixed points. Since we restrict to the finite case, the ability of the logic to unfold fixed points allows this logic to be complete for both the inductive nature of operational semantics and the coinductive nature of bisimulation. The  $\nabla$  quantifier helps with the delicate issues surrounding the scope of variables within  $\pi$ -calculus expressions and their executions (proofs). We illustrate several merits of the logical specifications permitted by this logic: they are natural and declarative; they contain no side-conditions concerning names of variables while maintaining a completely formal treatment of such variables; differences between late and open bisimulation relations arise from familiar logic distinctions; the interplay between the three quantifiers ( $\forall$ ,  $\exists$ , and  $\nabla$ ) and their scopes can explain the differences between early and late bisimulation and between various modal operators based on bound input and output actions; and proof search involving the application of inference rules, unification, and backtracking can provide complete proof systems for one-step transitions, bisimulation, and satisfaction in modal logic. We also illustrate how one can encode the  $\pi$ -calculus with replications, in an extended logic with induction and co-induction.

Categories and Subject Descriptors: F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs—*Specification Techniques*; F.4.1 [Mathematical Logic and Formal Languages]: Mathematical Logic—*Proof Theory*

General Terms: Theory, Verification

Additional Key Words and Phrases: proof search,  $\lambda$ -tree syntax,  $\nabla$  quantifier, generic judgments, higher-order abstract syntax,  $\pi$ -calculus, bisimulation, modal logics

---

Authors' addresses: A. Tiu, Logic and Computation Group, College of Engineering and Computer Science, Building 115, The Australian National University, Canberra, ACT 0200, Australia; D. Miller, Laboratoire d'Informatique (LIX), École Polytechnique, Rue de Saclay, 91128 Palaiseau Cedex, France.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 0000-0000/YY/00-0001 \$5.00

## 1. INTRODUCTION

We present formal specifications of various aspects of the  $\pi$ -calculus, including its syntax, operational semantics, bisimulation relations, and modal logics. We shall do this by using the  $FO\lambda^{\Delta\nabla}$  logic [Miller and Tiu 2005]. We provide a high-level introduction to this logic here before presenting more technical aspects of it in the next section.

Just as it is common to use meta-level application to represent object-level application (for example, the encoding of  $P + Q$  is via the meta-level application of the encoding for plus to the encoding of its two arguments), we shall use meta-level  $\lambda$ -abstractions to encode object-level abstractions. The term *higher-order abstract syntax* (HOAS) [Pfenning and Elliott 1988] is commonly used to describe this approach to mapping object-level abstractions into some meta-level abstractions. Of course, the nature of the resulting encodings varies as one varies the meta-level. For example, if the meta-level is a higher-order functional programming language or a higher-order type theory, the usual abstraction available constructs function spaces. In this case, HOAS maps object-level abstractions to semantically rich function spaces: determining whether or not two syntactic objects are equal is then mapped to the question of determining if two functions are equal (typically, an undecidable judgment). In such a setting, HOAS is less about syntax and more about a particular mathematical denotation of the syntax. In this paper, we start with an intuitionistic subset of the Simple Theory of Types [Church 1940] that does not contain the *mathematical* axioms of extensionality, description, choice, and infinity. In this setting,  $\lambda$ -abstraction is not strong enough to denote general computable functions and equality of  $\lambda$ -terms is decidable. As a result, this weaker logic provides term-level bindings that can be used to encode syntax with bindings. This style of describing syntax via a meta-logic containing a weak form of  $\lambda$ -abstraction has been called the  *$\lambda$ -tree syntax* [Miller 2000] approach to HOAS in order to distinguish it from the approaches that use function spaces. The  $\lambda$ -tree syntax approach to encoding expressions is an old one (*cf.* [Huet and Lang 1978; Miller and Nadathur 1986; 1987; Paulson 1986]) and is used in specifications written in the logic programming languages  $\lambda$ Prolog [Nadathur and Miller 1988] and Twelf [Pfenning and Schürmann 1999].

Following Church, we shall use  $\lambda$ -abstractions to encode both *term-level abstractions* and *formula-level abstractions* (*e.g.*, quantifiers). The computational aspects of the  $\pi$ -calculus are usually specified via structured operational semantics [Plotkin 1981]: here, such specifications are encoded directly as inference rules and proofs over primitive relational judgments (*e.g.*, one-step transitions). As a result, a formal account of the interaction of binding in syntax and binding in computation leads to notions of *proof-level abstractions*. One such binding is the familiar *eigenvariable* abstraction of [Gentzen 1969] used to encode a universally quantified variable that has scope over an entire sequent. A second proof-level binding was introduced in [Miller and Tiu 2005] to capture a notion of *generic judgment*: this proof-level binding has a scope over individual entries within a sequent and is closely associated with the formula-level binding introduced by the  $\nabla$ -quantifier. A major goal of this paper is to illustrate how the  $\nabla$ -quantifier and this second proof-level abstraction can be used to specify and reason about computation: the  $\pi$ -calculus has been

chosen, in part, because it is a small calculus in which bindings play an important role in computation.

A reading of the truth condition for  $\nabla x_\gamma.Bx$  is something like the following: this formula is true if  $Bx$  is true for the new element  $x$  of type  $\gamma$ . In particular, the formula  $\nabla x_\gamma \nabla y_\gamma. x \neq y$  is a theorem regardless of the intended interpretation of the domain  $\gamma$  since the bindings for  $x$  and  $y$  are distinct. In contrast, the truth value of the formula  $\forall x_\gamma \forall y_\gamma. x \neq y$  is dependent on the domain  $\gamma$ : this quantified inequality is true if and only if the interpretation of  $\gamma$  is empty.

The  $FO\lambda^{\Delta\nabla}$  logic is based on intuitionistic logic, a weaker logic than classical logic. One of the principles missing from intuitionistic logic is that of the excluded middle: that is,  $A \vee \neg A$  is not generally provable in intuitionistic logic. Consider, for example, the following formula concerning the variable  $w$ :

$$\forall x_\gamma [x = w \vee x \neq w]. \quad (*)$$

In classical logic, this formula is a trivial theorem. From a constructive point-of-view, it might not be desirable to admit this formula as a theorem in some cases. If the type of quantification  $\gamma$  is a conventional (closed) first-order datatype, then we might expect to have a decision procedure for equality. For example, if  $\gamma$  is the type for lists, then it is a simple matter to construct a procedure that decides whether or not two members of  $\gamma$  are equal by considering the top constructor of the list and, in the event of comparing two non-empty lists, making a recursive call (assuming a decision procedure is available for the elements of the list). In fact, it is possible to prove in an intuitionistic logic augmented with induction (see, for example, [Tiu 2004]) the formula (\*) for closed, first-order datatypes.

If the type  $\gamma$  is not given inductively, as is the usual case for names in intuitionistic formalizations of the  $\pi$ -calculus (see [Despeyroux 2000] and below), then the corresponding instance of (\*) is not provable. Thus, whether or not we allow instances of (\*) to be assumed can change the nature of a specification. In fact, we show in Section 5, that if we add to our specification of *open bisimulation* [Sangiorgi 1996] assumptions corresponding to (\*), then we get a specification of *late bisimulation*. If we were working with a classical logic, such a declarative presentation of these two bisimulations would not be so easy to describe.

The authors first presented the logic used in this paper in [Miller and Tiu 2003] and illustrated its usefulness with the  $\pi$ -calculus: in particular, the specifications of one-step transitions in Figure 2 and of late bisimulation in Figure 3 also appear in [Miller and Tiu 2003] but without proof. In this paper, we state the formal properties of our specifications, provide a specification of late bisimulation, and provide a novel comparison between open and late bisimulation. In particular, we show that the difference between open and late bisimulation (apart from the difference that arises from the use of types defined inductively or not) can be captured by the different quantification of free names using  $\forall$  and  $\nabla$ . We show in Section 5 that a natural class of name *distinctions* can be captured by the alternation of  $\forall$  and  $\nabla$  quantifiers and, in the case where we are interested only in checking open bisimilarity modulo the empty distinction, the notion of distinction that arises in the process of checking bisimilarity is completely subsumed by quantifier alternation. In Section 6 we show that “modal logics for mobility” can easily be handled as well and present, for the first time, a modal characterization of open bisimulation.

Since our focus in this paper is on names, scoping of names, dependency of names, and distinction of names, we have chosen to focus on the finite  $\pi$ -calculus. The treatment of the  $\pi$ -calculus with replication is presented in Section 7 through an example. In Section 8 we outline the automation of proof search based on these specifications: when such automation is applied to our specification of open bisimulation, a symbolic bisimulation procedure arises. In Section 9 we present some related and future work and Section 10 concludes the paper. In order to improve the readability of the main part of the paper, numerous technical proofs have been moved to an electronic appendix.

Parts of this paper, in their preliminary forms and without proofs, have been presented in [Tiu and Miller 2004; Tiu 2005]: in particular, the material on encoding bisimulations (Section 5) corresponds to [Tiu and Miller 2004] and the material on encoding modal logics for the  $\pi$ -calculus (Section 6) corresponds to [Tiu 2005].

## 2. OVERVIEW OF THE LOGIC

This paper is about the use of a certain logic to specify and reason about computation. We shall assume that the reader is not interested in an in-depth analysis of the logic but with its application. We state the most relevant results we shall need about this logic in order to reason about our  $\pi$ -calculus specifications. The reader who is interested in more details about this logic is referred to [Tiu 2004] and [Miller and Tiu 2005].

At the core of the logic  $FO\lambda^{\Delta\nabla}$  (pronounced “fold-nabla”) is a first-order logic for  $\lambda$ -terms (hence, the prefix  $FO\lambda$ ) that is the result of extending Gentzen’s LJ sequent calculus for first-order intuitionistic logic [Gentzen 1969] with simply typed  $\lambda$ -terms and with quantifiers that range over non-predicate types. The full logic is the result of making two extensions to this core. First, “fixed points” are added via the technical device of “definitions,” presented below and marked with the symbol  $\hat{=}$ . Fixed points can capture important forms of “must behavior” in the treatment of operational semantics [McDowell and Miller 2000; McDowell et al. 2003]. Fixed points also strengthen negation to encompass “negation-as-finite-failure.” In the presence of this stronger negation, the usual treatment of  $\lambda$ -tree syntax via “generic judgments” encoded as universal quantifiers is inadequate: a more intensional treatment of such judgments is provided by the addition of the  $\nabla$ -quantifier [Miller and Tiu 2005].

A *sequent* is an expression of the form  $B_1, \dots, B_n \multimap B_0$  where  $B_0, \dots, B_n$  are formulas and the elongated turnstile  $\multimap$  is the sequent arrow. To the left of the turnstile is a multiset: thus repeated occurrences of a formula are allowed. If the formulas  $B_0, \dots, B_n$  contain free variables, they are considered universally quantified outside the sequent, in the sense that if the above sequent is provable then every instance of it is also provable. In proof theoretical terms, such free variables are called *eigenvariables*.

A first attempt at using sequent calculus to capture judgments about the  $\pi$ -calculus could be to use eigenvariables to encode names in the  $\pi$ -calculus, but this is certainly problematic. For example, if we have a proof of the sequent  $\multimap Pxy$ , where  $x$  and  $y$  are different eigenvariables, then logic dictates that the sequent  $\multimap Pzz$  is also provable (given the universal quantifier reading of eigenvariables). If

the judgment  $P$  is about, say, bisimulation, then it is not likely that a statement about bisimulation involving two different names  $x$  and  $y$  remains true if they are identified to the same name  $z$ .

To address this problem, the logic  $FO\lambda^{\Delta\nabla}$  extends sequents with a new notion of “local scope” for proof-level bound variables (originally motivated in [Miller and Tiu 2003] to encode “generic judgments”). In particular, sequents in  $FO\lambda^{\Delta\nabla}$  are of the form

$$\Sigma; \sigma_1 \triangleright B_1, \dots, \sigma_n \triangleright B_n \multimap \sigma_0 \triangleright B_0$$

where  $\Sigma$  is a *global signature*, *i.e.*, the set of eigenvariables whose scope is over the entire sequent, and  $\sigma_i$  is a *local signature*, *i.e.*, a list of variables scoped over  $B_i$ . We shall consider sequents to be binding structures in the sense that the signatures, both the global and local ones, are abstractions over their respective scopes. The variables in  $\Sigma$  and  $\sigma_i$  will admit  $\alpha$ -conversion by systematically changing the names of variables in signatures as well as those in their scope, following the usual convention of the  $\lambda$ -calculus. The meaning of eigenvariables is as before except that now instantiation of eigenvariables has to be capture-avoiding with respect to the local signatures. The variables in local signatures act as locally scoped *generic constants*: that is, they do not vary in proofs since they will not be instantiated. The expression  $\sigma \triangleright B$  is called a *generic judgment* or simply a *judgment*. We use script letters  $\mathcal{A}$ ,  $\mathcal{B}$ , *etc* to denote judgments. We write simply  $B$  instead of  $\sigma \triangleright B$  if the signature  $\sigma$  is empty. We shall often write the list  $\sigma$  as a string of variables: *e.g.*, a judgment  $(x_1, x_2, x_3) \triangleright B$  will be written as  $x_1x_2x_3 \triangleright B$ . If the list  $x_1, x_2, x_3$  is known from context we shall also abbreviate the judgment as  $\bar{x} \triangleright B$ .

Following Church [1940], the type  $o$  is used to denote the type of formulas. The propositional constants of  $FO\lambda^{\Delta\nabla}$  are  $\wedge$  (conjunction),  $\vee$  (disjunction),  $\supset$  (implication),  $\top$  (true) and  $\perp$  (false). We shall abbreviate  $B \supset \perp$  as  $\neg B$  (intuitionistic negation). Syntactically, logical constants can be seen as typed constants: for example, the binary connectives have type  $o \rightarrow o \rightarrow o$ . For each simple type  $\gamma$  that does not contain  $o$ , there are three quantifiers in  $FO\lambda^{\Delta\nabla}$ : namely,  $\forall_\gamma$  (universal quantifier),  $\exists_\gamma$  (existential quantifier),  $\nabla_\gamma$  (nabla), each one of type  $(\gamma \rightarrow o) \rightarrow o$ . The subscript type  $\gamma$  is often dropped when it can be inferred from context or its value is not important. Since we do not allow quantification over predicates, this logic is proof-theoretically similar to first-order logic. The inference rules for  $FO\lambda^{\Delta\nabla}$  that do not deal with definitions are given in Figure 1.

During the search for proofs (reading rules bottom up), inference rules for  $\forall$  and  $\exists$  quantifier place new variables (eigenvariables) into the global signature while the inference rules for  $\nabla$  place new variables into a local signature. In the  $\forall\mathcal{R}$  and  $\exists\mathcal{L}$  rules, *raising* [Miller 1992] is used when replacing the bound variable  $x$  (which can be substituted for by terms containing variables in both the global signature and the local signature  $\sigma$ ) with the variable  $h$  (which can only be instantiated with terms containing variables in the global signature). In order not to miss substitution terms, the variable  $x$  is replaced by the term  $(h x_1 \dots x_n)$ : the latter expression is written simply as  $(h \sigma)$  where  $\sigma$  is the list  $x_1, \dots, x_n$ . As is usual, the eigenvariable  $h$  must not be free in the lower sequent of these rules. In  $\forall\mathcal{L}$  and  $\exists\mathcal{R}$ , the term  $t$  can have free variables from both  $\Sigma$  and  $\sigma$ , a fact that is given by the typing judgment  $\Sigma, \sigma \vdash t : \tau$ . The  $\nabla\mathcal{L}$  and  $\nabla\mathcal{R}$  rules have the proviso that  $y$  is not free

