Model Checking for π-Calculus Using Proof Search

Alwen Tiu

INRIA Lorraine
615 rue du Jardin Botanique
54602 Villers-lès-Nancy, France
Alwen.Tiu@loria.fr

Abstract. Model checking for transition systems specified in π-calculus has been a difficult problem due to the infinite-branching nature of input prefix, name-restriction and scope extrusion. We propose here an approach to model checking for π-calculus by encoding it into a logic which supports reasoning about bindings and fixed points. This logic, called $FO\lambda^{\Delta\nabla}$, is a conservative extension of Church's Simple Theory of Types with a "generic" quantifier. By encoding judgments about transitions in pi-calculus into this logic, various conditions on the scoping of names and restrictions on name instantiations are captured naturally by the quantification theory of the logic. Moreover, standard implementation techniques for (higher-order) logic programming are applicable for implementing proof search for this logic, as illustrated in a prototype implementation discussed in this paper. The use of logic variables and eigenvariables in the implementation allows for exploring the state space of processes in a symbolic way. Compositionality of properties of the transitions is a simple consequence of the meta theory of the logic (i.e., cut elimination). We illustrate the benefits of specifying systems in this logic by studying several specifications of modal logics for pi-calculus. These specifications are also executable directly in the prototype implementation of $FO\lambda^{\Delta\nabla}$.

1 Introduction

The π-calculus [16] provides a simple yet powerful framework for specifying communication systems with evolving communication structures. Its expressiveness derives mainly from the possibility of passing communication channels (names), restricting the scope of channels and scope extrusion. These are precisely the features that make model checking for π-calculus difficult. Model checking has traditionally been done with transitions which have finite state models. The name passing feature alone (input prefix) in π-calculus would yield infinite-branching transition systems, if implemented naively. Scope and scope extrusion add another significant layer of complexity, since in model checking the transition systems one has to take into account the exact scope and identity of various channel names. This is a problem which has been studied extensively, of course, due to the importance of π-calculus. A non-exhaustive list of existing works includes the work on *history dependent automata* [6] model of mobile processes, specific programming logics and decision procedures for model checking mobile processes [3, 4], the spatial logic model checker [2] using Gabbay-Pitts permutation techniques [7], and implementation using logic programming [27].

The approach to model checking π-caculus (or mobile processes in general) taken in this paper is based on the proof theory of sequent calculus, by casting the problem of reasoning about scoping and name-instantiation into the more general setting of proof theory for quantifiers in formal logic. More specifically, we encode judgments about transitions in π-calculus and several modal logics for π-calculus [17] into a meta logic, and proof search is used to model the operational semantics of these judgments. This meta logic, called $FO\lambda^{\Delta\nabla}$ [15], is an extension of Church's Simple Theory of Types (but without quantification over propositions, so the logic is essentially first-order) with a proof theoretical notion of *definitions* [22] and a new "generic" quantifier, ∇. The quantifier ∇, roughly summarized, facilitates reasoning about binders (more details will be given later). We summarize our approach as follows.

λ-tree syntax. We use the λ-tree syntax [14] to encode syntax with bindings. It is a variant of higher-order abstract syntax, where syntax of arbitrary system is encoded as λ-terms and the λ-abstraction is used to encode bindings within expressions. One of the advantages of adopting λ-tree syntax, or higher-order abstract syntax in general, is that all the side conditions involving bindings such as scoping of variables, α-conversion, etc., are handled uniformly at the level of the abstract syntax, using the known notions in λ-calculus. Another one is that efficient implementation techniques for manipulating this abstract syntax are well-understood, e.g., algorithms for doing pattern-matching and unification of simply typed λ-terms.

Definitional reflection. Proof search in traditional logics, e.g., variants of Gentzen's LJ or LK, is limited to model the *may-behaviour* of computation system. *Must-behaviour*, eg., notions like bisimulations, or in the interest of this paper, satisfiability of modal formulae, cannot be expressed directly in these logics. To encode such notions, it is necessary to move to a richer logic. Recent developments in the proof theory of *definitions* [10, 11] have shown that must-behaviour can indeed be captured in logics extended with this proof-theoretical notion of definitions. In a logic with definitions, an atomic proposition may be "defined" by another formula (which may contain the atomic proposition itself). Thus, a definition can be seen as expressing a fixed point equation. Proof search for a defined atomic formula is done by unfolding the definition of the formula. In the logic with definitions used in this paper, a provable formula like $\forall x.px \supset qx$, where p and q are some defined predicates, expresses the fact that for every term t and for every proof (computation) of pt, there is a proof (computation) of qt. If p and q are predicates encoding one-step transitions, then this formula expresses one-step simulation. If q is an encoding of some assertion in modal logics, then the formula expresses the fact that the modal assertion is true for all reachable "next states" associated with the transition relation encoded by p.

Eigenvariables and ∇. In proof search for a universal quantified formula, e.g., $\forall x.Bx$, the quantified variable x is replaced by a new constant c, and proof search is continued on Bc. Such constants are called *eigenvariables*, and in traditional intuitionistic or classical logic, they play the role of scoped constants as they are created dynamically as proof search progresses and are not instantiated during the proof search. In the meta theory of the logic, eigenvariables play the role of place holder for values, since from a proof for Bc where c is an eigenvariable, one can obtain a proof of Bt for any term t by substituting t into c. In the proof theory of definitions, these dual roles of eigenvariables are internalized in the proof rules of the logic. In particular, in unfolding a definition in a negative context (left-hand side of a sequent), eigenvariables are treated as variables, and in the positive context they are treated as scoped constants. Computation (or transition) states can be encoded using eigenvariables. This in conjunction with definitions allows for exploring the state space of a transition system symbolically.

Since eigenvariables are not used here entirely as scoped constants, to account for scoped names we make use of the ∇-quantifier, first introduced in the logic $FO\lambda^{\Delta\nabla}$ [15], to help encode the notion of "generic judgment" that occurs commonly when reasoning with λ-tree syntax. The ∇ quantifier is used to introduce new elements into a type within a given scope. In particular, a reading of the truth condition for $\nabla x_\gamma.Bx$ is something like: if given a new element, say c, of type γ, then check the truth of Bc. The difference between ∇ and $\forall$ appears in their interaction with definition rules: the constants introduced by ∇ are not subject to instantiation. Note that intended meaning of the ∇-quantifier is rather different from the "new" quantifier of Gabbay and Pitts [7], although they both address the same issue from a pragmatic point of view. In particular, in Gabbay-Pitts setting, an infinite number of names is assumed to be given, and equality between two names are decidable. In our approach here, no such assumptions are made concerning the type of names, not even the assumption that it is non-empty. Instead, new names are generated dynamically when needed, such as when inferring a transition involving extrusion of scopes.

An implementation of proof search. Proof search for $FO\lambda^{\Delta\nabla}$ can be implemented quite straightforwardly, using only the standard tools and techniques used in higher-order logic programming and theorem provers. An automated proof search engine for a fragment of $FO\lambda^{\Delta\nabla}$ has been implemented [24]. It was essentially done by plugging together different existing implementation: higher-order pattern unification [12, 18], stream-based approach to back-tracking, and parser for λ-terms. On top of this prototype implementation several specifications of process calculi and bisimulation have been implemented.[1] In most cases, the specifications are implemented almost without any modifications (except for the type-setting of course). A specification of modal logics has also been implemented in this prototype.

Outline of the papers. The rest of the paper is organized as follows. In Section 2, an overview of the meta logic $FO\lambda^{\Delta\nabla}$ is given. This is followed by the

[1] The prototype implementation along with the example specifications can be downloaded from the author's website: http://www.loria.fr/~tiu.

specification of the operational semantics of the late π-calculus in Section 3. The materials in these two sections have appeared in [15, 26]; they are included here since the main results of this paper are built on them. Section 4 presents the specification of modal logics introduced in [17] along with the adequacy results. Section 5 gives an overview of a prototype implementation of $FO\lambda^{\Delta\nabla}$ in which the specification of modal logics is implemented. These two sections constitute the main contribution of this paper. Section 6 discusses related and future work. Detailed proofs are in the Appendix.

2 Overview of the meta logic

The logic $FO\lambda^{\Delta\nabla}$ (pronounced "fold-nabla") is presented using a sequent calculus that is an extension of Gentzen's system LJ for first-order intuitionistic logic. A *sequent* is an expression of the form $B_1, \ldots, B_n \longrightarrow B_0$ where $B_0, \ldots, B_n$ are formulas and the elongated turnstile $\longrightarrow$ is the sequent arrow. To the left of the turnstile is a multiset: thus repeated occurrences of a formula are allowed. If the formulas $B_0, \ldots, B_n$ contain free variables, they are considered universally quantified outside the sequent, in the sense that if the above sequent is provable than 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 π-calculus could be to use eigenvariables to encode names in π-calculus, but this is certainly problematic. For example, if we have a proof for the sequent $\longrightarrow Pxy$, where x and y are different eigenvariables, then logic dictates that the sequent $\longrightarrow Pzz$ is also provable (given that the reading of eigenvariables is universal). 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 [15] to encode "generic judgments"). In particular, sequents in $FO\lambda^{\Delta\nabla}$ are of the form

$$\Sigma\,;\, \sigma_1 \triangleright B_1, \ldots, \sigma_n \triangleright B_n \longrightarrow \sigma_0 \triangleright B_0$$

where Σ is a *global signature*, i.e., the set of eigenvariables whose scope is over the whole sequent, and σ_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 Σ and σ_i will admit α-conversion by systematically changing the names of variables in signatures as well as those in their scope, following the usual convention of the λ-calculus. The meaning of eigenvariables is as before, only 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 σ is empty. We shall often write the list σ as a string of variables, e.g., a judgment $(x_1, x_2, x_3) \triangleright B$ will be written as $x_1 x_2 x_3 \triangleright B$. If the list x_1, x_2, x_3 is known from context we shall also abbreviate the judgment as $\bar{x} \triangleright B$.

The logical constants of $FO\lambda^{\Delta\nabla}$ are $\forall$ (universal quantifier), $\exists$ (existential quantifier), ∇, $\wedge$ (conjunction), $\vee$ (disjunction), $\supset$ (implication), $\top$ (true) and $\bot$ (false). The inference rules for the quantifiers are given in Figure 1. The complete set of inference rules can be found in [15]. Since we do not allow quantification over predicates, this logic is proof-theoretically similar to first-order logic (hence, the letters FO in $FO\lambda^{\Delta\nabla}$).

$$\frac{\Sigma, \sigma \vdash t : \gamma \qquad \Sigma\,;\, \sigma \triangleright B[t/x], \Gamma \longrightarrow \mathcal{C}}{\Sigma\,;\, \sigma \triangleright \forall_\gamma x.B, \Gamma \longrightarrow \mathcal{C}}\ \forall\mathcal{L} \qquad \frac{\Sigma, h\,;\, \Gamma \longrightarrow \sigma \triangleright B[(h\ \sigma)/x]}{\Sigma\,;\, \Gamma \longrightarrow \sigma \triangleright \forall x.B}\ \forall\mathcal{R}$$

$$\frac{\Sigma, h\,;\, \sigma \triangleright B[(h\ \sigma)/x], \Gamma \longrightarrow \mathcal{C}}{\Sigma\,;\, \sigma \triangleright \exists x.B, \Gamma \longrightarrow \mathcal{C}}\ \exists\mathcal{L} \qquad \frac{\Sigma, \sigma \vdash t : \gamma \qquad \Sigma\,;\, \Gamma \longrightarrow \sigma \triangleright B[t/x]}{\Sigma\,;\, \Gamma \longrightarrow \sigma \triangleright \exists_\gamma x.B}\ \exists\mathcal{R}$$

$$\frac{\Sigma\,;\, (\sigma, y) \triangleright B[y/x], \Gamma \longrightarrow \mathcal{C}}{\Sigma\,;\, \sigma \triangleright \nabla x\ B, \Gamma \longrightarrow \mathcal{C}}\ \nabla\mathcal{L} \qquad \frac{\Sigma\,;\, \Gamma \longrightarrow (\sigma, y) \triangleright B[y/x]}{\Sigma\,;\, \Gamma \longrightarrow \sigma \triangleright \nabla x\ B}\ \nabla\mathcal{R}$$

Fig. 1. The quantifier rules of $FO\lambda^{\Delta\nabla}$.

During the search for proofs (reading rules bottom up), inference rules for $\forall$ and $\exists$ quantifier place new eigenvariables into the global signature while the inference rules for ∇ place them into the local signature. In the $\forall\mathcal{R}$ and $\exists\mathcal{L}$ rules, raising [13] is used when moving the bound variable x, which can range over the variables in both the global signature and the local signature σ, with the variable h that can only range over variables in the global signature: so as not to miss substitution terms, the variable x is replaced by the term $(h\, x_1 \dots x_n)$, which we shall write simply as $(h\, \sigma)$, where σ is the list $x_1, \dots, x_n$ (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 Σ and σ. This is presented in the rule 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 in $\nabla x\ B$.

The standard inference rules of logic express introduction rules for logical constants. 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 [22], Eriksson [5], Girard [8], Stärk [23] and McDowell and Miller [10]. These inference rules for definitions allow for modest reasoning about the fixed points of definitions.

Definition 1. *A* definition clause *is written* $\forall \bar{x}[p\,\bar{t} \stackrel{\triangle}{=} B]$*, 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* $\stackrel{\triangle}{=}$ *is used simply to indicate a definitional clause: it is not a logical connective. The predicate* p *occurs strictly positively in* B*, that is, it does not occur to the left of any* $\supset$ *(implication).*

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

$$\forall h_1 \ldots \forall h_n.\bar{y} \triangleright H\theta \stackrel{\triangle}{=} \bar{y} \triangleright B\theta$$

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

The introduction rules for a defined judgment are as follow. 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.

$$\frac{\{\Sigma\theta\,;\,\mathcal{B}\theta, \Gamma\theta \longrightarrow \mathcal{C}\theta \mid \theta \in CSU(\mathcal{A}, \mathcal{H}) \text{ for some clause } \mathcal{H} \stackrel{\triangle}{=} \mathcal{B}\}}{\Sigma\,;\,\mathcal{A}, \Gamma \longrightarrow \mathcal{C}}\ \mathit{def}\mathcal{L}$$

$$\frac{\Sigma\,;\,\Gamma \longrightarrow \mathcal{B}\theta}{\Sigma\,;\,\Gamma \longrightarrow \mathcal{A}}\ \mathit{def}R, \quad \text{where } \mathcal{H} \stackrel{\triangle}{=} \mathcal{B} \text{ is a definition clause and } \mathcal{H}\theta = \mathcal{A}$$

In the above rules, we apply substitution to judgments. The result of applying a substitution θ to a generic judgment $x_1, \ldots, x_n \triangleright B$, written as $(x_1, \ldots, x_n \triangleright B)\theta$, is $y_1, \ldots, y_n \triangleright B'$, if $(\lambda x_1 \ldots \lambda x_n.B)\theta$ is equal (modulo λ-conversion) to $\lambda y_1 \ldots \lambda y_n.B'$. If Γ is a multiset of generic judgments, then $\Gamma\theta$ is the multiset $\{J\theta \mid J \in \Gamma\}$. In the $\mathit{def}\mathcal{L}$ rule, we use the notion of *complete set of unifiers* (CSU) [9]. 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 θ 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 θ'. In all the applications of $\mathit{def}\mathcal{L}$ 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 $\mathit{def}\mathcal{L}$ denotes a signature obtained from Σ by removing the variables in the domain of θ and adding the variables in the range of θ. In the $\mathit{def}\mathcal{L}$ rule, reading the rule bottom-up, eigenvariables can be instantiated in the premise, while in the $\mathit{def}R$ rule, eigenvariables are not instantiated. The set that is the premise of the $\mathit{def}\mathcal{L}$ 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.

3 Logical specification of one-step transition

We consider the late transition system for the π-calculus in [16], but we shall follow the operational semantics of π-calculus presented in [21]. The syntax of processes is defined as follows

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

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

One-step transition in the π-calculus is denoted by $\mathtt{P} \xrightarrow{\alpha} \mathtt{Q}$, where P and Q are processes and α is an action. The kinds of actions are *the silent action* τ, *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 like we did with processes, we use $\mathrm{fn}(\alpha)$, $\mathrm{bn}(\alpha)$ and $\mathrm{n}(\alpha)$ to denote free names, bound names, and names in α. An action without binding occurrences of names is a *free action*, otherwise it is a *bound action*.