$$\begin{array}{c}
\frac{}{\Sigma; \sigma \triangleright B, \Gamma \vdash \sigma \triangleright B} \textit{init} \quad \frac{\Sigma; \Delta \vdash \mathcal{B} \quad \Sigma; \mathcal{B}, \Gamma \vdash \mathcal{C}}{\Sigma; \Delta, \Gamma \vdash \mathcal{C}} \textit{cut} \\
\frac{\Sigma; \sigma \triangleright B, \sigma \triangleright C, \Gamma \vdash \mathcal{D}}{\Sigma; \sigma \triangleright B \wedge C, \Gamma \vdash \mathcal{D}} \wedge \mathcal{L} \quad \frac{\Sigma; \Gamma \vdash \sigma \triangleright B \quad \Sigma; \Gamma \vdash \sigma \triangleright C}{\Sigma; \Gamma \vdash \sigma \triangleright B \wedge C} \wedge \mathcal{R} \\
\frac{\Sigma; \sigma \triangleright B, \Gamma \vdash \mathcal{D} \quad \Sigma; \sigma \triangleright C, \Gamma \vdash \mathcal{D}}{\Sigma; \sigma \triangleright B \vee C, \Gamma \vdash \mathcal{D}} \vee \mathcal{L} \quad \frac{\Sigma; \Gamma \vdash \sigma \triangleright B}{\Sigma; \Gamma \vdash \sigma \triangleright B \vee C} \vee \mathcal{R} \\
\frac{}{\Sigma; \sigma \triangleright \perp, \Gamma \vdash \mathcal{B}} \perp \mathcal{L} \quad \frac{\Sigma; \Gamma \vdash \sigma \triangleright C}{\Sigma; \Gamma \vdash \sigma \triangleright B \vee C} \vee \mathcal{R} \\
\frac{\Sigma; \Gamma \vdash \sigma \triangleright B \quad \Sigma; \sigma \triangleright C, \Gamma \vdash \mathcal{D}}{\Sigma; \sigma \triangleright B \supset C, \Gamma \vdash \mathcal{D}} \supset \mathcal{L} \quad \frac{\Sigma; \sigma \triangleright B, \Gamma \vdash \sigma \triangleright C}{\Sigma; \Gamma \vdash \sigma \triangleright B \supset C} \supset \mathcal{R} \\
\frac{\Sigma, \sigma \vdash t : \gamma \quad \Sigma; \sigma \triangleright B[t/x], \Gamma \vdash \mathcal{C}}{\Sigma; \sigma \triangleright \forall \gamma x. B, \Gamma \vdash \mathcal{C}} \forall \mathcal{L} \quad \frac{\Sigma, h; \Gamma \vdash \sigma \triangleright B[(h \sigma)/x]}{\Sigma; \Gamma \vdash \sigma \triangleright \forall x. B} \forall \mathcal{R} \\
\frac{\Sigma, h; \sigma \triangleright B[(h \sigma)/x], \Gamma \vdash \mathcal{C}}{\Sigma; \sigma \triangleright \exists x. B, \Gamma \vdash \mathcal{C}} \exists \mathcal{L} \quad \frac{\Sigma, \sigma \vdash t : \gamma \quad \Sigma; \Gamma \vdash \sigma \triangleright B[t/x]}{\Sigma; \Gamma \vdash \sigma \triangleright \exists \gamma x. B} \exists \mathcal{R} \\
\frac{\Sigma; (\sigma, y) \triangleright B[y/x], \Gamma \vdash \mathcal{C}}{\Sigma; \sigma \triangleright \nabla x B, \Gamma \vdash \mathcal{C}} \nabla \mathcal{L} \quad \frac{\Sigma; \Gamma \vdash (\sigma, y) \triangleright B[y/x]}{\Sigma; \Gamma \vdash \sigma \triangleright \nabla x B} \nabla \mathcal{R} \\
\frac{\Sigma; \mathcal{B}, \mathcal{B}, \Gamma \vdash \mathcal{C}}{\Sigma; \mathcal{B}, \Gamma \vdash \mathcal{C}} c\mathcal{L} \quad \frac{\Sigma; \Gamma \vdash \mathcal{C}}{\Sigma; \mathcal{B}, \Gamma \vdash \mathcal{C}} w\mathcal{L} \quad \frac{}{\Sigma; \Gamma \vdash \sigma \triangleright \top} \top \mathcal{R}
\end{array}$$

Fig. 1. The inference rules of  $FO\lambda^{\Delta\nabla}$  not dealing with definitions.

in  $\nabla x B$ . The introduction rules for propositional connectives are the standard ones for intuitionistic logic. Reading the rules top down, the structural rule  $c\mathcal{L}$  (contraction) allows removal of duplicate judgments from the sequent and the rule  $w\mathcal{L}$  (weakening) allows introduction of a (possibly new) judgment into the sequent. Note that since the initial rule  $init$  has implicit weakening, the weakening rule  $w\mathcal{L}$  can actually be shown admissible, hence it is strictly speaking not necessary. It is, however, convenient for interactive proof search, since it allows one to remove irrelevant formulae (reading the rule bottom up) in a sequent.

While sequent calculus introduction rules generally only introduce logical connectives, the full logic  $FO\lambda^{\Delta\nabla}$  additionally allows introduction of atomic judgments; that is, judgments which do not contain any occurrences of logical constants. To each atomic judgment,  $\mathcal{A}$ , we associate a defining judgment,  $\mathcal{B}$ , the *definition* of  $\mathcal{A}$ . The introduction rule for the judgment  $\mathcal{A}$  is in effect done by replacing  $\mathcal{A}$  with  $\mathcal{B}$  during proof search. This notion of definitions is an extension of work by Schroeder-Heister [1993], Eriksson [1991], Girard [1992], Stärk [1994], and McDowell and Miller [2000]. These inference rules for definitions allow for modest reasoning about the fixed points of (recursive) definitions.

**DEFINITION 1.** A *definition clause* is written  $\forall \bar{x}[p\bar{t} \triangleq B]$ , where  $p$  is a predicate constant, every free variable of the formula  $B$  is also free in at least one term in the list  $\bar{t}$  of terms, and all variables free in  $p\bar{t}$  are contained in the list  $\bar{x}$  of variables. The atomic formula  $p\bar{t}$  is called the *head* of the clause, and the formula  $B$  is called the *body*. The symbol  $\triangleq$  is used simply to indicate a definitional clause: it is not a

logical connective.

Let  $\forall_{\tau_1} x_1 \dots \forall_{\tau_n} x_n. H \triangleq B$  be a definition clause. Let  $y_1, \dots, y_m$  be a list of variables of types  $\alpha_1, \dots, \alpha_m$ , respectively. The *raised definition clause* of  $H$  with respect to the signature  $\{y_1 : \alpha_1, \dots, y_m : \alpha_m\}$  is defined as

$$\forall h_1 \dots \forall h_n. \bar{y} \triangleright H\theta \triangleq \bar{y} \triangleright B\theta$$

where  $\theta$  is the substitution  $[(h_1 \bar{y})/x_1, \dots, (h_n \bar{y})/x_n]$  and  $h_i$  is of type  $\alpha_1 \rightarrow \dots \rightarrow \alpha_m \rightarrow \tau_i$ , for every  $i \in \{1, \dots, n\}$ . A *definition* is a set of definition clauses together with their raised clauses.

Recall that we use script letters, such as  $\mathcal{B}$ ,  $\mathcal{H}$ , etc., to refer to generic judgments. In particular, in referring to a raised definition clause, e.g.,

$$\forall h_1 \dots \forall h_n. \bar{y} \triangleright H\theta \triangleq \bar{y} \triangleright B\theta$$

we shall sometimes simply write  $\mathcal{H} \triangleq \mathcal{B}$  when the local signatures can be inferred from context or are unimportant to the discussion.

To guarantee the consistency (and cut-elimination) of the logic  $FO\lambda^{\Delta\nabla}$ , we need some kind of stratification of definition so as to avoid a situation where a definition of a predicate depends negatively on itself. For this purpose, we associate to each predicate  $p$  a natural number  $\text{lvl}(p)$ , the *level* of  $p$ . The notion of level is generalized to formulas as follows.

**DEFINITION 2.** Given a formula  $B$ , its *level*  $\text{lvl}(B)$  is defined as follows:

- (1)  $\text{lvl}(p\bar{t}) = \text{lvl}(p)$
- (2)  $\text{lvl}(\perp) = \text{lvl}(\top) = 0$
- (3)  $\text{lvl}(B \wedge C) = \text{lvl}(B \vee C) = \max(\text{lvl}(B), \text{lvl}(C))$
- (4)  $\text{lvl}(B \supset C) = \max(\text{lvl}(B) + 1, \text{lvl}(C))$
- (5)  $\text{lvl}(\forall x. B) = \text{lvl}(\nabla x. B) = \text{lvl}(\exists x. B) = \text{lvl}(B)$ .

We shall require that for every definition clause  $\forall \bar{x}[p\bar{t} \triangleq B]$ ,  $\text{lvl}(B) \leq \text{lvl}(p)$ .

Note that the stratification condition above implies that in a stratified definition, say  $\forall \bar{x}[p\bar{t} \triangleq B]$ , the predicate  $p$  can only occur strictly positively in  $B$  (if it occurs at all). All definitions considered in this paper can be easily stratified according to the above definition and cut-elimination holds for the logic using them. For the latter, we refer the reader to [Miller and Tiu 2005] for the full details.

The introduction rules for a defined judgment are as follows. When applying the introduction rules, we shall omit the outer quantifiers in a definition clause and assume implicitly that the free variables in the definition clause are distinct from other variables in the sequent.

$$\frac{\{\Sigma\theta; \mathcal{B}\theta, \Gamma\theta \vdash C\theta \mid \theta \in CSU(\mathcal{A}, \mathcal{H}) \text{ for some raised clause } \mathcal{H} \triangleq \mathcal{B}\}}{\Sigma; \mathcal{A}, \Gamma \vdash C} \text{ def}\mathcal{L}$$

$$\frac{\Sigma; \Gamma \vdash \mathcal{B}\theta}{\Sigma; \Gamma \vdash \mathcal{A}} \text{ def}\mathcal{R}, \quad \text{where } \mathcal{H} \triangleq \mathcal{B} \text{ is a raised definition clause and } \mathcal{H}\theta = \mathcal{A}$$

In the above rules, we apply substitution to judgments. The result of applying a substitution  $\theta$  to a generic judgment  $x_1, \dots, x_n \triangleright B$ , written as  $(x_1, \dots, x_n \triangleright B)\theta$ , is

$y_1, \dots, y_n \triangleright B'$ , if  $(\lambda x_1 \dots \lambda x_n. B)\theta$  is equal (modulo  $\lambda$ -conversion) to  $\lambda y_1 \dots \lambda y_n. B'$ . If  $\Gamma$  is a multiset of generic judgments, then  $\Gamma\theta$  is the multiset  $\{J\theta \mid J \in \Gamma\}$ . In the  $\text{def}\mathcal{L}$  rule, we use the notion of *complete set of unifiers* (CSU) [Huet 1975]. We denote by  $CSU(\mathcal{A}, \mathcal{H})$  the complete set of unifiers for the pair  $(\mathcal{A}, \mathcal{H})$ : that is, for any substitution  $\theta$  such that  $\mathcal{A}\theta = \mathcal{H}\theta$ , there is a substitution  $\rho \in CSU(\mathcal{A}, \mathcal{H})$  such that  $\theta = \rho \circ \theta'$  for some substitution  $\theta'$ . In all the applications of  $\text{def}\mathcal{L}$  in this paper, the set  $CSU(\mathcal{A}, \mathcal{H})$  is either empty (the two judgments are not unifiable) or contains a single substitution denoting the most general unifier. The signature  $\Sigma\theta$  in  $\text{def}\mathcal{L}$  denotes a signature obtained from  $\Sigma$  by removing the variables in the domain of  $\theta$  and adding the variables in the range of  $\theta$ . In the  $\text{def}\mathcal{L}$  rule, reading the rule bottom-up, eigenvariables can be instantiated in the premise, while in the  $\text{def}\mathcal{R}$  rule, eigenvariables are not instantiated. The set that is the premise of the  $\text{def}\mathcal{L}$  rule means that that rule instance has a premise for every member of that set: if that set is empty, then the premise is proved.

Equality for terms can be defined in  $FO\lambda^{\Delta\nabla}$  using the single definition clause  $[\forall x. x = x \triangleq \top]$ . Specializing the  $\text{def}\mathcal{L}$  and  $\text{def}\mathcal{R}$  rules to equality yields the inference rules

$$\frac{\{\Sigma\theta; \Gamma\theta \vdash \mathcal{C}\theta \mid \theta \in CSU(\lambda\bar{y}.s, \lambda\bar{y}.t)\}}{\Sigma; \bar{y} \triangleright s = t, \Gamma \vdash \mathcal{C}} \quad \frac{}{\Sigma; \Gamma \vdash \bar{y} \triangleright t = t}$$

Disequality  $s \neq t$ , the negation of equality, is an abbreviation for  $(s = t) \supset \perp$ .

One might find the following analogy with logic programming helpful: if a definition is viewed as a logic program, then the  $\text{def}\mathcal{R}$  rule captures backchaining and the  $\text{def}\mathcal{L}$  rule corresponds to *case analysis* on all possible ways an atomic judgment could be proved. In the case where the program has only finitely many computation paths, we can effectively encode *negation-as-failure* using  $\text{def}\mathcal{L}$  [Hallnäs and Schroeder-Heister 1991].

### 3. SOME META-THEORY OF THE LOGIC

Once we have written a computational specification as logical formulas, it is important that the underlying logic has formal properties that allow us to reason about that specification. In this section, we list a few formal properties of  $FO\lambda^{\Delta\nabla}$  that will be used later in this paper.

Cut-elimination for  $FO\lambda^{\Delta\nabla}$  [Miller and Tiu 2005; Tiu 2004] is probably the single most important meta-theoretic property needed. Beside guaranteeing the consistency of the logic, it also provides completeness for *cut-free* proofs: these proofs are used to help prove the *adequacy* of a logical specification. For example, the proof that a certain specification actually encodes the one-step transition relation or the bisimulation relation starts by examining the highly restricted structure of cut-free proofs. Also, cut-elimination allows use of modus ponens and substitutions into cut-free proofs and to be assured that another cut-free proof arises from that operation.

Another important structural property of provability is the *invertibility* of inference rules. An inference rule of logic is *invertible* if the provability of the conclusion implies the provability of the premise(s) of the rule. The following rules in  $FO\lambda^{\Delta\nabla}$  are invertible:  $\wedge\mathcal{R}, \wedge\mathcal{L}, \vee\mathcal{L}, \supset\mathcal{R}, \forall\mathcal{R}, \exists\mathcal{L}, \text{def}\mathcal{L}$  (see [Tiu 2004] for a proof). Knowing the invertibility of a rule can be useful in determining some structure of a proof.

For example, if we know that a sequent  $A \vee B, \Gamma \vdash C$  is provable, then by the invertibility of  $\vee\mathcal{L}$ , we know that it must be the case that  $A, \Gamma \vdash C$  and  $B, \Gamma \vdash C$  are provable.

We now present several meta-theoretic properties of provability that are specifically targeted at the  $\nabla$ -quantifier. These properties are useful when proving the adequacy of our specifications of bisimulation and modal logic in the following sections. These properties also provide some insights into the differences between the universal and the  $\nabla$  quantifiers. The proofs of the propositions listed in this section can be found in [Tiu 2004].

Throughout the paper, we shall use the following notation for provability: We shall write  $\vdash \Sigma; \Gamma \vdash C$  to denote the fact the sequent  $\Sigma; \Gamma \vdash C$  is provable, and  $\vdash B$  to denote provability of the sequent  $.; \vdash B$ .

The following proposition states that the global scope of an eigenvariable can be weakened to be a locally scoped variable when there are no assumptions.

**PROPOSITION 3.** *If  $\vdash \forall xB$  then  $\vdash \nabla xB$ .*

Notice that the implication  $\forall_\tau xB \supset \nabla_\tau xB$  does not necessarily hold. For example, if the type  $\tau$  is empty, then  $\forall_\tau xB$  may be true vacuously, independently of the structure of  $B$ , whereas attempting to prove  $\nabla xB$  reduces to attempting to prove  $B$  given the fresh element  $x$  of type  $\tau$ .

As we suggested in Section 1 with the formula  $\nabla x_\gamma \nabla y_\gamma. x \neq y$ , the converse of Proposition 3 is not true in general. That converse does hold, however, if we use definitions and formulas that do not contain implications and, consequently, do not contain negations (since these are formally defined as implications). Horn clauses provide an interesting fragment of logic that does not contain negations: in that setting, the distinction between  $\nabla$  and  $\forall$  cannot be observed using the proof system. More precisely, let  $\text{hc}^{\nabla\forall}$ -formulas (for Horn clauses formulas with  $\forall$  and  $\nabla$ ) be a formulas that do not contain occurrences of the logical constant  $\supset$  (implication). A  $\text{hc}^{\nabla\forall}$ -definition is a definition whose bodies are  $\text{hc}^{\nabla\forall}$ -formulas. For example, the definition of the one-step transition in Figure 2 is an  $\text{hc}^{\nabla\forall}$ -definition but the definition of bisimulation in Figure 3 is not a  $\text{hc}^{\nabla\forall}$ -definition.

**PROPOSITION 4.** *Let  $\mathcal{D}$  be a  $\text{hc}^{\nabla\forall}$ -definition and  $\forall xG$  be a  $\text{hc}^{\nabla\forall}$ -formula. Then, assuming  $\mathcal{D}$  is the only definition used,  $\forall xG$  is provable if and only if  $\nabla xG$  is provable.*

The above proposition highlights the fact that positive occurrences of  $\nabla$  are interchangeable with  $\forall$ . The specification of the operational semantics of the  $\pi$ -calculus in the next section uses only positive occurrences of  $\nabla$ , hence its specification can be done also in a logic without  $\nabla$ . However, our specifications of bisimulation and modal logics in the subsequent sections make use of implications in definitions and, as a result,  $\nabla$  cannot be replaced with  $\forall$ . We shall come back to this discussion on the distinction between  $\nabla$  and  $\forall$  when we present the specification of bisimulation in Section 5.

Finally, we state a technical result about proofs in  $FO\lambda^{\Delta\nabla}$  that states that provability of a sequent is not affected by the application of substitutions.