We encode the syntax of process expressions using higher-order syntax as follows. We shall require three primitive syntactic categories: n for names, p for processes, and a for actions, and the constructors corresponding to the operators in π-calculus. We do not assume any inhabitants of type n, therefore in our encoding a free name is translated to a variable of type n, which can later be either universally quantified or ∇-quantified, depending on whether we want to treat a certain name as instantiable or not. In this paper, however, we consider only ∇-quantified names. Universally quantified names are used in the encoding of open bisimulation in [26]. Since the rest of this paper is about the π-calculus, the ∇ quantifier will from now on only be used at type n. To encode actions, we use $\tau : a$ (for the silent action), and the two constants $\downarrow$ and $\uparrow$, both of type $n \to n \to a$ for building input and output actions. 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 η-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 $\downarrow x$). The process constructors are encoded using the following constants:

$$0 : p \qquad \tau : p \to p \qquad \mathit{out} : n \to n \to p \to p \qquad \mathit{in} : n \to (n \to p) \to p$$
$$+ : p \to p \to p \qquad | : p \to p \to p \qquad ! : p \to p$$
$$\mathit{match} : n \to n \to p \to p \qquad \nu : (n \to p) \to p$$

We use two predicates to encode the one-step transition semantics for the π-calculus. The predicate $\cdot \stackrel{\cdot}{\longrightarrow} \cdot$ of type $p \to a \to p \to o$ encodes transitions involving free values and the predicate $\cdot \stackrel{\cdot}{\rightharpoonup} \cdot$ of type $p \to (n \to a) \to (n \to p) \to o$ encodes transitions involving bound values. The precise translation of π-calculus syntax into simply typed λ-terms is given in the following definition.

Definition 2. *The following function* $[\![.]\!]$ *translates from process expressions to* $\beta\eta$*-long normal terms of type* p.

$$
\begin{array}{lll}
[\![0]\!] = 0 & [\![\mathtt{P}+\mathtt{Q}]\!] = [\![\mathtt{P}]\!] + [\![\mathtt{Q}]\!] & [\![\mathtt{P}|\mathtt{Q}]\!] = [\![\mathtt{P}]\!] \mid [\![\mathtt{Q}]\!] \\
[\![\tau.\mathtt{P}]\!] = \tau\ [\![\mathtt{P}]\!] & [\![[x=y]\mathtt{P}]\!] = match\ x\ y\ [\![\mathtt{P}]\!] & [\![\bar{x}y.\mathtt{P}]\!] = out\ x\ y\ [\![\mathtt{P}]\!] \\
[\![x(y).\mathtt{P}]\!] = in\ x\ \lambda y.[\![\mathtt{P}]\!] & [\![(x)\mathtt{P}]\!] = \nu\lambda x.[\![\mathtt{P}]\!] & [\![!\mathtt{P}]\!] = ![\![\mathtt{P}]\!]
\end{array}
$$

The one-step transition judgments are translated to atomic formulas as follows (we overload the symbol $[\![.]\!]$*).*

$$
\begin{array}{ll}
[\![\mathtt{P} \xrightarrow{\bar{x}y} \mathtt{Q}]\!] = [\![\mathtt{P}]\!] \xrightarrow{\uparrow xy} [\![\mathtt{Q}]\!] & [\![\mathtt{P} \xrightarrow{x(y)} \mathtt{Q}]\!] = [\![\mathtt{P}]\!] \xrightharpoonup{\downarrow x} \lambda y.[\![\mathtt{Q}]\!] \\
[\![\mathtt{P} \xrightarrow{\tau} \mathtt{Q}]\!] = [\![\mathtt{P}]\!] \xrightarrow{\tau} [\![\mathtt{Q}]\!] & [\![\mathtt{P} \xrightarrow{\bar{x}(y)} \mathtt{Q}]\!] = [\![\mathtt{P}]\!] \xrightharpoonup{\uparrow x} \lambda y.[\![\mathtt{Q}]\!] \\
[\![\mathtt{P} \xrightarrow{xy} \mathtt{Q}]\!] = [\![\mathtt{P}]\!] \xrightarrow{\downarrow xy} [\![\mathtt{Q}]\!] &
\end{array}
$$

We abbreviate $\nu\lambda x.P$ as simply $\nu x.P$. Notice that when τ is written as a prefix, it has type $p \to p$, and when it is written as an action, it has type a.

The operational semantics of the late transition system for π-calculus is given as a definition, called $\mathbf{D}_\pi$, in Figure 2. In the figure, we omit the symmetric cases for par, sum, close and com. 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 $n \to a$ and $n \to p$.

Notice that as a consequence of the use of HOAS in the encoding, the complicated side conditions in the original specifications of π-calculus [16] are no longer present. For example, the side condition that $X \neq y$ in the open rule is implicit, since X is outside the scope of y and therefore cannot be instantiated with y. The adequacy of our encoding is stated in the following lemma and proposition (their proofs can be found in [25]).

Lemma 3. *The function* $[\![.]\!]$ *is a bijection between* α*-equivalence classes of expressions.*

Proposition 4. *Let* $\mathtt{P}$ *and* $\mathtt{Q}$ *be processes and* α *an action. Let* $\bar{n}$ *be a list of free names containing the free names in* $\mathtt{P}$*,* $\mathtt{Q}$*, and* α*. The transition* $\mathtt{P} \xrightarrow{\alpha} \mathtt{Q}$ *is derivable in* π*-calculus if and only if* $.\,;\,.\ \vdash \nabla\bar{n}.[\![\mathtt{P} \xrightarrow{\alpha} \mathtt{Q}]\!]$ *in* $FO\lambda^{\Delta\nabla}$ *with the definition* $\mathbf{D}_\pi$*.*

Note that since in the translation from π-calculus to $FO\lambda^{\Delta\nabla}$ free names are translated to ∇-quantified variables, to get the completeness of the encoding, it is necessary to show that the transition in π-calculus is invariant under free-name renaming. This has been shown in [16]. In fact, most of the properties of interest in π-calculus, such as bisimulation and satisfiability of modal formulae, are closed under free-name renaming [17].

$$
\begin{array}{rl}
\text{TAU:} & \tau\ P \xrightarrow{\tau} P \triangleq \top \\
\text{IN:} & in\ X\ M \xrightharpoonup{\downarrow X} M \triangleq \top \\
\text{OUT:} & out\ x\ y\ P \xrightarrow{\uparrow xy} P \triangleq \top \\
\text{MATCH:} & match\ x\ x\ P \xrightarrow{A} Q \triangleq P \xrightarrow{A} Q \\
 & match\ x\ x\ P \xrightharpoonup{A} Q \triangleq P \xrightharpoonup{A} Q \\
\text{SUM:} & P + Q \xrightarrow{A} R \triangleq P \xrightarrow{A} R \\
 & P + Q \xrightharpoonup{A} R \triangleq P \xrightharpoonup{A} R \\
\text{PAR:} & P \mid Q \xrightarrow{A} P' \mid Q \triangleq P \xrightarrow{A} P' \\
 & P \mid Q \xrightharpoonup{A} \lambda n(M\,n \mid Q) \triangleq P \xrightharpoonup{A} M \\
\text{RES:} & \nu n.Pn \xrightarrow{A} \nu n.Qn \triangleq \nabla n(Pn \xrightarrow{A} Qn) \\
 & \nu n.Pn \xrightharpoonup{A} \lambda m\ \nu n.P'nm \triangleq \nabla n(Pn \xrightharpoonup{A} P'n) \\
\text{OPEN:} & \nu y.My \xrightharpoonup{\uparrow X} M' \triangleq \nabla y(My \xrightarrow{\uparrow Xy} M'y) \\
\text{CLOSE:} & P \mid Q \xrightarrow{\tau} \nu y.My \mid Ny \triangleq \exists X.P \xrightharpoonup{\downarrow X} M \wedge Q \xrightharpoonup{\uparrow X} N \\
\text{COM:} & P \mid Q \xrightarrow{\tau} MY \mid Q' \triangleq \exists X.P \xrightharpoonup{\downarrow X} M \wedge Q \xrightarrow{\uparrow XY} Q' \\
\text{REP-ACT:} & !P \xrightarrow{A} P' |!P \triangleq P \xrightarrow{A} P' \\
 & !P \xrightharpoonup{X} \lambda y(My|!P) \triangleq P \xrightharpoonup{X} M \\
\text{REP-COM:} & !P \xrightarrow{\tau} (P' \mid M\ Y)|!P \triangleq \exists X.P \xrightarrow{\uparrow XY} P' \wedge P \xrightharpoonup{\downarrow X} M \\
\text{REP-CLOSE:} & !P \xrightarrow{\tau} \nu z.(Mz \mid Nz)|!P \triangleq \exists X.P \xrightharpoonup{\uparrow X} M \wedge P \xrightharpoonup{\downarrow X} N
\end{array}
$$

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

4 Specification of modal logics

We now consider the modal logics for π-calculus introduced in [17]. In order not to confuse meta-level ($FO\lambda^{\Delta\nabla}$) formulas (or connectives) with the formulas (connectives) of modal logics under consideration, we shall refer to the latter as object formulas (respectively, object connectives). We shall work only with object formulas which are in negation normal form, i.e., negation appears only at the level of atomic object formulas. As a consequence, we introduce explicitly each dual pair of the object connectives. Note that since the only atomic object formulas are either true or false, by de Morgan duality $\neg$true $\equiv$ false and $\neg$false $\equiv$ true. Therefore we are in effect working with positive formulas only. The syntax of the object formulas is given by

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

In each of the formulas (and their dual 'boxed'-formulas) $\langle \bar{x}(y) \rangle \mathtt{A}$, $\langle x(y) \rangle \mathtt{A}$, $\langle x(y) \rangle^{L} \mathtt{A}$ and $\langle x(y) \rangle^{E} \mathtt{A}$, the occurrence of y in parentheses is a binding occurrence whose

(a) Propositional connectives and *basic* modality:

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

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

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

Fig. 3. Modal logics for π-calculus in λ-tree syntax

scope is $\mathtt{A}$. We use $\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}$, possibly with subscripts or primes, to range over object formulas. Note that we consider only finite conjunction since the transition system we are considering is finitely branching, and therefore (as noted in [17]) infinite conjunction is not needed. Note also that we do not consider free input modality $\langle xy \rangle$ since we restrict ourselves to late transition system (but adding early transition rules and free input modality does not pose any difficulty). We consider object formulas equivalent up to renaming of bound variables.

We introduce the types o' to denote object-level propositions, and the following constants for encoding the object connectives.

$$
\begin{array}{cc}
\text{true} : o', \quad \text{false} : o', & \& : o' \to o' \to o', \quad \hat{\vee} : o' \to o' \to o' \\
\langle \cdot \dot{=} \cdot \rangle \cdot : n \to n \to o' \to o', & [\cdot \dot{=} \cdot] \cdot : n \to n \to o' \to o', \\
\langle \cdot \rangle \cdot : a \to o' \to o', & [\cdot] \cdot : a \to o' \to o', \\
\langle \downarrow \cdot \rangle \cdot : n \to (n \to o') \to o', & [\downarrow \cdot] \cdot : n \to\to (n \to o') \to o' \\
\langle \downarrow \cdot \rangle^l \cdot : n \to (n \to o') \to o', & [\downarrow \cdot]^l \cdot : n \to (n \to o') \to o' \\
\langle \downarrow \cdot \rangle^e \cdot : n \to (n \to o') \to o', & [\downarrow \cdot]^e \cdot : n \to (n \to o') \to o'
\end{array}
$$

The precise translation from object-level modal formulas to λ-tree syntax is given in the following.

Definition 5. *The following function* $[\![.]\!]$ *translates from object formulas to* $\beta\eta$*-long normal terms of type* o'.

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

The satisfaction relation $\models$ between processes and formulas are encoded using the same symbol, which is given the type $p \to o' \to o$. The inference rules for this satisfaction relation are given as definition clauses in Figure 3. Some of the definition clauses make use of the syntactic equality predicate, which is defined as the definition: $X = X \stackrel{\triangle}{=} \top$. Note that the symbol $=$ here is a predicate symbol written in infix notation. The inequality $x \neq y$ is an abbreviation for $x = y \supset \bot$.

We refer to the definition shown in Figure 3 as $\mathcal{DA}$. This definition corresponds to the modal logic $\mathcal{A}$ defined in [17]. However, this definition is not complete, in the sense that there are true assertion of modal logics which are not provable using this definition alone. For instance, the modal judgment

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

is valid, but its encoding in $FO\lambda^{\Delta\nabla}$ is not provable without additional assumptions. It turns out that the only assumption we need to get completeness is the axiom of excluded middle on names:

$$\forall x \forall y. x = y \vee x \neq y.$$

Note that since we allow dynamic creation of scoped names (via ∇), we must also state this axiom for arbitrary extension of local signatures. We therefore define the following set of excluded middles on arbitrary finite extension of local signatures

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

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

We shall now state the adequacy of the encoding of modal logics. The proof of the adequacy result can be found in the appendix.

Proposition 6. *Let* $\mathtt{P}$ *be a process, let* $\mathtt{A}$ *be an object formula. Then* $\mathtt{P} \models \mathtt{A}$ *if and only if for some list* $\bar{n}$ *containing the free names of* $(\mathtt{P}, \mathtt{A})$ *and some* $\mathcal{X} \subseteq_f \mathcal{E}$*, the sequent* $\mathcal{X} \longrightarrow \nabla \bar{n}.([\![\mathtt{P}]\!] \models [\![\mathtt{A}]\!])$ *is provable in* $FO\lambda^{\Delta\nabla}$ *with definition* $\mathcal{DA}$.

Note that we quantify free names in the process-formula pair in the above proposition since, as we have mentioned previously, we do not assume any constants of type n. Of course, such constants can be introduced without affecting

$$
\begin{array}{lll}
P \models_L \langle\uparrow X\rangle A & \triangleq & \exists P'(P \xrightarrow{\uparrow X} P' \wedge \nabla y.P'y \models_{y::L} Ay).\\
P \models_L [\uparrow X]A & \triangleq & \forall P'(P \xrightarrow{\uparrow X} P' \supset \nabla y.P'y \models_{y::L} Ay).\\
P \models_L \langle\downarrow X\rangle A & \triangleq & \exists P'(P \xrightarrow{\downarrow X} P' \wedge \nabla z \exists y.y \in (z :: L) \wedge P'y \models_{z::L} Ay).\\
P \models_L [\downarrow X]A & \triangleq & \forall P'(P \xrightarrow{\downarrow X} P' \supset \nabla z \forall y.y \in (z :: L) \supset P'y \models_{z::L} Ay).\\
P \models_L \langle\downarrow X\rangle^l A & \triangleq & \exists P'(P \xrightarrow{\downarrow X} P' \wedge \nabla z \forall y.y \in (z :: L) \supset P'y \models_{z::L} Ay).\\
P \models_L [\downarrow X]^l A & \triangleq & \forall P'(P \xrightarrow{\downarrow X} P' \supset \nabla z \exists y.y \in (z :: L) \wedge P'y \models_{z::L} Ay).
\end{array}
$$

Fig. 4. A more concrete specification with explicit names representation.

the provability of the satisfaction judgments, but for simplicity in the meta-theory we consider the more uniform approach using ∇-quantified variables to encode names in process and object formulas. Note that adequacy result stated in Proposition 6 subsumes the adequacy for the specifications of the sublogics of $\mathcal{A}$.

5 Implementation of proof search

We now give an overview of a prototype implementation of a fragment of $FO\lambda^{\Delta\nabla}$, in which the specification of modal logics given in the previous section is implemented. This implementation, called *Level 0/1 prover* [24], is based on the duality of finite success and *finite failure* in proof search, or equally, the duality of proof and refutation. In particular, the finite failure in proving a goal $\exists x.G$ should give us a proof of $\neg(\exists x.G)$ and vice versa. We experiment with a simple class of formulae which exhibits this duality. This class of formulae is given by the following grammar:

Level 0: $G := \top \mid \bot \mid A \mid G \wedge G \mid G \vee G \mid \exists x.G \mid \nabla x.G$
Level 1: $D := \top \mid \bot \mid A \mid D \wedge D \mid D \vee D \mid G \supset D \mid \exists x.D \mid \nabla x.D \mid \forall x.D$
atomic: $A := p\, t_1 \ldots t_n$

Notice that the level-0 formula is basically Horn-goal extended with ∇ to allow dynamic creation of names. Level-0 formula is used to encode transition systems (via definitions). Level-1 formula allows for reflecting on the provability of level-0 formulae, and hence exploring all the paths of the transition systems encoded at level-0.