**PROPOSITION 5.** *Let  $\Pi$  be a proof of  $\Sigma; \Gamma \vdash C$ . Then for any substitution  $\theta$ ,*

there exists a proof  $\Pi'$  of  $\Sigma\theta; \Gamma\theta \vdash C\theta$  such that the height of proof of  $\Pi'$  is less than or equal to the height of  $\Pi$ .

#### 4. LOGICAL SPECIFICATION OF ONE-STEP TRANSITION

The finite  $\pi$ -calculus is the fragment of the  $\pi$ -calculus without recursion (or replication). In particular, process expressions are defined as

$$P ::= 0 \mid \bar{x}y.P \mid x(y).P \mid \tau.P \mid (x)P \mid [x = y]P \mid P|P \mid P + P.$$

We use the symbols  $P, Q, R, S, T$  to denote processes and lower case letters, *e.g.*,  $a, b, c, d, x, y, z$  to denote names. The occurrence of  $y$  in the processes  $x(y).P$  and  $(y)P$  is a binding occurrence with  $P$  as its scope. The set of free names in  $P$  is denoted by  $\text{fn}(P)$ , the set of bound names is denoted by  $\text{bn}(P)$ . We write  $\text{n}(P)$  for the set  $\text{fn}(P) \cup \text{bn}(P)$ . We consider processes to be equivalent if they are identical up to a renaming of bound variables.

The relation of one-step (late) transition [Milner et al. 1992] for the  $\pi$ -calculus is denoted by  $P \xrightarrow{\alpha} Q$ , where  $P$  and  $Q$  are processes and  $\alpha$  is an action. The kinds of actions are *the silent action*  $\tau$ , *the free input action*  $xy$ , *the free output action*  $\bar{x}y$ , *the bound input action*  $x(y)$ , and *the bound output action*  $\bar{x}(y)$ . The name  $y$  in  $x(y)$  and  $\bar{x}(y)$  is a binding occurrence. Just as we did with processes, we use  $\text{fn}(\alpha)$ ,  $\text{bn}(\alpha)$  and  $\text{n}(\alpha)$  to denote free names, bound names, and names in  $\alpha$ . An action without binding occurrences of names is a *free action* (this includes the silent action); otherwise it is a *bound action*.

Three primitive syntactic categories are used to encode the  $\pi$ -calculus into  $\lambda$ -tree syntax:  $n$  for names,  $p$  for processes, and  $a$  for actions. We do not assume any inhabitants of type  $n$ : as a consequence, a free name is translated to a variable of type  $n$  that is either universally or  $\nabla$ -quantified, depending on whether we want to allow names to be instantiated or not. For instance, when encoding late bisimulation, free names correspond to  $\nabla$ -quantified variables, while when encoding open bisimulation, free names correspond to universally quantified variables (Section 5). Since the rest of this paper is about the  $\pi$ -calculus, the  $\nabla$  quantifier will from now on only be used at type  $n$ .

There are three constructors for actions:  $\tau : a$  (for the silent action) and the two constants  $\downarrow$  and  $\uparrow$ , both of type  $n \rightarrow n \rightarrow a$  (for building input and output actions, respectively). The free output action  $\bar{x}y$ , is encoded as  $\uparrow xy$  while the bound output action  $\bar{x}(y)$  is encoded as  $\lambda y (\uparrow xy)$  (or the  $\eta$ -equivalent term  $\uparrow x$ ). The free input action  $xy$ , is encoded as  $\downarrow xy$  while the bound input action  $x(y)$  is encoded as  $\lambda y (\downarrow xy)$  (or simply  $\downarrow x$ ). Notice that bound input and bound output actions have type  $n \rightarrow a$  instead of  $a$ .

The following are process constructors, where  $+$  and  $|$  are written as infix:

$$\begin{aligned} 0 : p & \quad \tau : p \rightarrow p & \quad out : n \rightarrow n \rightarrow p \rightarrow p & \quad in : n \rightarrow (n \rightarrow p) \rightarrow p \\ + : p \rightarrow p \rightarrow p & \quad | : p \rightarrow p \rightarrow p & \quad match : n \rightarrow n \rightarrow p \rightarrow p & \quad \nu : (n \rightarrow p) \rightarrow p \end{aligned}$$

Notice  $\tau$  is overloaded by being used as a constructor of actions and of processes. The one-step transition relation is represented using two predicates: The predicate  $\cdot^1 \xrightarrow{\cdot^2} \cdot^3$  of type  $p \rightarrow a \rightarrow p \rightarrow o$ , where the first argument (indicated with  $\cdot^1$ ) is of type  $p$ , the second argument is of type  $a$ , and the third argument is of type  $p$ ,

encodes transitions involving the free actions while the predicate  $\cdot^1 \xrightarrow{\cdot^2} \cdot^3$  of type  $p \rightarrow (n \rightarrow a) \rightarrow (n \rightarrow p) \rightarrow o$  encodes transitions involving bound values. The precise translation of the  $\pi$ -calculus syntax into simply typed  $\lambda$ -terms is given in the following definition. We assume that names in  $\pi$ -calculus processes are translated to variables (of the same names) in the meta logic.

DEFINITION 6. The following function  $\llbracket \cdot \rrbracket$  translates process expressions to  $\beta\eta$ -long normal terms of type  $p$ .

$$\begin{aligned} \llbracket 0 \rrbracket &= 0 & \llbracket P + Q \rrbracket &= \llbracket P \rrbracket + \llbracket Q \rrbracket & \llbracket P|Q \rrbracket &= \llbracket P \rrbracket | \llbracket Q \rrbracket & \llbracket \tau.P \rrbracket &= \tau \llbracket P \rrbracket \\ \llbracket [x = y]P \rrbracket &= \text{match } x \ y \ \llbracket P \rrbracket & \llbracket \bar{x}y.P \rrbracket &= \text{out } x \ y \ \llbracket P \rrbracket \\ \llbracket x(y).P \rrbracket &= \text{in } x \ \lambda y. \llbracket P \rrbracket & \llbracket (x)P \rrbracket &= \nu \lambda x. \llbracket P \rrbracket \end{aligned}$$

We abbreviate  $\nu \lambda x.P$  as simply  $\nu x.P$ . The one-step transition judgments are translated to atomic formulas as follows (we overload the symbol  $\llbracket \cdot \rrbracket$ ).

$$\begin{aligned} \llbracket P \xrightarrow{xy} Q \rrbracket &= \llbracket P \rrbracket \xrightarrow{\downarrow xy} \llbracket Q \rrbracket & \llbracket P \xrightarrow{x(y)} Q \rrbracket &= \llbracket P \rrbracket \xrightarrow{\downarrow x} \lambda y. \llbracket Q \rrbracket \\ \llbracket P \xrightarrow{\bar{x}y} Q \rrbracket &= \llbracket P \rrbracket \xrightarrow{\uparrow xy} \llbracket Q \rrbracket & \llbracket P \xrightarrow{\bar{x}(y)} Q \rrbracket &= \llbracket P \rrbracket \xrightarrow{\uparrow x} \lambda y. \llbracket Q \rrbracket \\ \llbracket P \xrightarrow{\tau} Q \rrbracket &= \llbracket P \rrbracket \xrightarrow{\tau} \llbracket Q \rrbracket \end{aligned}$$

Notice that we mention encodings of free input actions and free input transition judgments. Since we shall be concerned only with late transition systems, these will not be needed in subsequent specifications. Giving these actions and judgments explicit encodings, however, simplifies the argument for the adequacy of representations of these syntactic judgments: that is, every  $\beta\eta$ -normal term of type  $a$  corresponds to an action in the  $\pi$ -calculus, and similarly, every atomic formula encoding of a one-step transition judgment (in  $\beta\eta$ -normal form) corresponds to a one-step transition judgment in the  $\pi$ -calculus.

Figure 2 contains a definition, called  $\mathbf{D}_\pi$ , that encodes the operational semantics of the late transition system for the finite  $\pi$ -calculus. In this specification, free variables are schema variables that are assumed to be universally scoped over the definition clause in which they appear. These schema variables have primitive types such as  $a$ ,  $n$ , and  $p$  as well as functional types such as  $n \rightarrow a$  and  $n \rightarrow p$ .

Notice that, as a consequence of using  $\lambda$ -tree syntax for this specification, the usual side conditions in the original specifications of the  $\pi$ -calculus [Milner et al. 1992] are no longer present. For example, the side condition that  $X \neq n$  in the open rule is implicit, since  $X$  is outside the scope of  $n$  and, therefore, cannot be instantiated with  $n$  (substitutions into logical expressions cannot capture bound variable names). The adequacy of our encoding is stated in the following lemma and proposition (their proofs can be found in [Tiu 2004]).

LEMMA 7. *The function  $\llbracket \cdot \rrbracket$  is a bijection between  $\alpha$ -equivalence classes of process expressions and  $\beta\eta$ -equivalence classes of terms of type  $p$  whose free variables (if any) are of type  $n$ .*

PROPOSITION 8. *Let  $P$  and  $Q$  be processes and  $\alpha$  an action. Let  $\bar{n}$  be a list of free names containing the free names in  $P$ ,  $Q$ , and  $\alpha$ . The transition  $P \xrightarrow{\alpha} Q$  is derivable in the  $\pi$ -calculus if and only if  $\cdot; \cdot \vdash \nabla \bar{n}. \llbracket P \xrightarrow{\alpha} Q \rrbracket$  is provable in  $FO\lambda^{\Delta\nabla}$  with the definition  $\mathbf{D}_\pi$ .*

$$\begin{array}{l}
\text{TAU:} \quad \tau P \xrightarrow{\tau} P \triangleq \top \\
\text{IN:} \quad \text{in } X M \xrightarrow{\downarrow X} M \triangleq \top \\
\text{OUT:} \quad \text{out } x y P \xrightarrow{\uparrow xy} P \triangleq \top \\
\text{MATCH:} \quad \text{match } x x P \xrightarrow{A} Q \triangleq P \xrightarrow{A} Q \\
\quad \text{match } x x P \xrightarrow{A} Q \triangleq P \xrightarrow{A} Q \\
\text{SUM:} \quad P + Q \xrightarrow{A} R \triangleq P \xrightarrow{A} R \\
\quad P + Q \xrightarrow{A} R \triangleq Q \xrightarrow{A} R \\
\quad P + Q \xrightarrow{A} R \triangleq P \xrightarrow{A} R \\
\quad P + Q \xrightarrow{A} R \triangleq Q \xrightarrow{A} R \\
\text{PAR:} \quad P | Q \xrightarrow{A} P' | Q \triangleq P \xrightarrow{A} P' \\
\quad P | Q \xrightarrow{A} P | Q' \triangleq Q \xrightarrow{A} Q' \\
\quad P | Q \xrightarrow{A} \lambda n (M n | Q) \triangleq P \xrightarrow{A} M \\
\quad P | Q \xrightarrow{A} \lambda n (P | N n) \triangleq Q \xrightarrow{A} N. \\
\text{RES:} \quad \nu n. P n \xrightarrow{A} \nu n. Q n \triangleq \nabla n (P n \xrightarrow{A} Q n) \\
\quad \nu n. P n \xrightarrow{A} \lambda m \nu n. P' n m \triangleq \nabla n (P n \xrightarrow{A} P' n) \\
\text{OPEN:} \quad \nu n. M n \xrightarrow{\uparrow X} M' \triangleq \nabla n (M n \xrightarrow{\uparrow X n} M' n) \\
\text{CLOSE:} \quad P | Q \xrightarrow{\tau} \nu n. (M n | N n) \triangleq \exists X. P \xrightarrow{\downarrow X} M \wedge Q \xrightarrow{\uparrow X} N \\
\quad P | Q \xrightarrow{\tau} \nu n. (M n | N n) \triangleq \exists X. P \xrightarrow{\uparrow X} M \wedge Q \xrightarrow{\downarrow X} N \\
\text{COM:} \quad P | Q \xrightarrow{\tau} M Y | Q' \triangleq \exists X. P \xrightarrow{\downarrow X} M \wedge Q \xrightarrow{\uparrow XY} Q' \\
\quad P | Q \xrightarrow{\tau} P' | N Y \triangleq \exists X. P \xrightarrow{\uparrow XY} P' \wedge Q \xrightarrow{\downarrow X} N
\end{array}$$

Fig. 2. Definition clauses for the late transition system.

If our goal was only to correctly encode one-step transitions for the  $\pi$ -calculus then we would need neither  $\nabla$  nor definitions. In particular, let  $\mathbf{D}_\pi^\forall$  be the result of replacing all  $\nabla$  quantifiers in  $\mathbf{D}_\pi$  with  $\forall$  quantifiers. A slight generalization of Proposition 4 (see [Miller and Tiu 2005; Tiu 2004]) allows us to conclude that  $.; . \vdash \nabla \bar{n}. \llbracket P \xrightarrow{\alpha} Q \rrbracket$  is provable in  $FO\lambda^{\Delta\nabla}$  with the definition  $\mathbf{D}_\pi$  if and only if  $.; . \vdash \forall \bar{n}. \llbracket P \xrightarrow{\alpha} Q \rrbracket$  is provable in  $FO\lambda^{\Delta\nabla}$  with the definition  $\mathbf{D}_\pi^\forall$ . Furthermore, we can also do with the simpler notions of *theory* or *assumptions* and not *definition*. In particular, let  $\mathbf{P}_\pi$  be the set of implications that result from changing all definition clauses in  $\mathbf{D}_\pi^\forall$  into reverse implications (i.e., the head is implied by the body). We can then conclude that  $.; . \vdash \forall \bar{n}. \llbracket P \xrightarrow{\alpha} Q \rrbracket$  is provable in  $FO\lambda^{\Delta\nabla}$  with the definition  $\mathbf{D}_\pi^\forall$  if and only if  $.; \mathbf{P}_\pi \vdash \forall \bar{n}. \llbracket P \xrightarrow{\alpha} Q \rrbracket$  is provable in intuitionistic (and classical) logic. In fact, such a specification of the one-step transitions in the  $\pi$ -calculus as a theory without  $\nabla$  dates back to at least Miller and Palamidessi [1999].

Definitions and  $\nabla$  are needed, however, for proving non-Horn properties (that is, properties requiring a strong notion of negation). The following proposition is a dual of Proposition 8. Its proof can be found in the electronic appendix.

PROPOSITION 9. Let  $P$  and  $Q$  be processes and  $\alpha$  an action. Let  $\bar{n}$  be a list of free names containing the free names in  $P$ ,  $Q$ , and  $\alpha$ . The transition  $P \xrightarrow{\alpha} Q$  is not derivable in the  $\pi$ -calculus if and only if  $\cdot \vdash \neg \nabla \bar{n}. \llbracket P \xrightarrow{\alpha} Q \rrbracket$  is provable in  $FO\lambda^{\Delta\nabla}$  with the definition  $\mathbf{D}_\pi$ .

The following example illustrates how a negation can be proved in  $FO\lambda^{\Delta\nabla}$ . When writing encoded process expressions, we shall use, instead, the syntax of the  $\pi$ -calculus along with the usual abbreviations: for example, when a name  $z$  is used as a prefix, it denotes the prefix  $z(w)$  where  $w$  is vacuous in its scope; when a name  $\bar{z}$  is used as a prefix it denotes the output prefix  $\bar{z}a$  for some fixed name  $a$ . We also abbreviate  $(y)\bar{x}y.P$  as  $\bar{x}(y).P$  and the process term  $0$  is omitted if it appears as the continuation of a prefix. We assume that the operators  $|$  and  $+$  associate to the right, *e.g.*, we write  $P + Q + R$  to denote  $P + (Q + R)$ .

EXAMPLE 10. Consider the process  $(y)([x = y]\bar{x}z)$ , which could be the continuation of some other process which inputs  $x$  on some channel, *e.g.*,  $a(x).(y)[x = y]\bar{x}z$ . Since the bound variable  $y$  is different from any name substituted for  $x$ , that process cannot make a transition and the following formula should be provable.

$$\forall x \forall z \forall Q \forall \alpha. \llbracket ((y)[x = y](\bar{x}z) \xrightarrow{\alpha} Q) \supset \perp \rrbracket$$

Since  $y$  is bound inside the scope of  $x$ , no instantiation for  $x$  can be equal to  $y$ . The formal derivation of the above formula is (ignoring the initial uses of  $\supset \mathcal{R}$  and  $\forall \mathcal{R}$ ):

$$\frac{\frac{\frac{}{\{x, z, Q', \alpha\}; y \triangleright ([x = y](\bar{x}z).0) \xrightarrow{\alpha} Q'y} \vdash \perp}{\{x, z, Q', \alpha\}; \cdot \triangleright \nabla y.([x = y](\bar{x}z).0) \xrightarrow{\alpha} Q'y} \vdash \perp}{\{x, z, Q, \alpha\}; \cdot \triangleright ((y)[x = y](\bar{x}z).0) \xrightarrow{\alpha} Q} \vdash \perp}{\{x, z, Q', \alpha\}; y \triangleright ([x = y](\bar{x}z).0) \xrightarrow{\alpha} Q'y} \vdash \perp} \text{def}\mathcal{L} \quad \nabla \mathcal{L} \quad \text{def}\mathcal{L}$$

The success of the topmost instance of *def* $\mathcal{L}$  depends on the failure of the unification problem  $\lambda y.x = \lambda y.y$ . Notice that the scoping of term-level variables is maintained at the proof-level by the separation of (global) eigenvariables and (locally bound) generic variables. The “newness” of  $y$  is internalized as a  $\lambda$ -abstraction and, hence, it is not subject to instantiation.