The proof search implementation for level-0 formula is the standard logic-programming implementation. It is actually a subset of λProlog (with $\forall$ replacing ∇). That is, existentially quantified variables are replaced by logic variables, ∇-quantified variables are replaced with (scoped) constants. The non-standard part in Level 0/1 prover is the proof search for level-1 goals. Proof search for a level-1 goal $G_1 \supset G_2$ proceeds as follows:

1. Run the prover with the goal G_1, treating eigenvariables as logic variables.

2. If Step 1 fails, then proof search for $G_1 \supset G_2$ succeeds. Otherwise, collect all answer substitutions produced in Step 1, and for each answer susbtitution θ, proceed with proving $G_2\theta$

There is some restriction on the occurrence of logic variables in Step 2, which however does not affect the encoding of modal logics considered in this paper. We refer the interested readers to [24] for more details.

We now consider the problem of automating model-checking for a given process $\mathtt{P}$ against a given assertion $\mathtt{A}$ of sublogics of $\mathcal{A}$. There are two main difficulties in automating the model checking: when to use the excluded middle on names, and guessing how many names to be provided in advance. There seems to be two extremes in dealing with these problems: one in which excluded middles are omitted and the set of names are fixed to the free names of the processes and assertions involved, the other is to keep track of the set of free names explicitly and to instantiate any universally quantified name with all the names in this set. For the former, the implementation is straightforward: we simply use the specification given in Figure 3. The problem is of course that it is incomplete, although it may cover quite a number of interesting cases. We experiment here on the second approach using explicit handling of names which is complete but less efficient. The essential modifications to the specification in Figure 3 are those concerning input modalities. We list some modified clauses in Figure 4, the complete "implementation" can be found in an extended version of this paper. We shall refer to this definition as $\mathcal{DA}'$ The satisfiability relation $\models$ now takes an extra argument which is a list of names. The empty list is denoted with nil and the list constructor with ::. Here we use an additional defined predicate for list membership. It is defined in the standard way (writing the membership predicate in infix notation): $X \in (X :: L) \stackrel{\triangle}{=} \top$ and $X \in (Y :: L) \stackrel{\triangle}{=} X \in L$.

Proposition 7. *Let* $\mathtt{P}$ *be a process, let* $\mathtt{A}$ *be an object formula and let* $\bar{n}$ *be a list containing the free names of* $(\mathtt{P}, \mathtt{A})$*. Then* $\mathtt{P} \models \mathtt{A}$ *if and only the formula* $\nabla \bar{n}.[\![\mathtt{P}]\!] \models_{\bar{n}} [\![\mathtt{A}]\!]$ *is provable in* $FO\lambda^{\Delta\nabla}$ *with definition* $\mathcal{DA}'$*. Moreover, proof search in the Level 0/1 prover for the formula terminates.*

6 Related and future work

Perhaps the closest to our approach is Mads Dam's work on model checking mobile processes [3, 4]. However, our approach differs from his work in that the proof system we introduce is modular; different transition systems can be incorporated via definitions, while in his system, specifications of transition systems (π-calculus) are tightly integrated into the proof rules of the logic. Another difference is that we use the labelled transitions to encode the operational semantics which yields a simpler formalization (not having to deal with structural congruence) while Dam uses commitment relation with structural congruence. Another notable difference is that the use of relativised correctness assertions in his work which make explicit various conditions on names. In our approach, the conditions on names are partly taken care of implicitly by the meta logic (e.g., scoping, α-conversion, "newness"). However, Dam's logic is certainly more expressive in the sense that it can handle modal μ-calculus as well, via some global discharge conditions in proofs. We plan to investigate how to extend $FO\lambda^{\Delta\nabla}$ with such global discharge conditions.

History dependent automata (see, e.g., [6]) is a rather general model theoretic approach to model checking mobile processes. Its basis in automata models makes it closer to existing efficient implementation of model checkers. Our approach is certainly different from a conceptual view, so the sensible comparison would be in terms of performance comparison. However, at the current stage of our implementation, meaningful comparison cannot yet be made. A point to note, however, is that in the approach using history dependent automata, the whole state space of a process is constructed before checking the satisfiability of an assertion. In our approach, states of processes are constructed only when needed, that is, it is guided by the syntax of the process and the assertion it is being checked against.

Model checkers for π-calculus have also been implemented in XSB tabled logic programming [27]. The logic programming language used is a first-order one, and consequently, they have to encode bindings, α-conversion, etc. using first-order syntax. Such encodings make it hard to reason about the correctness of their specification. Compared to this work, our approach here is more declarative and meta theoretic analysis on the specification of the model checkers is available. Model checking for a richer logic than the modal logics we consider has been done in [2]. In this work, the issue concerning fresh names generation is dealt with using the permutation techniques of Gabbay-Pitts [7]. As in Dam's work, names here are dealt with explicitly via some algorithms for computing fresh names, while in our approach, the notion of freshness of names is captured implicitly by their scoping. More in-depth comparison is left for future work.

We plan to improve our current implementation to use the tabling methods in logic programming. Its use in implementing model checkers has been demonstrated in XSB [27] and also in [20]. Implementation of tabled deduction for higher-order logic programming has also been studied in [19], which can potentially be used in the implementation of $FO\lambda^{\Delta\nabla}$. We also plan to study other process calculi and their related notions of equivalences and modal logics, in particular the spi-calculus [1] and its related notions of bisimulation.

Acknowledgment. The author would like to thank the anonynimous referees for useful comments and suggestions. This work is based partly on a joint work with Dale Miller (INRIA Futurs/École polytechnique).

References

1. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 99.
2. L. Caries. Behavioral and spatial observations in a logic for the pi-calculus. In I. Walukiewicz, editor, *Proc. of FoSSaCs 2004*, 2004.
3. M. Dam. Model checking mobile processes. *Inf. Comput.*, 129(1):35–51, 1996.

4. M. Dam. Proof systems for pi-calculus logics. *Logic for concurrency and synchronisation*, pages 145–212, 2003.
5. L.-H. Eriksson. A finitary version of the calculus of partial inductive definitions. Vol. 596 of *LNAI*, pages 89–134. Springer-Verlag, 1991.
6. G.-L. Ferrari, S. Gnesi, U. Montanari, and M. Pistore. A model-checking verification environment for mobile processes. *ACM Trans. Softw. Eng. Methodol.*, 12(4):440–473, 2003.
7. M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2001.
8. J.-Y. Girard. A fixpoint theorem in linear logic. Email to the linear@cs.stanford.edu mailing list, February 1992.
9. G. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.
10. R. McDowell and D. Miller. Cut-elimination for a logic with definitions and induction. *Theoretical Computer Science*, 232:91–119, 2000.
11. R. McDowell, D. Miller, and C. Palamidessi. Encoding transition systems in sequent calculus. *Theoretical Computer Science*, 294(3):411–437, 2003.
12. D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. Vol. 475 of *LNAI*, pp. 253–281. Springer, 1991.
13. D. Miller. Unification under a mixed prefix. *J. of Symboluc Computation*, 14(4):321–358, 1992.
14. D. Miller and C. Palamidessi. Foundational aspects of syntax. *ACM Comp. Surveys Symp. on Theoretical Computer Science: A Perspective*, vol. 31. ACM, 1999.
15. D. Miller and A. Tiu. A proof theory for generic judgments: An extended abstract. In *Proc. of LICS 2003*, pages 118–127. IEEE, June 2003.
16. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Part II. *Information and Computation*, pages 41–77, 1992.
17. R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114(1):149–171, 1993.
18. T. Nipkow. Functional unification of higher-order patterns. In M. Vardi, editor, *Proc. of LICS'93*, pages 64–74. IEEE, June 1993.
19. B. Pientka. *Tabled Higher-Order Logic Programming*. PhD thesis, Carnegie Mellon University, December 2003.
20. Y. S. Ramakrishna, C. R. Ramakrishnan, I. V. Ramakrishnan, S. A. Smolka, T. Swift, and D. S. Warren. Efficient model checking using tabled resolution. In *Proc. of CAV97*, vol. 1254 of LNCS, pages 143–154, 1997.
21. D. Sangiorgi and D. Walker. *π-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
22. P. Schroeder-Heister. Rules of definitional reflection. In M. Vardi, editor, *Proc. of LICS'93*, pages 222–232. IEEE, June 1993.
23. R. F. Stärk. Cut-property and negation as failure. *International Journal of Foundations of Computer Science*, 5(2):129–164, 1994.
24. A. Tiu. *Level 0/1 Prover: A tutorial*, September 2004. Available online.
25. A. Tiu. *A Logical Framework for Reasoning about Logical Specifications*. PhD thesis, Pennsylvania State University, May 2004.
26. A. Tiu and D. Miller. A proof search specification of the π-calculus. In *3rd Workshop on the Foundations of Global Ubiquitous Computing*, Sept. 2004.
27. P. Yang, C. Ramakrishnan, and S. Smolka. A logical encoding of the π-calculus: model checking mobile processes using tabled resolution. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(1):38–66, July 2004.

Appendix

A. Some meta theory of $FO\lambda^{\Delta\nabla}$

$$\frac{}{\Sigma\,;\,\sigma \triangleright B, \Gamma \longmapsto \sigma \triangleright B}\ init \qquad \frac{\Sigma\,;\,\Delta \longmapsto \mathcal{B} \qquad \Sigma\,;\,\mathcal{B}, \Gamma \longmapsto \mathcal{C}}{\Sigma\,;\,\Delta, \Gamma \longmapsto \mathcal{C}}\ cut$$

$$\frac{\Sigma\,;\,\sigma \triangleright B, \sigma \triangleright C, \Gamma \longmapsto \mathcal{D}}{\Sigma\,;\,\sigma \triangleright B \wedge C, \Gamma \longmapsto \mathcal{D}}\ \wedge\mathcal{L} \qquad \frac{\Sigma\,;\,\Gamma \longmapsto \sigma \triangleright B \qquad \Sigma\,;\,\Gamma \longmapsto \sigma \triangleright C}{\Sigma\,;\,\Gamma \longmapsto \sigma \triangleright B \wedge C}\ \wedge\mathcal{R}$$

$$\frac{\Sigma\,;\,\sigma \triangleright B, \Gamma \longmapsto \mathcal{D} \qquad \Sigma\,;\,\sigma \triangleright C, \Gamma \longmapsto \mathcal{D}}{\Sigma\,;\,\sigma \triangleright B \vee C, \Gamma \longmapsto \mathcal{D}}\ \vee\mathcal{L} \qquad \frac{\Sigma\,;\,\Gamma \longmapsto \sigma \triangleright B}{\Sigma\,;\,\Gamma \longmapsto \sigma \triangleright B \vee C}\ \vee\mathcal{R}$$

$$\frac{}{\Sigma\,;\,\sigma \triangleright \bot, \Gamma \longmapsto \mathcal{B}}\ \bot\mathcal{L} \qquad \frac{\Sigma\,;\,\Gamma \longmapsto \sigma \triangleright C}{\Sigma\,;\,\Gamma \longmapsto \sigma \triangleright B \vee C}\ \vee\mathcal{R}$$

$$\frac{\Sigma\,;\,\Gamma \longmapsto \sigma \triangleright B \qquad \Sigma\,;\,\sigma \triangleright C, \Gamma \longmapsto \mathcal{D}}{\Sigma\,;\,\sigma \triangleright B \supset C, \Gamma \longmapsto \mathcal{D}}\ \supset\mathcal{L} \qquad \frac{\Sigma\,;\,\sigma \triangleright B, \Gamma \longmapsto \sigma \triangleright C}{\Sigma\,;\,\Gamma \longmapsto \sigma \triangleright B \supset C}\ \supset\mathcal{R}$$

$$\frac{\Sigma, \sigma \vdash t : \gamma \qquad \Sigma\,;\,\sigma \triangleright B[t/x], \Gamma \longmapsto \mathcal{C}}{\Sigma\,;\,\sigma \triangleright \forall_\gamma x.B, \Gamma \longmapsto \mathcal{C}}\ \forall\mathcal{L} \qquad \frac{\Sigma, h\,;\,\Gamma \longmapsto \sigma \triangleright B[(h\ \sigma)/x]}{\Sigma\,;\,\Gamma \longmapsto \sigma \triangleright \forall x.B}\ \forall\mathcal{R}$$

$$\frac{\Sigma, h\,;\,\sigma \triangleright B[(h\ \sigma)/x], \Gamma \longmapsto \mathcal{C}}{\Sigma\,;\,\sigma \triangleright \exists x.B, \Gamma \longmapsto \mathcal{C}}\ \exists\mathcal{L} \qquad \frac{\Sigma, \sigma \vdash t : \gamma \qquad \Sigma\,;\,\Gamma \longmapsto \sigma \triangleright B[t/x]}{\Sigma\,;\,\Gamma \longmapsto \sigma \triangleright \exists_\gamma x.B}\ \exists\mathcal{R}$$

$$\frac{\Sigma\,;\,(\sigma, y) \triangleright B[y/x], \Gamma \longmapsto \mathcal{C}}{\Sigma\,;\,\sigma \triangleright \nabla x\ B, \Gamma \longmapsto \mathcal{C}}\ \nabla\mathcal{L} \qquad \frac{\Sigma\,;\,\Gamma \longmapsto (\sigma, y) \triangleright B[y/x]}{\Sigma\,;\,\Gamma \longmapsto \sigma \triangleright \nabla x\ B}\ \nabla\mathcal{R}$$

$$\frac{\Sigma\,;\,\mathcal{B}, \mathcal{B}, \Gamma \longmapsto \mathcal{C}}{\Sigma\,;\,\mathcal{B}, \Gamma \longmapsto \mathcal{C}}\ c\mathcal{L} \qquad \frac{\Sigma\,;\,\Gamma \longmapsto \mathcal{C}}{\Sigma\,;\,\mathcal{B}, \Gamma \longmapsto \mathcal{C}}\ w\mathcal{L} \qquad \frac{}{\Sigma\,;\,\Gamma \longmapsto \sigma \triangleright \top}\ \top\mathcal{R}$$

Fig. 5. The core rules of $FO\lambda^{\Delta\nabla}$.

We state some meta theory of $FO\lambda^{\Delta\nabla}$. Proofs for the following propositions can be found in [25].

Proposition 8. *If* $\vdash B$ *and* x *is not free in* B*, then* $\vdash \nabla x B$.

Proposition 9. *If* $\vdash \forall x B$ *then* $\vdash \nabla x B$.

Proposition 10. *The formula* $\nabla x \nabla y\ B$ *is provable if and only if* $\nabla y \nabla x\ B$ *is provable.*

Proposition 11. *Let* Π *be a proof of* $\Sigma\,;\,\Gamma \longmapsto \mathcal{C}$*. Then for any substitution, there exists a proof* Π' *of* $\Sigma\theta\,;\,\Gamma\theta \longmapsto \mathcal{C}\theta$ *such that the length of proof of* Π' *is less or equal to the length of proof of* Π*.*

B. Properties of one-step transitions

To prove the adequacy results for modal logics, we shall consider some derived rules which allow us to enumerate all possible next states from a given process.

The rule one_f enumerates all possible free-actions that a process can perform:

$$\frac{\{\Sigma\,;\,\Gamma\theta \vdash \mathcal{C}\theta \mid\ (\bar{n} \triangleright P \xrightarrow{A} Q)\theta \text{ is provable}\}}{\Sigma, \Sigma'\,;\,\bar{n} \triangleright P \xrightarrow{A} Q, \Gamma \vdash \mathcal{C}}\ one_f$$

where P is a process term whose free variables, if any, are among $\bar{n}$, Σ' contains only eigenvariables appearing in A and Q and θ is a *closed substitution* (i.e., it maps variables to closed terms) such that $\Sigma' \subseteq \mathrm{dom}(\theta)$ and $\Sigma \cap \mathrm{dom}(\theta) = \emptyset$. The corresponding rule for bound input or bound output transition is defined analogously, i.e.,

$$\frac{\{\Sigma\,;\,\Gamma\theta \vdash \mathcal{C}\theta \mid\ (\bar{n} \triangleright P \xrightharpoonup{X} M)\theta \text{ is provable}\}}{\Sigma, \Sigma'\,;\,\bar{n} \triangleright P \xrightharpoonup{X} M, \Gamma \vdash \mathcal{C}}\ one_b$$

with analogous side conditions as in one_f. We show that these two rules are sound, that is, we can go from the premises to the conclusion of the rules using only rules in $FO\lambda^{\Delta\nabla}$.