The ability to prove a negation is implied by any proof system that can also prove bisimulation for the  $\pi$ -calculus (at least for the finite fragment): for example, the negation above holds because the process  $(y)([x = y]\bar{x}z)$  is bisimilar to  $0$  (see the next section).

## 5. LOGICAL SPECIFICATIONS OF STRONG BISIMILARITY

We consider specifying three notions of bisimilarity tied to the late transition system: the strong early bisimilarity, the strong late bisimilarity and the strong open bisimilarity. As it turns out, the definition clauses corresponding to strong late and strong open bisimilarity coincide. Their essential differences are in the quantification of free names and in the presence (or the absence) of the axiom of excluded middle on the equality of names. The difference between early and late bisimulation is tied to the scope of the quantification of names in the case involving bound

input (see the definitions below). The original definitions of early, late, and open bisimilarity are given in [Milner et al. 1992; Sangiorgi and Walker 2001]. Here we choose to make the side conditions explicit, instead of adopting the bound variable convention in [Sangiorgi and Walker 2001].

Given a relation on processes  $\mathcal{R}$ , we write  $P \mathcal{R} Q$  to denote  $(P, Q) \in \mathcal{R}$ .

DEFINITION 11. A process relation  $\mathcal{R}$  is a *strong late bisimulation* if  $\mathcal{R}$  is symmetric and whenever  $P \mathcal{R} Q$ ,

- (1) if  $P \xrightarrow{\alpha} P'$  and  $\alpha$  is a free action, then there is  $Q'$  such that  $Q \xrightarrow{\alpha} Q'$  and  $P' \mathcal{R} Q'$ ;
- (2) if  $P \xrightarrow{x(z)} P'$  and  $z \notin n(P, Q)$  then there is  $Q'$  such that  $Q \xrightarrow{x(z)} Q'$  and, for every name  $y$ ,  $P'[y/z] \mathcal{R} Q'[y/z]$ ; and
- (3) if  $P \xrightarrow{\bar{x}(z)} P'$  and  $z \notin n(P, Q)$  then there is  $Q'$  such that  $Q \xrightarrow{\bar{x}(z)} Q'$  and  $P' \mathcal{R} Q'$ .

The processes  $P$  and  $Q$  are *strong late bisimilar*, written  $P \sim_l Q$ , if there is a strong late bisimulation  $\mathcal{R}$  such that  $P \mathcal{R} Q$ .

DEFINITION 12. A process relation  $\mathcal{R}$  is a *strong early bisimulation* if  $\mathcal{R}$  is symmetric and whenever  $P \mathcal{R} Q$ ,

- (1) if  $P \xrightarrow{\alpha} P'$  and  $\alpha$  is a free action, then there is  $Q'$  such that  $Q \xrightarrow{\alpha} Q'$  and  $P' \mathcal{R} Q'$ ,
- (2) if  $P \xrightarrow{x(z)} P'$  and  $z \notin n(P, Q)$  then for every name  $y$ , there is  $Q'$  such that  $Q \xrightarrow{x(z)} Q'$  and  $P'[y/z] \mathcal{R} Q'[y/z]$ ,
- (3) if  $P \xrightarrow{\bar{x}(z)} P'$  and  $z \notin n(P, Q)$  then there is  $Q'$  such that  $Q \xrightarrow{\bar{x}(z)} Q'$  and  $P' \mathcal{R} Q'$ .

The processes  $P$  and  $Q$  are *strong early bisimilar*, written  $P \sim_e Q$ , if there is a strong early bisimulation  $\mathcal{R}$  such that  $P \mathcal{R} Q$ .

DEFINITION 13. A *distinction*  $D$  is a finite symmetric and irreflexive relation on names. A substitution  $\theta$  *respects* a distinction  $D$  if  $(x, y) \in D$  implies  $x\theta \neq y\theta$ . We refer to the substitution  $\theta$  as a *D-substitution*. Given a distinction  $D$  and a  $D$ -substitution  $\theta$ , the result of applying  $\theta$  to all variables in  $D$ , written  $D\theta$ , is another distinction. We denote by  $\text{fn}(D)$  the set of names occurring in  $D$ .

Since distinctions are symmetric by definition, when we enumerate a distinction, we often omit the symmetric part of the distinction. For instance, we shall write  $\{(a, b)\}$  to mean the distinction  $\{(a, b), (b, a)\}$ , and we shall also write  $D \cup (S \times T)$ , for some distinction  $D$  and finite sets of names  $S$  and  $T$ , to mean the distinction  $D \cup (S \times T) \cup (T \times S)$ .

Following Sangiorgi [Sangiorgi 1996], we use a set of relations, each indexed by a distinction, to define open bisimulation.

DEFINITION 14. The indexed set  $\mathcal{S} = \{\mathcal{S}_D\}_D$  of process relations is an *indexed open bisimulation* if for every distinction  $D$ , the relation  $\mathcal{S}_D$  is symmetric and for every  $\theta$  that respects  $D$ , if  $P \mathcal{S}_D Q$  then:

$$\begin{aligned}
 \text{ebisim } P \ Q \triangleq & \forall A \forall P' [P \xrightarrow{A} P' \supset \exists Q'. Q \xrightarrow{A} Q' \wedge \text{ebisim } P' \ Q'] \wedge \\
 & \forall A \forall Q' [Q \xrightarrow{A} Q' \supset \exists P'. P \xrightarrow{A} P' \wedge \text{ebisim } Q' \ P'] \wedge \\
 & \forall X \forall P' [P \xrightarrow{\downarrow X} P' \supset \forall w \exists Q'. Q \xrightarrow{\downarrow X} Q' \wedge \text{ebisim } (P'w) \ (Q'w)] \wedge \\
 & \forall X \forall Q' [Q \xrightarrow{\downarrow X} Q' \supset \forall w \exists P'. P \xrightarrow{\downarrow X} P' \wedge \text{ebisim } (Q'w) \ (P'w)] \wedge \\
 & \forall X \forall P' [P \xrightarrow{\uparrow X} P' \supset \exists Q'. Q \xrightarrow{\uparrow X} Q' \wedge \nabla w. \text{ebisim } (P'w) \ (Q'w)] \wedge \\
 & \forall X \forall Q' [Q \xrightarrow{\uparrow X} Q' \supset \exists P'. P \xrightarrow{\uparrow X} P' \wedge \nabla w. \text{ebisim } (Q'w) \ (P'w)] \\
 \\
 \text{lbisim } P \ Q \triangleq & \forall A \forall P' [P \xrightarrow{A} P' \supset \exists Q'. Q \xrightarrow{A} Q' \wedge \text{lbisim } P' \ Q'] \wedge \\
 & \forall A \forall Q' [Q \xrightarrow{A} Q' \supset \exists P'. P \xrightarrow{A} P' \wedge \text{lbisim } Q' \ P'] \wedge \\
 & \forall X \forall P' [P \xrightarrow{\downarrow X} P' \supset \exists Q'. Q \xrightarrow{\downarrow X} Q' \wedge \forall w. \text{lbisim } (P'w) \ (Q'w)] \wedge \\
 & \forall X \forall Q' [Q \xrightarrow{\downarrow X} Q' \supset \exists P'. P \xrightarrow{\downarrow X} P' \wedge \forall w. \text{lbisim } (Q'w) \ (P'w)] \wedge \\
 & \forall X \forall P' [P \xrightarrow{\uparrow X} P' \supset \exists Q'. Q \xrightarrow{\uparrow X} Q' \wedge \nabla w. \text{lbisim } (P'w) \ (Q'w)] \wedge \\
 & \forall X \forall Q' [Q \xrightarrow{\uparrow X} Q' \supset \exists P'. P \xrightarrow{\uparrow X} P' \wedge \nabla w. \text{lbisim } (Q'w) \ (P'w)]
 \end{aligned}$$

 Fig. 3. Specification of strong early, *ebisim*, and late, *lbisim*, bisimulations.

- (1) if  $P\theta \xrightarrow{\alpha} P'$  and  $\alpha$  is a free action, then there is  $Q'$  such that  $Q\theta \xrightarrow{\alpha} Q'$  and  $P' \mathcal{S}_{D\theta} Q'$ ,
- (2) if  $P\theta \xrightarrow{x(z)} P'$  and  $z \notin n(P\theta, Q\theta)$  then there is  $Q'$  such that  $Q\theta \xrightarrow{x(z)} Q'$  and  $P' \mathcal{S}_{D\theta} Q'$ ,
- (3) if  $P\theta \xrightarrow{\bar{x}(z)} P'$  and  $z \notin n(P\theta, Q\theta)$  then there is  $Q'$  such that  $Q\theta \xrightarrow{\bar{x}(z)} Q'$  and  $P' \mathcal{S}_{D'} Q'$  where  $D' = D\theta \cup (\{z\} \times \text{fn}(P\theta, Q\theta, D\theta))$ .

The processes  $P$  and  $Q$  are *strong open  $D$ -bisimilar*, written  $P \sim_o^D Q$ , if there is an indexed open bisimulation  $\mathcal{S}$  such that  $P \mathcal{S}_D Q$ . The processes  $P$  and  $Q$  are *strong open bisimilar* if  $P \sim_o^\emptyset Q$ .

Note that we strengthen a bit the condition 3 in Definition 14 to include the distinction  $(\{z\} \times \text{fn}(D\theta))$ . Strengthening the distinction this way does not change the open bisimilarity, as noted in [Sangiorgi and Walker 2001], but in our encoding of open bisimulation, the distinction  $D$  is part of the specification and the modified definition above helps us account for names better.

Early and late bisimulation can be specified in  $FO\lambda^{\Delta\nabla}$  using the definition clauses in Figure 3. The definition clause for open bisimulation is the same as the one for late bisimulation. The exact relationship between these definitions and the bisimulation relations repeated above will be stated later in this section.

In reasoning about the specifications of early/late bisimulation, we encode free names as  $\nabla$ -quantified variables whereas in the specification of open bisimulation we encode free names as  $\forall$ -quantified variables. For example, the processes  $Pxy = (x|\bar{y})$  and  $Qxy = (x.\bar{y} + \bar{y}.x)$  are late bisimilar. The corresponding encoding in  $FO\lambda^{\Delta\nabla}$  would be  $\nabla x \nabla y. \text{lbisim } (Pxy) \ (Qxy)$ . The free names  $x$  and  $y$  should not be  $\forall$ -quantified for the following, simple reason: in logic we have the

implication  $\forall x \forall y \text{ lbisim } (Pxy) (Qxy) \supset \forall z \text{ lbisim } (Pzz) (Qzz)$ . That is, either  $\forall x \forall y \text{ lbisim } (Pxy) (Qxy)$  is not provable, or it is provable and we have a proof of  $\forall z \text{ lbisim } (Pzz) (Qzz)$ . In either case we lose the adequacy of the encoding.

The definition clauses shown in Figure 3 do not fully capture early and late bisimulations, since there is an implicit assumption in the definition of these bisimulations that name equality is decidable. This basic assumption on the ability to decide the equality among names is one of the differences between open and late bisimulation. Consider, for example, the processes (taken from [Sangiorgi 1996])

$$P = x(u).(\tau.\tau + \tau) \quad \text{and} \quad Q = x(u).(\tau.\tau + \tau + \tau.[u = z]\tau).$$

As shown in [Sangiorgi 1996]  $P$  and  $Q$  are late bisimilar but not open bisimilar: establishing late bisimulation makes use of a case analysis that depends on whether the input name  $u$  is equal to  $z$  or not. Decidability of name equality, in the case of early and late bisimulation, is encoded as an additional axiom of excluded middle on names, i.e., the formula  $\forall x \forall y (x = y \vee x \neq y)$ . Note that since we allow dynamic creation of scoped names (via  $\nabla$ ), we must also state this axiom for arbitrary extensions of local signatures. The following set collects together such generalized excluded middle formulas:

$$\mathcal{E} = \{\nabla n_1 \cdots \nabla n_k \forall x \forall y (x = y \vee x \neq y) \mid k \geq 0\}.$$

We shall write  $\mathcal{X} \subseteq_f \mathcal{E}$  to indicate that  $\mathcal{X}$  is a finite subset of  $\mathcal{E}$ .

The following theorem states the soundness and completeness of the *ebisim* and *lbisim* specifications with respect to the notions of early and late bisimilarity in the  $\pi$ -calculus. By soundness we mean that, given a pair of processes  $P$  and  $Q$ , if the encoding of the late (early) bisimilarity is provable in  $FO\lambda^{\Delta\nabla}$  then the processes  $P$  and  $Q$  are late (early) bisimilar. Completeness is the converse. The soundness and completeness of the open bisimilarity encoding is presented at the end of this section, where we consider the encoding of the notion of distinction in the  $\pi$ -calculus.

**THEOREM 15.** *Let  $P$  and  $Q$  be two processes and let  $\bar{n}$  be the free names in  $P$  and  $Q$ . Then  $P \sim_l Q$  if and only if the sequent  $.; \mathcal{X} \vdash \nabla \bar{n}. \text{lbisim } P \ Q$  is provable for some  $\mathcal{X} \subseteq_f \mathcal{E}$ .*

**THEOREM 16.** *Let  $P$  and  $Q$  be two processes and let  $\bar{n}$  be the free names in  $P$  and  $Q$ . Then  $P \sim_e Q$  if and only if the sequent  $.; \mathcal{X} \vdash \nabla \bar{n}. \text{ebisim } P \ Q$  is provable for some  $\mathcal{X} \subseteq_f \mathcal{E}$ .*

It is well-known that the late bisimulation relation is not a congruence since it is not preserved by the input prefix. Part of the reason why the congruence property fails is that in the late bisimilarity there is no syntactic distinction made between names which can be instantiated and names which cannot be instantiated. Addressing this difference between names is one of the motivations behind the introduction of distinctions and open bisimulation. There is another important difference between open and late bisimulation; in open bisimulation names are instantiated *lazily*, i.e., only when needed. The lazy instantiation of names is intrinsic in  $FO\lambda^{\Delta\nabla}$ ; eigenvariables are instantiated only when applying the *def $\mathcal{L}$* -rule. The syntactic distinction between names that can be instantiated and those that cannot be instantiated are reflected in  $FO\lambda^{\Delta\nabla}$  by the difference between the quantifier  $\forall$

and  $\nabla$ . The alternation of quantifiers in  $FO\lambda^{\Delta\nabla}$  gives rise to a particular kind of distinction, the precise definition of which is given below.

DEFINITION 17. A *quantifier prefix* is a list  $Q_1x_1Q_2x_2 \dots Q_nx_n$  for some  $n \geq 0$ , where  $Q_i$  is either  $\nabla$  or  $\forall$ . If  $Q\bar{x}$  is the above quantifier prefix, then the  $Q\bar{x}$ -*distinction* is the distinction

$$\{(x_i, x_j), (x_j, x_i) \mid i \neq j \text{ and } Q_i = Q_j = \nabla, \text{ or } i < j \text{ and } Q_i = \forall \text{ and } Q_j = \nabla\}.$$

Notice that if  $Q\bar{x}$  consists only of universal quantifiers then the  $Q\bar{x}$ -distinction is empty. Obviously, the alternation of quantifiers does not capture all possible distinction, *e.g.*, the distinction

$$\{(x, y), (y, x), (x, z), (z, x), (u, z), (z, u)\}$$

does not correspond to any quantifier prefix. However, we can encode the full notion of distinction by an explicit encoding of the unequal pairs, as shown later.

It is interesting to see the effect of substitutions on  $D$  when  $D$  corresponds to a prefix  $Q\bar{x}$ . Suppose  $Q\bar{x}$  is the prefix  $Q_1\bar{u}\forall xQ_2\bar{v}\forall yQ_3\bar{w}$ . Since any two  $\forall$ -quantified variables are not made distinct in the definition of  $Q\bar{x}$  prefix, there is a  $\theta$  which respects  $D$  and which can identify  $x$  and  $y$ . Applying  $\theta$  to  $D$  changes  $D$  to some  $D'$  which corresponds to the prefix  $Q_1\bar{u}\forall zQ_2\bar{v}Q_3\bar{w}$ . Interestingly, these two prefixes are related by logical implication:

$$Q_1\bar{u}\forall xQ_2\bar{v}\forall yQ_3\bar{w}.P \supset Q_1\bar{u}\forall zQ_2\bar{v}Q_3\bar{w}.P[z/x, z/y]$$

for any formula  $P$ . This observation suggests the following lemma.

LEMMA 18. *Let  $D$  be a  $Q\bar{x}$ -distinction and let  $\theta$  be a  $D$ -substitution. Then the distinction  $D\theta$  corresponds to some prefix  $Q'\bar{y}$  such that  $Q\bar{x}.P \supset Q'\bar{y}.P\theta$  for any formula  $P$  such that  $\text{fv}(P) \subseteq \{\bar{x}\}$ .*

DEFINITION 19. Let  $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$  be a distinction. The distinction  $D$  is translated as the formula  $\llbracket D \rrbracket = x_1 \neq y_1 \wedge \dots \wedge x_n \neq y_n$ . If  $n = 0$  then  $\llbracket D \rrbracket$  is the logical constant  $\top$  (the empty conjunction).

THEOREM 20. *Let  $P$  and  $Q$  be two processes, let  $D$  be a distinction and let  $Q\bar{x}$  be a quantifier prefix, where  $\bar{x}$  contains the free names in  $P, Q$  and  $D$ . If the formula  $Q\bar{x}.\llbracket D \rrbracket \supset \text{lbisim } P \ Q$  is provable then  $P \sim_o^{D'} Q$ , where  $D'$  is the union of  $D$  and the  $Q\bar{x}$ -distinction.*

THEOREM 21. *If  $P \sim_o^D Q$  then the formula  $\forall \bar{x}.\llbracket D \rrbracket \supset \text{lbisim } P \ Q$  is provable, where  $\bar{x}$  are the free names in  $P, Q$  and  $D$ .*

If a distinction  $D$  corresponds to a quantifier prefix  $Q\bar{x}$ , then it is easy to show that  $Q\bar{x}.\llbracket D \rrbracket$  is derivable in  $FO\lambda^{\Delta\nabla}$ . Therefore, we can state more concisely the adequacy result for the class of  $D$ -open bisimulations in which  $D$  corresponds to a quantifier prefix. The following corollary follows from Theorem 20, Theorem 21 and Proposition 3.

COROLLARY 22. *Let  $D$  be a distinction, let  $P$  and  $Q$  be two processes and let  $Q\bar{x}$  be a quantifier prefix such that  $\bar{x}$  contains the free names of  $D, P$  and  $Q$ , and  $D$  corresponds to the  $Q\bar{x}$ -distinction. Then  $P \sim_o^D Q$  if and only if  $\vdash Q\bar{x}.\text{lbisim } P \ Q$ .*

Note that, by Lemma 18, the property of being a quantifier-prefix distinction is closed under  $D$ -substitution. Note also that in Definition 14(3), if  $D\theta$  is a quantifier-prefix distinction then so is

$$D' = D\theta \cup (\{z\} \times \text{fn}(\mathbf{P}\theta, \mathbf{Q}\theta, D\theta)).$$

That is, if  $D\theta$  corresponds to a quantifier prefix  $Q\vec{x}$ , then  $D'$  corresponds to the quantifier prefix  $Q\vec{x}\nabla z$ . Taken together, these facts imply that one can define an open bisimulation relation which is indexed only by quantifier-prefix distinctions. That is, the family of relations  $\{\mathcal{S}_D\}_D$ , where each  $D$  is a quantifier-prefixed distinction and each  $\mathcal{S}_D$  is defined as

$$\mathcal{S}_D = \{(\mathbf{P}, \mathbf{Q}) \mid \mathbf{P} \sim_o^D \mathbf{Q}\},$$

is an indexed open bisimulation.

Notice the absence of the excluded middle assumption on names in the specification of open bisimulation. Since  $FO\lambda^{\Delta\nabla}$  is intuitionistic, this difference between late and open bisimulation is easily observed. This would not be the case if the specification logic were classical. Since the axiom of excluded middle is present as well in the specification of early bisimulation (Theorem 16), one might naturally wonder if there is a meaningful notion of bisimulation obtained from removing the excluded middle in the specification of early bisimulation and  $\forall$ -quantify the free names. In other words, we would like to see if there is a notion of “open-early” bisimulation. In fact, the resulting bisimulation relation is exactly the same as open “late” bisimulation.

**THEOREM 23.** *Let  $\mathbf{P}$  and  $\mathbf{Q}$  be two processes and let  $\bar{n}$  be the free names in  $\mathbf{P}$  and  $\mathbf{Q}$ . Then  $\forall \bar{n}. \text{Ibisim } \mathbf{P} \ \mathbf{Q}$  is provable if and only if  $\forall \bar{n}. \text{ebisim } \mathbf{P} \ \mathbf{Q}$  is provable.*

We note that while it is possible to prove the impossibility of transitions (Proposition 9) within  $FO\lambda^{\Delta\nabla}$ , it is in general not the case with non-bisimilarity (which is not even recursively enumerable in the infinite setting). If we have evidence that two processes are not bisimilar, say, because one has a trace that the other does not have, then this trace information can be used in the proof a non-bisimulation. Probably a good approach to this is to rely on the modal logics developed later in the paper: if processes are not bisimilar, there is an assertion formula that separates them. We have not planned to develop this particular theme since it seems to us to not be the main thrust of this paper: describing proofs of non-bisimilarity in the finite pi-calculus case is an interesting thing that could be developed on top of the foundation we provide.

To conclude this section, we should explicitly compare the two specifications of early bisimulation in Definition 12 and in Theorem 16, the two specifications of late bisimulation in Definition 11 and in Theorem 15 and the two specifications of open bisimulation in Definition 14 and in Corollary 22. Notice that those specifications that rely on logic are written without the need for any explicit conditions on variable names or any need to mention distinctions explicitly. These various conditions are, of course, present in the detailed description of the proof theory of our logic, but it seems desirable to push the details of variable names, substitutions, free and bound-occurrence, and equalities into logic, where they have elegant and standard solutions.

(a) Propositional connectives and *basic* modality:

$$\begin{aligned}
 (\text{true } : ) \quad P \models \text{true} &\triangleq \top. \\
 (\text{and } : ) \quad P \models A \& B &\triangleq P \models A \wedge P \models B. \\
 (\text{or } : ) \quad P \models A \hat{\vee} B &\triangleq P \models A \vee P \models B. \\
 (\text{match } : ) \quad P \models \langle X \doteq X \rangle A &\triangleq P \models A. \\
 (\text{match } : ) \quad P \models [X \doteq Y] A &\triangleq (X = Y) \supset P \models A. \\
 (\text{free } : ) \quad P \models \langle X \rangle A &\triangleq \exists P' (P \xrightarrow{X} P' \wedge P' \models A). \\
 (\text{free } : ) \quad P \models [X] A &\triangleq \forall P' (P \xrightarrow{X} P' \supset P' \models A). \\
 (\text{out } : ) \quad P \models \langle \uparrow X \rangle A &\triangleq \exists P' (P \xrightarrow{\uparrow X} P' \wedge \nabla y. P'y \models Ay). \\
 (\text{out } : ) \quad P \models [\uparrow X] A &\triangleq \forall P' (P \xrightarrow{\uparrow X} P' \supset \nabla y. P'y \models Ay). \\
 (\text{in } : ) \quad P \models \langle \downarrow X \rangle A &\triangleq \exists P' (P \xrightarrow{\downarrow X} P' \wedge \exists y. P'y \models Ay). \\
 (\text{in } : ) \quad P \models [\downarrow X] A &\triangleq \forall P' (P \xrightarrow{\downarrow X} P' \supset \forall y. P'y \models Ay).
 \end{aligned}$$

$$\begin{aligned}
 (\text{b) Late modality:} \quad P \models \langle \downarrow X \rangle^l A &\triangleq \exists P' (P \xrightarrow{\downarrow X} P' \wedge \forall y. P'y \models Ay). \\
 P \models [\downarrow X]^l A &\triangleq \forall P' (P \xrightarrow{\downarrow X} P' \supset \exists y. P'y \models Ay).
 \end{aligned}$$

$$\begin{aligned}
 (\text{c) Early modality:} \quad P \models \langle \downarrow X \rangle^e A &\triangleq \forall y \exists P' (P \xrightarrow{\downarrow X} P' \wedge P'y \models Ay). \\
 P \models [\downarrow X]^e A &\triangleq \exists y \forall P' (P \xrightarrow{\downarrow X} P' \supset P'y \models Ay).
 \end{aligned}$$

Fig. 4. Modal logics for the  $\pi$ -calculus in  $\lambda$ -tree syntax

## 6. SPECIFICATION OF MODAL LOGICS

We now present the modal logics for the  $\pi$ -calculus that were introduced in [Milner et al. 1993]. In order not to confuse meta-level ( $FO\lambda^{\Delta\nabla}$ ) formulas (or connectives) with the formulas (connectives) of the modal logics under consideration, we shall refer to the latter as object formulas (respectively, object connectives). We shall work only with positive object formulas, i.e., we do not permit negations in those formulas. Note that since there are no atomic formulas in these modal logics (in particular, true or false are not atomic), de Morgan identities can be used to remove all occurrences of negations from such formulas. The syntax of the object formulas is as follows.

$$\begin{aligned}
 \mathbf{A} ::= & \text{true} \mid \text{false} \mid \mathbf{A} \wedge \mathbf{A} \mid \mathbf{A} \vee \mathbf{A} \mid [x = z]\mathbf{A} \mid \langle x = z \rangle \mathbf{A} \\
 & \mid \langle \alpha \rangle \mathbf{A} \mid [\alpha]\mathbf{A} \mid \langle \bar{x}(y) \rangle \mathbf{A} \mid [\bar{x}(y)]\mathbf{A} \mid \langle x(y) \rangle \mathbf{A} \mid [x(y)]\mathbf{A} \\
 & \mid \langle x(y) \rangle^L \mathbf{A} \mid [x(y)]^L \mathbf{A} \mid \langle x(y) \rangle^E \mathbf{A} \mid [x(y)]^E \mathbf{A}
 \end{aligned}$$

The symbol  $\alpha$  denotes a free action, i.e., a free input, a free output, or the silent action. In each of the formulas  $\langle \bar{x}(y) \rangle \mathbf{A}$ ,  $\langle x(y) \rangle \mathbf{A}$ ,  $\langle x(y) \rangle^L \mathbf{A}$  and  $\langle x(y) \rangle^E \mathbf{A}$  (and their dual ‘boxed’-formulas), the occurrence of  $y$  in parentheses is a binding occurrence whose scope is  $\mathbf{A}$ . We use  $\mathbf{A}$ ,  $\mathbf{B}$ ,  $\mathbf{C}$ ,  $\mathbf{D}$  to range over object formulas. Note that we consider only finite conjunctions since the transition system we are considering is finitely branching, and, therefore, an infinite conjunction is not needed (as noted in [Milner et al. 1993]). We consider object formulas equivalent up to renaming of bound variables.

To encode object formulas we introduce the type  $o'$  to denote such formulas and

introduce the following constants for encoding the object connectives: true and false of type  $o'$ ;  $\&$  and  $\hat{\vee}$  of type  $o' \rightarrow o' \rightarrow o'$ ;  $\langle \cdot \dot{=} \cdot \rangle$ ,<sup>3</sup> and  $[\cdot \dot{=} \cdot]$ ,<sup>3</sup> of type  $n \rightarrow n \rightarrow o' \rightarrow o'$ ;  $\langle \cdot \cdot \rangle$ ,<sup>2</sup> and  $[\cdot \cdot]$ ,<sup>2</sup> of type  $a \rightarrow o' \rightarrow o'$ ; and  $\langle \uparrow \cdot \rangle$ ,<sup>2</sup>,  $[\uparrow \cdot]$ ,<sup>2</sup>,  $\langle \downarrow \cdot \rangle$ ,<sup>2</sup>,  $[\downarrow \cdot]$ ,<sup>2</sup>,  $\langle \downarrow \cdot \rangle^l$ ,<sup>2</sup>,  $[\downarrow \cdot]^l$ ,<sup>2</sup>,  $\langle \downarrow \cdot \rangle^e$ ,<sup>2</sup>, and  $[\downarrow \cdot]^e$ ,<sup>2</sup> of type  $n \rightarrow (n \rightarrow o') \rightarrow o'$ . The translation of object formulas to  $\lambda$ -tree syntax is given in the following definition.

DEFINITION 24. The following function  $\llbracket \cdot \rrbracket$  translates object formulas to  $\beta\eta$ -long normal terms of type  $o'$ .

$$\begin{array}{ll} \llbracket \text{true} \rrbracket = \text{true} & \llbracket \text{false} \rrbracket = \text{false} \\ \llbracket \mathbf{A} \wedge \mathbf{B} \rrbracket = \llbracket \mathbf{A} \rrbracket \& \llbracket \mathbf{B} \rrbracket & \llbracket \mathbf{A} \vee \mathbf{B} \rrbracket = \llbracket \mathbf{A} \rrbracket \hat{\vee} \llbracket \mathbf{B} \rrbracket \\ \llbracket \langle x = y \rangle \mathbf{A} \rrbracket = \langle x \dot{=} y \rangle \llbracket \mathbf{A} \rrbracket & \llbracket \langle x = y \rangle \mathbf{A} \rrbracket = \langle x \dot{=} y \rangle \llbracket \mathbf{A} \rrbracket \\ \llbracket \langle \alpha \rangle \mathbf{A} \rrbracket = \langle \alpha \rangle \llbracket \mathbf{A} \rrbracket & \llbracket [\alpha] \mathbf{A} \rrbracket = [\alpha] \llbracket \mathbf{A} \rrbracket \\ \llbracket \langle \bar{x}(y) \rangle \mathbf{A} \rrbracket = \langle \uparrow x \rangle (\lambda y \llbracket \mathbf{A} \rrbracket) & \llbracket [\bar{x}(y)] \mathbf{A} \rrbracket = [\uparrow x] (\lambda y \llbracket \mathbf{A} \rrbracket) \\ \llbracket \langle x(y) \rangle \mathbf{A} \rrbracket = \langle \downarrow x \rangle (\lambda y \llbracket \mathbf{A} \rrbracket) & \llbracket [x(y)] \mathbf{A} \rrbracket = [\downarrow x] (\lambda y \llbracket \mathbf{A} \rrbracket) \\ \llbracket \langle x(y) \rangle^L \mathbf{A} \rrbracket = \langle \downarrow x \rangle^l (\lambda y \llbracket \mathbf{A} \rrbracket) & \llbracket [x(y)]^L \mathbf{A} \rrbracket = [\downarrow x]^l (\lambda y \llbracket \mathbf{A} \rrbracket) \\ \llbracket \langle x(y) \rangle^E \mathbf{A} \rrbracket = \langle \downarrow x \rangle^e (\lambda y \llbracket \mathbf{A} \rrbracket) & \llbracket [x(y)]^E \mathbf{A} \rrbracket = [\downarrow x]^e (\lambda y \llbracket \mathbf{A} \rrbracket) \end{array}$$

In specifying the satisfaction relation  $\models$  between processes and formulas, we restrict to the class of formulas which do not contain occurrences of the free input modality. This is because we consider only the late transition system and the semantics of the free input modality is defined with respect to the early transition system. But we note that adding this input modality and the early transition system does not pose any difficulty. Following Milner et. al., we shall identify an object logic with the set of formulas it allows. We shall refer to the object logic without the free input modalities as  $\mathcal{A}^-$ .

The satisfaction relation  $\models$  is encoded using the same symbol, which is given the type  $p \rightarrow o' \rightarrow o$ . This satisfaction relation is defined by the clauses in Figure 4. This definition, called  $\mathcal{DA}^-$ , corresponds to the modal logic  $\mathcal{A}$  defined in [Milner et al. 1993], minus the clauses for the free input modality. Notice that  $\mathcal{DA}^-$  interprets object-level disjunction and conjunction with, respectively, meta-level disjunction and conjunction. Since the modal logic  $\mathcal{A}^-$  is classical and the meta-logic  $FO\lambda^{\Delta\nabla}$  is intuitionistic, one may wonder whether such an encoding is complete. But since we consider only negation-free object formulas and since there are no atomic formulas, classical and intuitionistic provability coincide for the non-modal fragment of  $\mathcal{A}^-$ . The definition  $\mathcal{DA}^-$  is, however, incomplete for the full logic  $\mathcal{A}^-$ , in the sense that there are true assertions of modal logics that are not provable using this definition alone. Using the ‘box’ modality, one can still encode some limited forms of negation, e.g., inequality of names. For instance, the modal judgment

$$x(y).x(z).0 \models \langle x(y) \rangle \langle x(z) \rangle (\langle x = z \rangle \text{true} \hat{\vee} [x = z] \text{false}),$$

which essentially asserts that any two names are equal or unequal, is valid in  $\mathcal{A}$ , but its encoding in  $FO\lambda^{\Delta\nabla}$  is not provable without additional assumptions. It turns out that, as in the case with the specification of late bisimulation, the only assumption we need to assure completeness is the axiom of excluded middle on the equality of names:  $\forall x \forall y. x = y \vee x \neq y$ . Again, as in the specification of late bisimulation, we must also state this axiom for arbitrary extensions of local signatures. The adequacy of the specification of modal logics is stated in the following theorem.