Lemma 12. *The rules* one_f *and* one_b *are derivable in* $FO\lambda^{\Delta\nabla}$ *with the definition* $\mathbf{D}_\pi$*.*

Proof. For each rule, we give a bottom-up construction of a derivation from the root sequent. We then show that the resulting derivation tree covers all the premises specified in the rule. The two rules are derived using the same strategy: repeatedly apply any applicable left logical rules (no contraction and weakening is restricted to $\top$) to the transition judgment and the subsequent judgments that result from the application of the rules. We show here how to derive one_f, the other case is similar. By inspecting the definition of one-step transitions, we see that the applicable rules are among $def\mathcal{L}$, $\vee\mathcal{L}$, $\wedge\mathcal{L}$, $\exists\mathcal{L}$ and $\nabla\mathcal{L}$. Notice that these are all invertible rules, and therefore the particular order in which the rules are applied (if there is more than one applicable rules) are unimportant. However, to simplify the proof we shall fix the application order by considering the order of appearence of judgments in the sequent from left to right (i.e., by considering sequent context as a list instead of multiset). This strategy can be easily shown to be terminating, since the operational semantics of the transition always decomposes the principal process (i.e., the process to the left of the transition arrow), and since we start with a ground principal process, any application of $def\mathcal{L}$ always results in transition judgments with smaller principal (ground) processes.

Note that this search strategy generates a derivation tree where each branching results from an application of $def\mathcal{L}$, and a substitution is applied to each premise of the rule. The leaves of the derivation tree can be either *open* or *closed*. A closed leaf results from the failure of unification in $def\mathcal{L}$, while an open leaf results from a success in unification and no further applicable left-rules.

Therefore the open leaves are sequents of the form $\Sigma\,;\,\Gamma\theta \vdash \mathcal{C}\theta$ where θ is the substitution:

$$\theta_1 \circ \theta_2 \circ \cdots \circ \theta_k$$

where each θ_i is a substitution occuring in the premise of the i-th application of *def*$\mathcal{L}$ on the path from the root to the leaf (reading the derivation bottom-up).

It remains to show the following properties of this derivation tree:

1. each substitution θ in an open leaf is indeed a closed substitution,
2. $(\bar{n} \rhd P \xrightarrow{A} Q)\theta$ is provable for each substitution θ in the open leaves,
3. and for any closed substitution θ such that $(\bar{n} \rhd P \xrightarrow{A} Q)\theta$ is provable, the sequent $\Sigma\,;\,\Gamma\theta \vdash \mathcal{C}\theta$ is among the open leaves.

To show (1), observe that an open node must end (modulo the weakening rule on $\top$) with *def*$\mathcal{L}$ with one of the (raised) transition rules

$$\begin{aligned} \bar{m} \rhd \tau\ (P'\bar{m}) &\xrightarrow{\tau} (P'\bar{m}) \stackrel{\triangle}{=} \bar{m} \rhd \top, \\ \bar{m} \rhd in\ (X'\bar{m})\ (M'\bar{m}) &\xrightarrow{\downarrow(X'\bar{m})} (M'\bar{m}) \stackrel{\triangle}{=} \bar{m} \rhd \top, \\ \bar{m} \rhd out\ (Y\,\bar{m})\ (Z\,\bar{m})\ (P'\bar{m}) &\xrightarrow{\uparrow(Y\,\bar{m})(Z\,\bar{m})} (P'\,\bar{m}) \stackrel{\triangle}{=} \bar{m} \rhd \top. \end{aligned}$$

In all three cases, we unify the principal process with a ground process term, and hence in effect all variables are instantiated to closed terms.

Property (2) is proved by induction on the length of the path from the root sequent ending in an open leaf $\Sigma\,;\,\Gamma\theta \vdash \mathcal{C}\theta$. Note that since Γ and $\mathcal{C}$ play no part in the following construction, we shall omit them when writing down the sequents. We can therefore write down the path as a sequence of lists, starting with the root sequent and ending with the leaf sequent. We shall denote the first i-th composition of substitutions, i.e., $\theta_1 \circ \cdots \circ \theta_i$, with $\bar{\theta}_i$. So in particular, if there are k-applications of *def*$\mathcal{L}$, then θ can be written as $\bar{\theta}_k$. For instance, the following sequence denotes a path in the derivation tree

$$\{\bar{n} \rhd P \xrightarrow{A} Q\}, \cdots, \{\mathcal{A}\bar{\theta}_i, \Delta\bar{\theta}_i\}, \{\mathcal{B}\bar{\theta}_{i+1}, \Delta\bar{\theta}_{i+1}\}, \cdots, \{\}$$

Here it is shown the $(i+1)$-st application of *def*$\mathcal{L}$, where $\mathcal{A}\bar{\theta}_i$ is unified against some definition clause $\mathcal{H} \stackrel{\triangle}{=} \mathcal{B}$. That is, $\mathcal{A}\bar{\theta}_i \circ \theta_{i+1} = \mathcal{H}\theta_{i+1}$. Notice that we write in the succeeding sequent $\mathcal{B}\bar{\theta}_{i+1}$ instead of $\mathcal{B}\theta_{i+1}$ for uniformity, since we can anyway choose variables in $\mathcal{H}$ and $\mathcal{B}$ so that they are distinct from the variables in $\bar{\theta}_i$. The proof of $(\bar{n} \rhd P \xrightarrow{A} Q)\theta$ can now be constructed inductively as follows: for each list $\{\mathcal{A}_1\theta_i, \cdots, \mathcal{A}_l\theta_i\}$ construct a proof for each $\mathcal{A}_j\theta$, where $1 \leq j \leq l$. The non-trivial case is when *def*$\mathcal{L}$ is involved. That is, suppose the path starts with a *def*$\mathcal{L}$, i.e.,

$$\{\mathcal{A}\bar{\theta}_i, \Delta\bar{\theta}_i\}, \{\mathcal{B}\bar{\theta}_{i+1}, \Delta\bar{\theta}_{i+1}\}, \cdots, \{\}$$

given some definition clause $\mathcal{H} \stackrel{\triangle}{=} \mathcal{B}$ such that $\mathcal{A}\bar{\theta}_i \circ \theta_{i+1} = \mathcal{H}\theta_{i+1}$. By induction hypothesis we have a proof of $\mathcal{B}\theta$ and each of $\Delta\theta$. We need only to construct a proof of $\mathcal{A}\theta$. But this follows easily from the fact that $\mathcal{A}\bar{\theta_i} \circ \theta_{i+1} = \mathcal{H}\bar{\theta_i} \circ \theta_{i+1}$, and hence $\mathcal{A}\theta = \mathcal{H}\theta$, and therefore $def R$ can be applied to $\mathcal{A}\theta$, resulting in $\mathcal{B}\theta$.

Property (3) is the converse of (2). In this case we need to show that if $(\bar{n} \triangleright P \xrightarrow{A} Q)\theta$ is provable, then its proof corresponds to a path in the derivation tree of $\bar{n} \triangleright P \xrightarrow{A} Q \vdash$. We prove this by induction on the derivation tree. We need to show that, given any intermediate sequent (list) $\Delta \vdash \cdot$ in the derivation tree, if there is a closed substitution θ such that each of $\Delta\theta$ is provable, then $\Gamma\theta \vdash \mathcal{C}\theta$ is among the leaves of the derivation tree. We look again at the case involving $def\mathcal{L}$. Suppose $def\mathcal{L}$ is applied to some sequent $\{\mathcal{A}, \Delta\}$, and suppose that $\mathcal{A}\theta$ and each of $\Delta\theta$ are provable, for some closed substitution θ. In this case, the proof for $\mathcal{A}\theta$ would have to end with $def R$ with premise $\mathcal{B}\theta'$, that is, $\mathcal{A}\theta = \mathcal{H}\theta'$, for some definition clause $\mathcal{H} \stackrel{\triangle}{=} \mathcal{B}$. Again, we can assume that the domain of θ and θ' are disjoint, and that θ' is a closed substitution (since $\mathcal{A}\theta$ is a closed term) and so $\theta \circ \theta'$ is a (closed) unifier of $\mathcal{A}$ and $\mathcal{H}$. Therefore one of the premises of $def\mathcal{L}$ would be $\{\mathcal{B}(\theta \circ \theta'), \Delta(\theta \circ \theta')\}$. But we have $\mathcal{B}(\theta \circ \theta') = \mathcal{B}\theta'$ and $\Delta(\theta \circ \theta') = \Delta\theta$, and hence all judgments in this premise are provable, and by induction hypothesis, $\Gamma(\theta \circ \theta') \vdash \mathcal{C}(\theta \circ \theta')$ is among the leaves of the derivation tree. But $\Gamma(\theta \circ \theta') = \Gamma\theta$ and $\mathcal{C}(\theta \circ \theta') = \mathcal{C}\theta$, and therefore $\Gamma\theta \vdash \mathcal{C}\theta$ is among the leaves. □

The following proposition is a simple consequence of the above lemma.