**THEOREM 25.** *Let  $P$  be a process, let  $A$  be an object formula of the modal logic  $\mathcal{A}^-$ . Then  $P \models A$  if and only if for some list  $\bar{n}$  such that  $\text{fn}(P, A) \subseteq \{\bar{n}\}$  and some  $\mathcal{X} \subseteq_f \mathcal{E}$ , the sequent  $\mathcal{X} \vdash \nabla \bar{n}.(\llbracket P \rrbracket \models \llbracket A \rrbracket)$  is provable in  $FO\lambda^{\Delta\nabla}$  with definition  $\mathcal{DA}^-$ .*

The adequacy result stated in Theorem 25 subsumes the adequacy for the specifications of the sublogics of  $\mathcal{A}^-$ . Note that we quantify free names in the process-formula pair in the above theorem since we do not assume any constants of type  $n$ . Of course, such constants can be introduced without affecting the provability of the satisfaction judgments, but for simplicity, we repeat our treatment of names in the late bisimulation setting here as well.

Notice that the list of names  $\bar{n}$  in Theorem 25 can contain more than just the free names of  $P$  and  $A$ . This is important for the adequacy of the specification, since in the modal logics for the  $\pi$ -calculus, we can specify a modal formula  $A$  and a process  $P$  such that the assertion  $P \models A$  is true only if there exists a new name which is not among the free names of both  $P$  and  $A$ . Consider, for example, the assertion

$$a(x).0 \models [a(x)]^L[x = a]\text{false}$$

and its encoding in  $FO\lambda^{\Delta\nabla}$  as the formula

$$\text{in } a \ (\lambda x.0) \models [\downarrow a]^l(\lambda x.[x \doteq a]\text{false}).$$

If we do not allow extra new names in the quantifier prefix in Theorem 25, then we would have to prove the formula

$$\nabla a.(\text{in } a \ (\lambda x.0) \models [\downarrow a]^l(\lambda x.[x \doteq a]\text{false})).$$

It is easy to see that provability of this formula reduces to provability of

$$\nabla a \exists x.(0 \models [x \doteq a]\text{false}).$$

Since we do not assume any constants of type  $n$ , the only way to prove this would be to instantiate  $x$  with  $a$ , hence,

$$\nabla a.(0 \models [a \doteq a]\text{false}) \quad \text{and} \quad \nabla a.(a = a) \supset 0 \models \text{false}.$$

must be provable. This is, in turn, equivalent to  $\nabla a.0 \models \text{false}$  which should not be provable for the adequacy result to hold. The key step here is the instantiation of  $\exists x$ . For the original formula to be provable,  $x$  has to be instantiated with a name that is distinct from  $a$ . This can be done only if we allow extra names in the quantifier prefix: for example, the following formula is provable.

$$\nabla a \nabla b.(\text{in } a \ (\lambda x.0) \models [\downarrow a]^l(\lambda x.[x \doteq a]\text{false}))$$

Note that in the statement of Theorem 25, the list of names  $\bar{n}$  is existentially quantified. If one is to implement model checking for  $\mathcal{A}^-$  using the specification in Figure 4, the issue of how these names are chosen needs to be addressed. Obviously, the free names of  $\text{fn}(P, A)$  needs to be among  $\bar{n}$ . It remains to calculate how many new names need to be added. An inspection on the definition in Figure 4 shows that such new names may be needed only when bound input modalities are present in the modal formula. More specifically, when instantiating the name quantification

$$\begin{aligned}
!P &\xrightarrow{A} P' | !P \triangleq P \xrightarrow{A} P' \\
!P &\xrightarrow{X} \lambda y (My | !P) \triangleq P \xrightarrow{X} M \\
!P &\xrightarrow{\tau} (P' | M Y) | !P \triangleq \exists X. P \xrightarrow{\uparrow^{XY}} P' \wedge P \xrightarrow{\downarrow^X} M \\
!P &\xrightarrow{\tau} \nu z. (Mz | Nz) | !P \triangleq \exists X. P \xrightarrow{\uparrow^X} M \wedge P \xrightarrow{\downarrow^X} N
\end{aligned}$$

Fig. 5. Definition clauses for the  $\pi$ -calculus with replication

( $\forall$  or  $\exists$ ) in a definition clause for a bound input modality, such as in the definition clause

$$P \models \langle \downarrow^X \rangle^l A \triangleq \exists P' (P \xrightarrow{\downarrow^X} P' \wedge \forall y. P'y \models Ay),$$

we need to consider only cases where  $y$  is instantiated to a free name in  $P \models \langle \downarrow^X \rangle^l A$ , and where  $y$  is instantiated to a new name. For the latter, the particular choice of the new name is unimportant, since the satisfiability relation for  $\mathcal{A}^-$  is closed under substitution with new names (cf. Lemma 3.4. in [Milner et al. 1993]). One can thus calculate the number of new names needed based on the number of bound input modalities in  $A$ .

In [Milner et al. 1993], late bisimulation was characterized by the sublogic  $\mathcal{LM}$  of  $\mathcal{A}^-$  that arises from restricting the formulas to contain only the propositional connectives and the following modalities:  $\langle \tau \rangle$ ,  $\langle \bar{x}y \rangle$ ,  $\langle \bar{x}(y) \rangle$ ,  $[x = y]$ ,  $\langle x(y) \rangle^L$ , and their duals. We shall now show a similar characterization for open bisimulation.

The following theorem states that by dropping the excluded middle and changing the quantification of free names from  $\nabla$  to  $\forall$ , we get exactly a characterization of open bisimulation by the encoding of the sublogic  $\mathcal{LM}$ .

**THEOREM 26.** *Let  $P$  and  $Q$  be two processes. Then  $P \sim_o^\emptyset Q$  if and only if for every  $\mathcal{LM}$ -formula  $A$ , it holds that  $\vdash \forall \bar{n} (\llbracket P \rrbracket \models \llbracket A \rrbracket)$  if and only if  $\vdash \forall \bar{n} (\llbracket Q \rrbracket \models \llbracket A \rrbracket)$ , where  $\bar{n}$  is the list of free names in  $P$ ,  $Q$  and  $A$ .*

## 7. ALLOWING REPLICATION IN PROCESS EXPRESSIONS

We now consider an extension to the finite  $\pi$ -calculus which will allow us to represent non-terminating processes. There are at least two ways to encode non-terminating processes in the  $\pi$ -calculus; *e.g.*, via recursive definitions or replications [Sangiorgi and Walker 2001]. We consider here the latter approach since it leads to a simpler presentation of the operational semantics. To the syntax of the finite  $\pi$ -calculus we add the process expression  $!P$ . The process  $!P$  can be understood as the infinite parallel composition of  $P$ , *i.e.*,  $P|P|\dots|P|\dots$ . Thus it is possible to have a process that retains a copy of itself after making a transition; *e.g.*,  $!P \xrightarrow{\alpha} P' | !P$ . The operational semantics for one-step transitions of the  $\pi$ -calculus with replication is given as the definition clauses Figure 5, adapted to the  $\lambda$ -tree syntax from the original presentation in [Sangiorgi and Walker 2001]. We use the same symbol to encode replication in  $\lambda$ -tree syntax, *i.e.*,  $! : p \rightarrow p$ .

In order to reason about bisimulation of processes involving  $!$ , we need to move to a stronger logic which incorporates both induction and co-induction proof rules. We consider the logic *Linc* [Tiu 2004], which is an extension of  $FO\lambda^{\Delta\nabla}$  with induction

and co-induction proof rules. We first need to extend the notion of definitions to include inductive and co-inductive definitions.

DEFINITION 27. An inductive definition clause is written

$$\forall \bar{x}. p\bar{x} \stackrel{\mu}{=} B p \bar{x}$$

where  $B$  is a closed term. The symbol  $\stackrel{\mu}{=}$  is used to indicate that the definition is inductive. Similarly, a co-inductive definition clause is written

$$\forall \bar{x}. p\bar{x} \stackrel{\nu}{=} B p \bar{x}.$$

The notion of definition given in Definition 1 shall be referred to as *basic definition*. An *extended definition* is a collection of basic, inductive, or co-inductive definition clauses.

A definition clause can be seen as a fixed point equation: in fact, Baldle & Miller [2007] provide an alternative approach to inductive and co-inductive definitions similar to what is available in the  $\mu$ -calculus. When definitions are seen as fixed points, provability of  $p\bar{t}$ , depending on whether  $p$  is basic, inductive or co-inductive, means that  $\bar{t}$  is, respectively, in a fixed point, the least fixed point, and the greatest fixed point of the underlying fixed point equation defining  $p$ .

Notice that the head of the (co-)inductive definition clauses contains a predicate with arguments that are only variables and not more general terms: this restriction simplifies the presentation of the induction and co-induction inference rules. Arguments that are more general terms can be encoded as explicit equalities in the body of the clause. We also adopt a higher-order notation in describing the body of clauses, *i.e.*, we use  $B p \bar{x}$  to mean that  $B$  is a top-level abstraction that has no free occurrences of the predicate symbol  $p$  and the variables  $\bar{x}$ . This notation simplifies the presentation of the (co-)induction rules: in particular, it simplifies the presentation of predicate substitutions.

There must be some stratification on the extended definition so as not to introduce inconsistency into the logic. For the details of such stratification we refer the interested readers to [Tiu 2004]. For our current purpose, it should be sufficient to understand that mutual recursive (co-)inductive definitions are not allowed, and dependencies through negation are forbidden as it already is in basic definitions.

Let  $p\bar{x} \stackrel{\mu}{=} B p \bar{x}$  be an inductive definition. Its left and right introduction rules are

$$\frac{\bar{x}; B S \bar{x} \vdash S \bar{x} \quad \Sigma; \bar{z} \triangleright S \bar{t}, \Gamma \vdash C}{\Sigma; \bar{z} \triangleright p \bar{t}, \Gamma \vdash C} \mu\mathcal{L} \qquad \frac{\Sigma; \Gamma \vdash \bar{z} \triangleright B p \bar{t}}{\Sigma; \Gamma \vdash \bar{z} \triangleright p \bar{t}} \mu\mathcal{R}$$

where  $S$  is the *induction invariant*, and it is a closed term of the same type as  $p$ . The introduction rules for co-inductively defined predicates are dual to the inductive ones. In this case, we suppose that  $p$  is defined by the co-inductive clause  $p\bar{x} \stackrel{\nu}{=} B p \bar{x}$ .

$$\frac{\Sigma; \bar{z} \triangleright B p \bar{t}, \Gamma \vdash C}{\Sigma; \bar{z} \triangleright p \bar{t}, \Gamma \vdash C} \nu\mathcal{L} \qquad \frac{\Sigma; \Gamma \vdash \bar{z} \triangleright S \bar{t} \quad \bar{x}; S \bar{x} \vdash B S \bar{x}}{\Sigma; \Gamma \vdash \bar{z} \triangleright p \bar{t}} \nu\mathcal{R}$$

Here  $S$  is a closed term denoting the co-induction invariant or *simulation*. Induction rules cannot be applied to co-inductive predicates and vice versa. The  $\text{def}\mathcal{R}$  and

$def\mathcal{L}$  rules, strictly speaking, are applicable only to basic definitions. But as it is shown in [Tiu 2004], these rules are derivable for (co-)inductive definitions: that is, for these definitions,  $def\mathcal{R}$  can be shown to be a special case of  $\nu\mathcal{R}$  and  $def\mathcal{L}$  a special case of  $\mu\mathcal{L}$ .

The definitions in  $FO\lambda^{\Delta\nabla}$  we have seen so far can be carried over to Linc with some minor bureaucratic changes: *e.g.*, in the case of bisimulations, we now need to indicate explicitly that it is a co-inductive definition. For instance, the definition of  $l\text{bisim}$  should now be indicated as a co-inductive definition by changing the symbol  $\stackrel{\Delta}{\equiv}$  with  $\stackrel{\nabla}{\equiv}$ . We shall now present an example of proving bisimulation using explicit induction and co-induction rules. We shall not go into details of the technical theorems of the adequacy results: these can be found in [Tiu 2004].

EXAMPLE 28. Let  $P =!(z)(\bar{z}a \mid z(y).\bar{x}y)$  and  $Q =!\tau.\bar{x}a$ . The only action  $P$  can make is the silent action  $\tau$  since the channel  $z$  is restricted internally within the process. It is easy to see that  $P \xrightarrow{\tau} (z)(0 \mid \bar{x}a) \mid P$ . That is, the continuation of  $P$  is capable of outputting a free name  $a$  or making a silent transition. Obviously  $Q$  can make the same  $\tau$  action and results in a bisimilar continuation. Let us try to prove  $l\text{bisim } P \ Q$ . The simple proof strategy of unfolding the  $l\text{bisim}$  clause via  $def\mathcal{R}$  will not work here since after the first  $def\mathcal{R}$  on  $l\text{bisim}$  (but before the second  $def\mathcal{R}$  on  $l\text{bisim}$ ) we arrive at the sequent  $l\text{bisim } ((z)(0 \mid \bar{x}a) \mid P) (\bar{x}a \mid Q)$ . Since  $P$  and  $Q$  still occur in the continuation pair, it is obvious that this strategy is non terminating. We need to use the co-induction proof rules instead.

An informal proof starts by finding a bisimulation (a set of pairs of processes)  $\mathcal{S}$  such that  $(P, Q) \in \mathcal{S}$ . Let

$$\mathcal{S}' = \{(R_1 \mid \dots \mid R_n \mid P, T_1 \mid \dots \mid T_n \mid Q) \mid n \geq 0, R_i \text{ is } (z)(0 \mid \bar{x}a) \text{ or } (z)(0 \mid 0) \text{ and } T_i \text{ is either } \bar{x}a \text{ or } 0\}.$$

Define  $\mathcal{S}$  to be the symmetric closure of  $\mathcal{S}'$ . It can be verified that  $\mathcal{S}$  is a bisimulation set by showing the set is closed with respect to one-step transitions. To prove this formally in Linc we need to represent the set  $\mathcal{S}$ . We code the set  $\mathcal{S}$  as the following inductive definition (we allow ourselves to put general terms in the head of this definition and to have more than one clause: it is straightforward to translate this definition to the restricted one give above).

$$\begin{aligned} inv \ P \ Q &\stackrel{\Delta}{\equiv} \top. \quad inv \ Q \ P \stackrel{\Delta}{\equiv} \top. \\ inv \ ((z)(0 \mid 0) \mid M) \ (0 \mid N) &\stackrel{\Delta}{\equiv} inv \ M \ N. \\ inv \ (0 \mid N) \ ((z)(0 \mid 0) \mid M) &\stackrel{\Delta}{\equiv} inv \ N \ M. \\ inv \ ((z)(0 \mid \bar{x}a) \mid M) \ (\bar{x}a \mid N) &\stackrel{\Delta}{\equiv} inv \ M \ N. \\ inv \ (\bar{x}a \mid N) \ ((z)(0 \mid \bar{x}a) \mid M) &\stackrel{\Delta}{\equiv} inv \ N \ M. \end{aligned}$$

Note that for simplicity of presentation, we assume that we have two constants of type  $n$ , namely,  $x$  and  $a$ , in the logic (but we note that this assumption is not necessary). The set of pairs encoded by  $inv$  can be shown to be symmetric, i.e., the formula  $\forall R \forall T. inv \ R \ T \supset inv \ T \ R$  is provable inductively (using the same formula as the induction invariant).

To now prove the sequent  $\vdash l\text{bisim } P \ Q$ , we can use the  $\nu\mathcal{R}$  rule with the predicate  $inv$  as the invariant. The premises of the  $\nu\mathcal{R}$  rule are the two sequents  $\vdash inv \ P \ Q$

and  $R, T; \text{inv } R T \vdash B R T$ , where  $B R T$  is the following large conjunction

$$\begin{aligned}
 & \forall A \forall R' [(R \xrightarrow{A} R') \supset \exists T'. (T \xrightarrow{A} T') \wedge \text{inv } R' T'] \wedge \\
 & \forall A \forall T' [(T \xrightarrow{A} T') \supset \exists R'. (R \xrightarrow{A} R') \wedge \text{inv } T' R'] \wedge \\
 & \forall X \forall R' [(R \xrightarrow{\downarrow X} R') \supset \exists T'. (T \xrightarrow{\downarrow X} T') \wedge \forall w. \text{inv } (R' w) (T' w)] \wedge \\
 & \forall X \forall T' [(T \xrightarrow{\downarrow X} T') \supset \exists R'. (R \xrightarrow{\downarrow X} R') \wedge \forall w. \text{inv } (T' w) (R' w)] \wedge \\
 & \forall X \forall R' [(R \xrightarrow{\uparrow X} R') \supset \exists T'. (T \xrightarrow{\uparrow X} T') \wedge \nabla w. \text{inv } (R' w) (T' w)] \wedge \\
 & \forall X \forall T' [(T \xrightarrow{\uparrow X} T') \supset \exists R'. (R \xrightarrow{\uparrow X} R') \wedge \nabla w. \text{inv } (T' w) (R' w)].
 \end{aligned}$$

The sequent reads, intuitively, that the set defined by  $\text{inv}$  is closed under one-step transitions. This is proved by induction on  $\text{inv}$ . Formally, this is done by applying  $\mu\mathcal{L}$  to  $\text{inv } R T$ , using the invariant

$$\lambda R \lambda T. \text{inv } R T \supset B R T.$$

The sequents corresponding to the base cases of the induction are

$$\text{inv } P Q \vdash B P Q \quad \text{and} \quad \text{inv } Q P \vdash B Q P$$

and the inductive cases are given by

$$\begin{aligned}
 & \text{inv } R T \supset B R T \vdash \text{inv } ((z)(0|0) | R) (0 | T) \supset B((z)(0|0) | R)(0 | T), \\
 & \text{inv } R T \supset B R T \vdash \text{inv } ((z)(0 | \bar{x}a) | R) (\bar{x}a | T) \supset B((z)(0 | \bar{x}a) | R)(\bar{x}a | T)
 \end{aligned}$$

and their symmetric variants. The full proof involves a number of cases of which we show one here: the other cases can be proved similarly.

We consider a case for free output, where we have the sequent (after applying some right-introduction rules)

$$\left\{ \begin{array}{l} \text{inv } R T \supset B R T \\ \text{inv } ((z)(0 | \bar{x}a) | R) (\bar{x}a | T) \\ ((z)(0 | \bar{x}a) | R) \xrightarrow{A} R' \end{array} \right\} \vdash \exists T'. (\bar{x}a | T) \xrightarrow{A} T' \wedge \text{inv } R' T' \quad (1)$$

to prove. Its symmetric case can be proved analogously. The sequent (1) can be simplified by applying  $\text{def}\mathcal{L}$  to the  $\text{inv}$  predicate, followed by an instance of  $\supset \mathcal{L}$ . The resulting sequent is

$$\left\{ \begin{array}{l} B R T, \text{inv } R T \\ ((z)(0 | \bar{x}a) | R) \xrightarrow{A} R' \end{array} \right\} \vdash \exists T'. (\bar{x}a | T) \xrightarrow{A} T' \wedge \text{inv } R' T' \quad (2)$$

There are three ways in which the one-step transition in the left-hand side of the sequent (1) can be inferred (via  $\text{def}\mathcal{L}$ ), *i.e.*, either  $A$  is  $\bar{x}a$  and  $R'$  is  $((z)(0|0) | R)$ , or  $R \xrightarrow{A} R''$  and  $R'$  is  $(z)(0 | \bar{x}a) | R''$ , or  $A$  is  $\tau$  and  $R \xrightarrow{\downarrow X} M$ ,  $R'$  is  $((z)(0|0) | M a)$  for some  $X$  and  $M$ . These three cases correspond to the following sequents.

$$\begin{aligned}
 & B R T, \text{inv } R T \vdash \exists T'. (\bar{x}a | T) \xrightarrow{\bar{x}a} T' \wedge \text{inv } ((z)(0|0) | R) T' \\
 & B R T, \text{inv } R T, R \xrightarrow{A} R'' \vdash \exists T'. (\bar{x}a | T) \xrightarrow{A} T' \wedge \text{inv } (z)(0 | \bar{x}a) | R'' T' \\
 & B R T, \text{inv } R T, R \xrightarrow{\downarrow X} M \vdash \exists T'. (\bar{x}a | T) \xrightarrow{\tau} T' \wedge \text{inv } ((z)(0|0) | M a) T'
 \end{aligned}$$

$$\frac{\frac{\overline{\dots \vdash \top} \quad \top \mathcal{R}}{\dots \vdash (\bar{x}a | T) \xrightarrow{\bar{x}a} (0 | T)} \quad \text{def}\mathcal{R} \quad \frac{\overline{\dots, \text{inv } R T \vdash \text{inv } R T} \quad \text{init}}{\dots, \text{inv } R T \vdash \text{inv } ((z)(0 | 0) | R) (0 | T)} \quad \text{def}\mathcal{R}}{\frac{B R T, \text{inv } R T \vdash (\bar{x}a | T) \xrightarrow{\bar{x}a} (0 | T) \wedge \text{inv } ((z)(0 | 0) | R) (0 | T)}{B R T, \text{inv } R T \vdash \exists T'. (\bar{x}a | T) \xrightarrow{\bar{x}a} T' \wedge \text{inv } ((z)(0 | 0) | R) T'} \exists \mathcal{R}} \wedge \mathcal{R}$$

Fig. 6. A derivation in Linc

$$\frac{\frac{\frac{\overline{R \xrightarrow{A} R'' \vdash R \xrightarrow{A} R''} \quad \text{init}}{R \xrightarrow{A} R'' \supset \exists V.T \xrightarrow{A'} V \wedge \text{inv } R'' V, R \xrightarrow{A} R'' \vdash \dots} \supset \mathcal{L}}{\forall U \forall A' R \xrightarrow{A} U \supset \exists V.T \xrightarrow{A'} V \wedge \text{inv } U V, R \xrightarrow{A} R'' \vdash \dots} \forall \mathcal{L}; \forall \mathcal{L}}{\frac{\Pi}{B R T, R \xrightarrow{A} R'' \vdash \exists T'. (\bar{x}a | T) \xrightarrow{A} T' \wedge \text{inv } (z)(0 | \bar{x}a) | R''} T'} \wedge \mathcal{L}$$

where  $\Pi$  is

$$\frac{\frac{\frac{\overline{T \xrightarrow{A} V \vdash T \xrightarrow{A} V} \quad \text{init}}{T \xrightarrow{A} V \vdash (\bar{x}a | T) \xrightarrow{A} (\bar{x}a | V)} \quad \text{def}\mathcal{R} \quad \frac{\overline{\text{inv } R'' V \vdash \text{inv } R'' V} \quad \text{init}}{\text{inv } R'' V \vdash \text{inv } ((z)(0 | \bar{x}a) | R'') (\bar{x}a | V)} \quad \text{def}\mathcal{R}}{\frac{T \xrightarrow{A} V, \text{inv } R'' V \vdash (\bar{x}a | T) \xrightarrow{A} (\bar{x}a | V) \wedge \text{inv } (z)(0 | \bar{x}a) | R''} {T \xrightarrow{A} V, \text{inv } R'' V \vdash \exists T'. (\bar{x}a | T) \xrightarrow{A} T' \wedge \text{inv } (z)(0 | \bar{x}a) | R''} \exists \mathcal{R}} \exists \mathcal{L}; \wedge \mathcal{L}} \exists V.T \xrightarrow{A} V \wedge \text{inv } R'' V \vdash \exists T'. (\bar{x}a | T) \xrightarrow{A} T' \wedge \text{inv } (z)(0 | \bar{x}a) | R''} T'$$

Fig. 7. A derivation in Linc given in two parts

The proof of the first sequent is given in Figure 6 and of the second sequent is given in Figure 7. The proof for the third sequent is not given but it is easy to see that it has a similar structure to the proof of the second one.

## 8. AUTOMATION OF PROOF SEARCH

The above specifications for one-step transitions, for late, early, and open bisimulation, and for modal logics are not only declarative and natural, they can also, in many cases, be turned into effective and *symbolic* implementations by using techniques from the proof search literature. In this section we outline high-level aspects of the proof theory of  $FO\lambda^{\Delta\nabla}$  that can be directly exploited to provide implementations of significant parts of this logic: we also describe how such general aspects can be applied to some of our  $\pi$ -calculus examples.

### 8.1 Focused proof search

Since the cut-elimination theorem holds for  $FO\lambda^{\Delta\nabla}$ , the search for a proof can be restricted to *cut-free proofs*. It is possible to significantly constrain cut-free proofs to *focused proofs* while still preserving completeness. The search for focused proofs has a simple structure that is organized into two phases. The *asynchronous* phase applies only invertible inference rules in any order and until no additional invertible

rules can be applied. The *synchronous* phase involves the selection of (possibly) non-invertible inference rule and the hereditary (focused) application of such inference rules until invertible rules are possible again. Andreoli [Andreoli 1992] provided such a focused proof system for linear logic and proved its completeness. Subsequently, many focusing systems for intuitionistic and classical logic have been developed, *cf.* [Liang and Miller 2007] for a description of several of them. Baelde and Miller [2007] present a focusing proof system for the multiplicative and additive linear logic (MALL) extended with fixed points and show that that proof system provides a focusing proof system for a large subset of  $FO\lambda^\Delta$ . Focused proof systems are generally the basis for the automation of logic programming languages and they generalize the notion of *uniform proofs* [Miller et al. 1991].

## 8.2 Unification

Unification can be used in the implementation of  $FO\lambda^{\Delta\nabla}$  proof search in two different ways. One way involves the implementation of the *def $\mathcal{L}$*  inference rule and the other way involves the determination of appropriate terms for instantiating the  $\exists$  quantifier in the  $\exists\mathcal{R}$  inference rule and the  $\forall$  quantifier in the  $\forall\mathcal{L}$  inference rule. In the specifications presented here, unification only requires the decidable and determinate subset of higher-order unification called *higher-order pattern* (or  $L_\lambda$ ) unification [Miller 1991]. This style of unification, which can be described as first-order unification extended to allow for bound variables and their mobility within terms, formulas, and proofs, is known to have efficient and practical unification algorithms that compute most general unifiers whenever unifiers exist [Nipkow 1993; Nadathur and Linnell 2005]. The Teyjus implementation [Nadathur and Mitchell 1999; Nadathur 2005] of  $\lambda$ Prolog provides an effective implementation of such unification, as does Isabelle [Paulson 1990] and Twelf [Pfenning and Schürmann 1999].

## 8.3 Proof search for one-step transitions.

Computing one-step transitions can be done entirely using a conventional, higher-order logic programming language, such as  $\lambda$ Prolog: since the definition  $\mathbf{D}_\pi$  for one-step transitions is Horn, we can use Proposition 4 to show that for the purposes of computing one-step transitions, all occurrences of  $\nabla$  in  $\mathbf{D}_\pi$  can be changed to  $\forall$ . The resulting definition is then a  $\lambda$ Prolog logic program for which Teyjus provides an effective implementation. In particular, after loading that definition, we would simply ask the query  $P \xrightarrow{A} P'$ , where  $P$  is the encoding of a particular  $\pi$ -calculus expression and  $A$  and  $P'$  are free variables. Standard logic programming interpreters would then systematically bind these two variables to the actions and continuations that  $P$  can make. Similarly, if the query was  $P \xrightarrow{A} P'$ , logic programming search would systematically return all bound actions (here,  $A$  has type  $n \rightarrow a$ ) and corresponding bound continuations (here,  $P'$  has type  $n \rightarrow p$ ).

## 8.4 Proof search for open bisimulation.

Theorem proving establishing a bisimulation goal is not done via a conventional logic programming system like  $\lambda$ Prolog since such systems do not implement the  $\nabla$ -quantifier and the case analysis and unification of eigenvariables that is required for the *def $\mathcal{L}$*  inference rule. None-the-less, the implementation of proof search for

open bisimulation is easy to specify using the following key steps. (Sequents missing from this outline are trivial to address.) In the following, we use the quantifier prefix  $\mathcal{Q}$  to denote either  $\forall x$  or  $\nabla x$  or the empty quantifier prefix.

- (1) When searching for a proof of  $\Sigma; \vdash \sigma \triangleright \mathcal{Q}.l\text{bisim } P \ Q$  apply right-introduction rules: *i.e.*, simply introduce the quantifier  $\mathcal{Q}$  (if it is non-empty) and then open the definition of *lbisim*.
- (2) If the sequent has a formula on its left-hand side, then that formula is  $\sigma \triangleright P \xrightarrow{A} P'$ , where  $P$  denotes a particular term where all its non-ground subterms are of type  $n$ , and  $A$  and  $P'$  are terms, possibly containing eigenvariables. In this case, select the *defL* inference rule: the premises of this inference rule will then be either (i) the empty-set of premises (which represents the only way that proof search terminates), or (ii) a set of premises that are all again of the form of one-step judgments, or (iii) the premise contains  $\top$  instead of an atom on the left, in which case, we must consider the remaining case that follows (after using the weakening *wL* inference rule).
- (3) If the sequent has the form  $\Sigma; \vdash \sigma \triangleright \exists Q'[Q \xrightarrow{A} Q' \wedge B(P', Q')]$ , where  $B(P', Q')$  involves a recursive call to *lbisim* and where  $P'$  is a closed term, then we must instantiate the existential quantifier with an appropriate substitution. Standard logic programming techniques can be used to find a substitution for  $Q'$  such that  $Q \xrightarrow{A} Q'$  is provable (during this search, eigenvariables and locally scoped variables are treated as constants and  $P$  and  $A$  denote particular closed terms). There might be several ways to prove such a formula and, as a result, there might be several different substitutions for  $Q'$ . If one chooses the term  $T$  to instantiate  $Q'$ , then one proceeds to prove the sequent  $\Sigma; \vdash \sigma \triangleright \mathcal{Q}.l\text{bisim } P' \ T$ . If the sequent has instead the form  $\Sigma; \vdash \sigma \triangleright \exists Q'[Q \xrightarrow{A} Q' \wedge B(P', Q')]$ , then one proceeds in an analogous manner.

Proof search for the first two cases is invertible (no backtracking is needed for those cases). On the other hand, the third case is not invertible and backtracking on possibly all choices of substitution term  $T$  might be necessary to ensure completeness.

### 8.5 The Bedwyr model checker

The various implementation techniques mentioned above—unification of  $\lambda$ -terms, backtracking focused proof search, unfolding definitions—have all been implemented within the Bedwyr model checking system [Baelde et al. 2007], which implements proof search for a simple fragment [Tiu et al. 2005] of  $FO\lambda^{\Delta\nabla}$ . The definitions of one-step transitions and of bisimulation are in this fragment and the Bedwyr system is a complete implementation of open bisimulation for the finite  $\pi$ -calculus: in particular, it provides a decision procedure for open-bisimulation. Bedwyr also implements limited forms of the modal logic described in Section 6. It is also possible to use Bedwyr to explore why two  $\pi$ -calculus processes might not be bisimilar: for example, it is easy to define traces for such processes and then to search for a trace that holds of one process but not of the other.

Since Bedwyr is limited to intuitionistic reasoning, it does not fully implement late bisimulation. We now speculate briefly on how one might extend a system like

Bedwyr to treat late bisimulation.

### 8.6 Proof search for late bisimulation.

The main difference between doing proof search for open bisimulation and late bisimulation is that in the latter we need to select and instantiate formulas from the set  $\mathcal{E}$  and explore the cases generated by the resulting  $\forall\mathcal{L}$  rule. For example, consider a sequent of the form  $\Sigma, x; \mathcal{E}, \Gamma_x \vdash C_x$ , where  $\Gamma_x \cup \{C_x\}$  is a set of formulas which may have  $x$  free. One way to proceed with the search for a proof would be to instantiate  $\forall z(x = z \vee x \neq z)$  twice with the constants  $a$  and  $b$ . We would then need to consider proofs of the sequent  $\Sigma, x; x = a \vee x \neq a, x = b \vee x \neq b, \Gamma_x \vdash C_x$ . Using the  $\forall\mathcal{L}$  rule twice, we are left with four sequents to prove:

- (1)  $\Sigma, x; x = a, x = b, \Gamma_x \vdash C_x$  which is proved trivially since the equalities are contradictory;
- (2)  $\Sigma, x; x = a, x \neq b, \Gamma_x \vdash C_x$ , which is equivalent to  $\Sigma; \Gamma_a \vdash C_a$ ;
- (3)  $\Sigma, x; x \neq a, x = b, \Gamma_x \vdash C_x$ , which is equivalent to  $\Sigma; \Gamma_b \vdash C_b$ ; and
- (4)  $\Sigma, x; x \neq a, x \neq b, \Gamma_x \vdash C_x$ .

In this way, the excluded middle can be used with a set of  $n$  items to produce  $n + 1$  sequents: one for each member of the set and one extra sequent to handle all other cases (if there are any).

The main issue for implementing proof search with this specification of late bisimulation is to determine what instances of the excluded middle are needed: answering this question would then reduce proof search to one similar to open bisimulation. There seems to be two extreme approaches to take. At one extreme, we can take instances for all possible names that are present in our process expressions: determining such instances is simple but might lead to many more cases to consider than is necessary. The other extreme would be more lazy: an instance of the excluded middle is suggested only when there seems to be a need to consider that instance. The failure of a *def $\mathcal{R}$*  rule because of a mismatch between an eigenvariable and a constant would, for example, suggest that excluded middle should be invoked for that eigenvariable and that constant. The exact details of such schemes and their completeness are left for future work.

## 9. RELATED AND FUTURE WORK

There are many papers on topics related to the encoding of the operational semantics of the  $\pi$ -calculus into formal systems. An encoding of one-step transitions for the  $\pi$ -calculus using Coq was presented in [Despeyroux 2000] but the problem of computing bisimulation was not considered. Honsell, Miculan, and Scagnetto [Honsell et al. 2001] give a more involved encoding of the  $\pi$ -calculus in Coq and assume that there are an infinite number of global names. They then build formal mechanisms to support notions such as “freshness” within a scope, substitution of names, occurrences of names in expressions, etc. Gabbay [Gabbay 2003] does something similar but uses the set theory developed in [Gabbay and Pitts 2001] to help develop his formal mechanisms. This formalism is later given a first-order axiomatization by Pitts [Pitts 2003], resulting in an extension of first-order logic called *nominal logic*. Aspects of nominal reasoning have been incorporated into the

proof assistant Isabelle [Urban and Tasson 2005] and there has been some recent work in formalizing the meta theory of the  $\pi$ -calculus in this framework [Bengtson and Parrow 2007]. Hirschhoff [Hirschhoff 1997] also used Coq but employed deBruijn numbers [de Bruijn 1972] instead of explicit names. In the papers that address bisimulation, formalizing names and their scopes, occurrences, freshness, and substitution is considerable work. In our approach, much of this same work is required, of course, but it is available in rather old technology, particularly, via Church's Simple Theory of Types (where bindings in terms and formulas were put on a firm foundation via  $\lambda$ -terms), Gentzen's sequent calculus, Huet's unification procedure for  $\lambda$ -terms [Huet 1975], etc. More modern work on proof search in higher-order logics is also available to make our task easier and more declarative.

The encoding of transitions for the  $\pi$ -calculus into logics and type systems have been studied in a number of previous works [Honsell et al. 1998; Despeyroux 2000; Honsell et al. 2001; Röckl et al. 2001; Bengtson and Parrow 2007]. Our encoding, presented as a definition in Figure 2, has appeared in [Miller and Palamidessi 1999; Miller and Tiu 2003]. The material on proof automation in Section 8 clearly seems related to *symbolic bisimulation* (for example, see [Hennessy and Lin 1995; Boreale and Nicola 1996]) and on using unification and logic programming techniques to compute symbolic bisimulations (for example, see [Basu et al. 2001; Boreale 2001]). Since the technologies used to describe these other approaches are rather different than what is described here, a detailed comparison is left for future work.

It is, of course, interesting to consider the general  $\pi$ -calculus where infinite behaviors are allowed (by including ! or recursive definitions). In such cases, one might be able to still do many proofs involving bisimulation if the proof system included induction and co-induction inference rules. We have illustrated with a simple example in Section 7 how such a proof might be done. Inference rules for induction and co-induction appropriate for the sequent calculus have been presented in [Momigliano and Tiu 2003] and a version of these rules that also involves the  $\nabla$  quantifier has been presented in the first author's PhD thesis [Tiu 2004]. Open bisimulation, however, has not been studied in this setting. We plan to investigate further how these stronger proof systems can be used to establish properties about  $\pi$ -calculus expressions with infinite behaviors.

Specifications of operational semantics using a logic should make it possible to formally prove properties concerning that operational semantics. This was the case, for example, with specifications of the evaluation and typing of simple functional and imperative programming languages: a number of common theorems (determinacy of evaluation, subject-reduction, etc) can be naturally inferred using logical specifications [McDowell and Miller 2002]. We plan to investigate using our logic (also incorporating rules for induction and co-induction) for formally proving parts of the theory of the  $\pi$ -calculus. It seems, for example, rather transparent to prove that open bisimilarity is a congruence in our setting (see [Ziegler et al. 2005] for a more general class of congruence relations).

## 10. CONCLUSION

In this paper we presented a meta-logic that allows for declarative specifications of judgments related to the  $\pi$ -calculus. These specifications are done entirely within

the logic and without any additional side conditions. The management of name bindings in the specification of one-step transition, bisimulation, and modal logic is handled completely by the logic's three levels of binding, namely,  $\lambda$ -bindings within terms, the formula-level binders (quantifiers)  $\forall$ ,  $\exists$ , and  $\nabla$ , and the proof-level bindings for eigenvariables and local (generic) contexts.

This paper can be seen as part of a tradition of treating syntax more abstractly. The early, formal treatments of syntax by, for example, Church and Gödel, formalized terms and formulas as strings. Eventually, that treatment of syntax was replaced by more abstract objects such as parse trees: it is on parse trees that most syntactic descriptions of the  $\lambda$ -calculus and  $\pi$ -calculus are now given. Unfortunately, parse trees do not come equipped with primitive notions of bindings. To fix that problem, for example, Prawitz introduced "discharge functions" [Prawitz 1965] and de Bruijn introduced "nameless dummies" [de Bruijn 1972]. The move from parse trees to  $\lambda$ -trees, along with the use of a logic able to deal intimately with syntactic abstractions, is another way to fix this problem.

A significant part of this paper deals with establishing adequacy results that show a formal connection between the "standard" definitions of judgments concerning the  $\pi$ -calculus and the definitions given in logic (see the electronic appendix for the details). These adequacy results are all rather tedious and shallow but seem necessary to ensure that we have not invented our own problems for which we provide good solutions. It would seem, however, that the tediousness nature of the adequacy results can be attributed to the large gap between our proof-theory approach and the "standard" approach used to encode the  $\pi$ -calculus: now that some of these basic adequacy results have been written down, the adequacy results for any additional logical specifications using  $\lambda$ -tree syntax should follow more immediately.

We note that our effort in developing a proof theoretic setting for the  $\pi$ -calculus has led us to find new description for, in particular, the underlying assumptions on names in open and late bisimulations. This examination has led us to characterize the differences between open and late bisimulations in a simple and logical fashion: in particular, as the difference in name quantification and in the assumption about decidability of name equality.

## ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library by visiting the following URL: <http://www.acm.org/pubs/citations/journals/jn/20YY-V-N/p1-tiu>.

## ACKNOWLEDGMENTS

We are grateful to the reviewers of earlier drafts of this paper for their detailed and useful comments. We also benefited from support from INRIA through the "Equipes Associées" Slimmer, and from the Australian Research Council through the Discovery Projects funding scheme (project number DP0880549).

## REFERENCES

- ANDREOLI, J.-M. 1992. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation* 2, 3, 297–347.

- BAELDE, D., GACEK, A., MILLER, D., NADATHUR, G., AND TIU, A. 2007. The Bedwyr system for model checking over syntactic expressions. In *21th Conference on Automated Deduction (CADE)*, F. Pfenning, Ed. Number 4603 in LNAI. Springer, 391–397.
- BAELDE, D. AND MILLER, D. 2007. Least and greatest fixed points in linear logic. In *International Conference on Logic for Programming and Automated Reasoning (LPAR)*, N. Dershowitz and A. Voronkov, Eds. LNCS, vol. 4790. 92–106.
- BASU, S., MUKUND, M., RAMAKRISHNAN, C. R., RAMAKRISHNAN, I. V., AND VERMA, R. M. 2001. Local and symbolic bisimulation using tabled constraint logic programming. In *International Conference on Logic Programming (ICLP)*. LNCS, vol. 2237. Springer, Paphos, Cyprus, 166–180.
- BENGTSON, J. AND PARROW, J. 2007. Formalising the  $\pi$ -calculus using nominal logic. In *Proceedings of FOSSACS 2007*. LNCS, vol. 4423. Springer, 63–77.
- BOREALE, M. 2001. Symbolic trace analysis of cryptographic protocols. In *Proceedings of ICALP 2001*. LNCS, vol. 2076. Springer-Verlag, 667–681.
- BOREALE, M. AND NICOLA, R. D. 1996. A symbolic semantics for the  $\pi$ -calculus. *Information and Computation* 126, 1 (Apr.), 34–52.
- CHURCH, A. 1940. A formulation of the simple theory of types. *J. of Symbolic Logic* 5, 56–68.
- DE BRULIN, N. G. 1972. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser Theorem. *Indag. Math.* 34, 5, 381–392.
- DESPEYROUX, J. 2000. A higher-order specification of the  $\pi$ -calculus. In *Proc. of the IFIP International Conference on Theoretical Computer Science, IFIP TCS'2000, Sendai, Japan, August 17-19, 2000*. 425–439.
- ERIKSSON, L.-H. 1991. A finitary version of the calculus of partial inductive definitions. In *Proceedings of the Second International Workshop on Extensions to Logic Programming*, L.-H. Eriksson, L. Hallnäs, and P. Schroeder-Heister, Eds. LNAI, vol. 596. Springer-Verlag, 89–134.
- GABBAY, M. J. 2003. The  $\pi$ -calculus in FM. In *Thirty-five years of Automath*, F. Kamareddine, Ed. Kluwer, 80–149.
- GABBAY, M. J. AND PITTS, A. M. 2001. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing* 13, 341–363.
- GENTZEN, G. 1969. Investigations into logical deductions. In *The Collected Papers of Gerhard Gentzen*, M. E. Szabo, Ed. North-Holland, Amsterdam, 68–131.
- GIRARD, J.-Y. 1992. A fixpoint theorem in linear logic. An email posting to the mailing list linear@cs.stanford.edu.
- HALLNÄS, L. AND SCHROEDER-HEISTER, P. 1991. A proof-theoretic approach to logic programming. II. Programs as definitions. *J. of Logic and Computation* 1, 5 (Oct.), 635–660.
- HENNESSY, M. AND LIN, H. 1995. Symbolic bisimulations. *Theoretical Computer Science* 138, 2 (Feb.), 353–389.
- HIRSCHKOFF, D. 1997. A full formalization of pi-calculus theory in the Calculus of Constructions. In *International Conference on Theorem Proving in Higher Order Logics (TPHOLs'97)*, E. Gunter and A. Felty, Eds. Number 1275 in LNCS. Murray Hill, New Jersey, 153–169.
- HONSELL, F., LENISA, M., MONTANARI, U., AND PISTORE, M. 1998. Final semantics for the  $\pi$ -calculus. In *Proc. of PROCOMET'98*.
- HONSELL, F., MICULAN, M., AND SCAGNETTO, I. 2001.  $\pi$ -calculus in (co)inductive type theories. *Theoretical Computer Science* 2, 253, 239–285.
- HUET, G. 1975. A unification algorithm for typed  $\lambda$ -calculus. *Theoretical Computer Science* 1, 27–57.
- HUET, G. AND LANG, B. 1978. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica* 11, 31–55.
- LIANG, C. AND MILLER, D. 2007. Focusing and polarization in intuitionistic logic. In *CSL 2007: Computer Science Logic*, J. Duparc and T. A. Henzinger, Eds. LNCS, vol. 4646. Springer, 451–465. Extended version to appear in TCS.

- MCDOWELL, R. AND MILLER, D. 2000. Cut-elimination for a logic with definitions and induction. *Theoretical Computer Science* 232, 91–119.
- MCDOWELL, R. AND MILLER, D. 2002. Reasoning with higher-order abstract syntax in a logical framework. *ACM Trans. on Computational Logic* 3, 1, 80–136.
- MCDOWELL, R., MILLER, D., AND PALAMIDESSI, C. 2003. Encoding transition systems in sequent calculus. *Theoretical Computer Science* 294, 3, 411–437.
- MILLER, D. 1991. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. of Logic and Computation* 1, 4, 497–536.
- MILLER, D. 1992. Unification under a mixed prefix. *Journal of Symbolic Computation* 14, 4, 321–358.
- MILLER, D. 2000. Abstract syntax for variable binders: An overview. In *Computational Logic - CL 2000*, J. Lloyd and et. al., Eds. Number 1861 in LNAI. Springer, 239–253.
- MILLER, D. AND NADATHUR, G. 1986. Some uses of higher-order logic in computational linguistics. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Morristown, New Jersey, 247–255.
- MILLER, D. AND NADATHUR, G. 1987. A logic programming approach to manipulating formulas and programs. In *IEEE Symposium on Logic Programming*, S. Haridi, Ed. San Francisco, 379–388.
- MILLER, D., NADATHUR, G., PFENNING, F., AND SCEDROV, A. 1991. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic* 51, 125–157.
- MILLER, D. AND PALAMIDESSI, C. 1999. Foundational aspects of syntax. *ACM Computing Surveys* 31.
- MILLER, D. AND TIU, A. 2003. A proof theory for generic judgments: An extended abstract. In *18th Symp. on Logic in Computer Science*, P. Kolaitis, Ed. IEEE, 118–127.
- MILLER, D. AND TIU, A. 2005. A proof theory for generic judgments. *ACM Trans. on Computational Logic* 6, 4 (Oct.), 749–783.
- MILNER, R., PARROW, J., AND WALKER, D. 1992. A calculus of mobile processes, Part II. *Information and Computation* 100, 1, 41–77.
- MILNER, R., PARROW, J., AND WALKER, D. 1993. Modal logics for mobile processes. *Theoretical Computer Science* 114, 1, 149–171.
- MOMIGLIANO, A. AND TIU, A. 2003. Induction and co-induction in sequent calculus. In *Post-proceedings of TYPES 2003*, M. Coppo, S. Berardi, and F. Damiani, Eds. Number 3085 in LNCS. 293–308.
- NADATHUR, G. 2005. A treatment of higher-order features in logic programming. *Theory and Practice of Logic Programming* 5, 3, 305–354.
- NADATHUR, G. AND LINNELL, N. 2005. Practical higher-order pattern unification with on-the-fly raising. In *ICLP 2005: 21st International Logic Programming Conference*. LNCS, vol. 3668. Springer, Sitges, Spain, 371–386.
- NADATHUR, G. AND MILLER, D. 1988. An Overview of  $\lambda$ Prolog. In *Fifth International Logic Programming Conference*. MIT Press, Seattle, 810–827.
- NADATHUR, G. AND MITCHELL, D. J. 1999. System description: Teyjus — A compiler and abstract machine based implementation of  $\lambda$ Prolog. In *16th Conference on Automated Deduction (CADE)*, H. Ganzinger, Ed. Number 1632 in LNAI. Springer, Trento, 287–291.
- NIPKOW, T. 1993. Functional unification of higher-order patterns. In *Proc. 8th IEEE Symposium on Logic in Computer Science (LICS 1993)*, M. Vardi, Ed. IEEE, 64–74.
- PAULSON, L. C. 1986. Natural deduction as higher-order resolution. *Journal of Logic Programming* 3, 237–258.
- PAULSON, L. C. 1990. Isabelle: The next 700 theorem provers. In *Logic and Computer Science*, P. Odifreddi, Ed. Academic Press, 361–386.
- PFENNING, F. AND ELLIOTT, C. 1988. Higher-order abstract syntax. In *Proceedings of the ACM-SIGPLAN Conference on Programming Language Design and Implementation*. ACM Press, 199–208.

- PFENNING, F. AND SCHÜRMAN, C. 1999. System description: Twelf — A meta-logical framework for deductive systems. In *16th Conference on Automated Deduction (CADE)*, H. Ganzinger, Ed. Number 1632 in LNAI. Springer, Trento, 202–206.
- PITTS, A. M. 2003. Nominal logic, A first order theory of names and binding. *Information and Computation* 186, 2, 165–193.
- PLOTKIN, G. 1981. A structural approach to operational semantics. DAIMI FN-19, Aarhus University, Aarhus, Denmark. Sept.
- PRAWITZ, D. 1965. *Natural Deduction*. Almqvist & Wiksell, Uppsala.
- RÖCKL, C., HIRSCHKOFF, D., AND BERGHOFER, S. 2001. Higher-order abstract syntax with induction in Isabelle/HOL: Formalizing the pi-calculus and mechanizing the theory of contexts. In *Proc. FOSSACS'01*, F. Honsell and M. Miculan, Eds. LNCS, vol. 2030. Springer, 364–378.
- SANGIORGI, D. 1996. A theory of bisimulation for the  $\pi$ -calculus. *Acta Informatica* 33, 1, 69–97.
- SANGIORGI, D. AND WALKER, D. 2001.  *$\pi$ -Calculus: A Theory of Mobile Processes*. Cambridge University Press.
- SCHROEDER-HEISTER, P. 1993. Rules of definitional reflection. In *Eighth Annual Symposium on Logic in Computer Science*, M. Vardi, Ed. IEEE Computer Society Press, IEEE, 222–232.
- STÄRK, R. F. 1994. Cut-property and negation as failure. *International Journal of Foundations of Computer Science* 5, 2, 129–164.
- TIU, A. 2004. A logical framework for reasoning about logical specifications. Ph.D. thesis, Pennsylvania State University.
- TIU, A. 2005. Model checking for  $\pi$ -calculus using proof search. In *CONCUR*, M. Abadi and L. de Alfaro, Eds. LNCS, vol. 3653. Springer, 36–50.
- TIU, A. AND MILLER, D. 2004. A proof search specification of the  $\pi$ -calculus. In *3rd Workshop on the Foundations of Global Ubiquitous Computing*. ENTCS, vol. 138. 79–101.
- TIU, A., NADATHUR, G., AND MILLER, D. 2005. Mixing finite success and finite failure in an automated prover. In *Empirically Successful Automated Reasoning in Higher-Order Logics (ESHOL'05)*. 79–98.
- URBAN, C. AND TASSON, C. 2005. Nominal techniques in Isabelle/HOL. In *20th Conference on Automated Deduction (CADE)*, R. Nieuwenhuis, Ed. LNCS, vol. 3632. Springer, 38–53.
- ZIEGLER, A., MILLER, D., AND PALAMIDESSI, C. 2005. A congruence format for name-passing calculi. In *Structural Operational Semantics (SOS'05)*. Electronic Notes in Theoretical Computer Science. Elsevier Science B.V., Lisbon, Portugal, 169–189.

Received May 2008; revised December 2008; accepted February 2009