Proposition 13. *Let* $\mathtt{P}$ *and* $\mathtt{Q}$ *be processes and* α *an action. Let* $\bar{n}$ *be a list of free names containing the free names in* $\mathtt{P}$*,* $\mathtt{Q}$*, and* α*. The transition* $\mathtt{P} \xrightarrow{\alpha} \mathtt{Q}$ *is not derivable in* π*-calculus if and only if* $.\,;\,. \vdash \neg\nabla\bar{n}.[\![\mathtt{P} \xrightarrow{\alpha} \mathtt{Q}]\!]$ *in* $FO\lambda^{\Delta\nabla}$ *with the definition* $\mathbf{D}_\pi$*.*

Proof. This is a simple corollary of Lemma 12. Consider for instance the case for free-action transitions (the other case with bound input or output is analogous). The application of one_f (after applying the necessary introduction rules for logical connectives) must produce empty premise:

$$\frac{}{.\,;\,\bar{n} \triangleright [\![\mathtt{P} \xrightarrow{\alpha} \mathtt{Q}]\!] \vdash \bot}\ one_f$$

since otherwise $\bar{n} \triangleright [\![\mathtt{P} \xrightarrow{\alpha} \mathtt{Q}]\!]$ would be provable, and this is impossible by the adequacy result (Proposition 4). □

C. Adequacy of the specification of modal logics

As noted earlier in this paper that since we encode free names in processes and assertions as ∇-quantified variables, we practically work with satisfiability judgments modulo renaming of free names. Therefore in the following proof of adequacy, when we state a modal judgment $\mathtt{P} \models \mathtt{A}$, we assume implicitly that it represents an equivalent classes of judgments modulo renaming of free variables in $(\mathtt{P}, \mathtt{A})$.

Proof of Proposition 6. *Soundness:* if $. ; \mathcal{X} \vdash \nabla\bar{n}.[\![\texttt{P} \models \texttt{A}]\!]$ then $\texttt{P} \models \texttt{A}$. This is proved by induction on the length of some "normal proof" of the sequent. We define a specific notion of normal proof just for provability of modal judgments. We first introduce the following derived rule:

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

This rule is derived from the $\forall\mathcal{L}$, $\vee\mathcal{L}$ and $def\mathcal{L}$ (on the syntactic equality $=$) rules. It can be shown using standard permutability argument that if the above modal judgment is provable, then it is provable using only the right-rules, the one_f-rule, the one_b-rule, the $def\mathcal{L}$-rule on syntactic equality and the ex-rule. Moreover, the invertible left-rules take precedence over any other rules. This will be our notion normal proof.

We now proceed to proving the main statement. We shall work with sequents in which the ∇ quantifier is already introduced, that is, sequents of the form $. ; \mathcal{X} \vdash \bar{n} \triangleright [\![\texttt{P} \models \texttt{A}]\!]$. Note that in sequents with no eigenvariables, e.g.,

$$\cdot\,;\,\bar{m} \triangleright \forall x \forall y. x = y \vee x \neq y, \Gamma \vdash \mathcal{C},$$

the application of the ex-rule on the excluded middle axiom becomes trivial, since in this case, x and y in $\bar{m} \triangleright \forall x \forall y. x = y \vee x \neq y$ can only be instantiated with names from the list $\bar{m}$. If x and y are instantiated to the same name, then the only premise of the ex rule would be $\cdot\,;\,\Gamma \vdash \mathcal{C}$. If x and y are instantiated to different names, say, a and b, then the only premise is $\cdot\,;\,\bar{m} \triangleright a \neq b, \Gamma \vdash \mathcal{C}$. It can then be easily shown that the hypothesis $\bar{m} \triangleright a \neq b$ can be removed from the proof of this sequent without changing the length of the proof. Therefore, any application of the ex-rule on a sequent with no eigenvariables can be removed. We shall see that we need not deal with non-trivial instances of ex-rule. The idea is to trivialize the non-trivial ex-rule (by instantiating all eigenvariables in the sequent to closed terms) before applying the induction hypothesis. This step is valid since substitution does not increase the length of proofs.

Having dealt with the excluded middle axioms, we now look at the more interesting cases involving logical rules. We detail a few cases here; all other cases can be proved in a similar way. In the following we denote with Π a normal proof of $. ; \mathcal{X} \vdash \nabla\bar{n}.[\![\texttt{P} \models \texttt{A}]\!]$.

match: Suppose $\texttt{A}$ is $[x = y]\texttt{B}$. We need to show that if x and y are the same name, then $\texttt{P} \models \texttt{B}$. Suppose x and y denote the same name a. Then Π must end with the following sequence of rules (recall that trivial instances of ex-rule can be removed):

$$\cfrac{\cfrac{\cfrac{\cfrac{\Pi'}{\cdot\,;\,\mathcal{X} \vdash \bar{n} \triangleright [\![\texttt{P} \models \texttt{B}]\!]}}{\cdot\,;\,\mathcal{X}, \bar{n} \triangleright a = a \vdash \bar{n} \triangleright [\![\texttt{P} \models \texttt{B}]\!]}\ def\mathcal{L}}{\cdot\,;\,\mathcal{X} \vdash \bar{n} \triangleright a = a \supset [\![\texttt{P} \models \texttt{B}]\!]}\ \supset\mathcal{R}}{\cdot\,;\,\mathcal{X} \vdash \bar{n} \triangleright [\![\texttt{P} \models [a = a]\texttt{B}]\!]}\ defR$$

By induction hypothesis on Π', we have that $\texttt{P} \models \texttt{B}$.

out: Suppose $\mathtt{A}$ is $[\bar{x}(y)]\mathtt{B}$. We need to show that for every $\mathtt{P}'$ such that $\mathtt{P} \xrightarrow{\bar{x}(y)} \mathtt{P}'$, we have $\mathtt{P}' \models \mathtt{B}$ (since by α-conversion we can assume without loss of generality that y is not free in $\mathtt{P}$ and $\mathtt{A}$). Note that here the occurrence of y in $\mathtt{P}'$ is bound in the transition judgment $\mathtt{P} \xrightarrow{\bar{x}(y)} \mathtt{P}'$. In this case, Π must be of the following shape:

$$\dfrac{\dfrac{\dfrac{\dfrac{\left\{ \dfrac{\begin{array}{c}\Pi_\theta \\ \cdot ; \mathcal{X} \vdash (\bar{n}y \triangleright M\bar{n}y \models [\![\mathtt{B}]\!])\theta\end{array}}{\cdot ; \mathcal{X} \vdash (\bar{n} \triangleright \nabla y.M\bar{n}y \models [\![\mathtt{B}]\!])\theta}\ \nabla\mathcal{R} \right\}_\theta}{M ; \mathcal{X}, \bar{n} \triangleright [\![\mathtt{P}]\!] \xrightarrow{\uparrow x} M\bar{n} \vdash \bar{n} \triangleright \nabla y.M\bar{n}y \models [\![\mathtt{B}]\!]}\ one_b}{M ; \mathcal{X} \vdash \bar{n} \triangleright [\![\mathtt{P}]\!] \xrightarrow{\uparrow x} M\bar{n} \supset \nabla y.M\bar{n}y \models [\![\mathtt{B}]\!]}\ \supset\mathcal{R}}{\cdot ; \mathcal{X} \vdash \bar{n} \triangleright \forall Q.[\![\mathtt{P}]\!] \xrightarrow{\uparrow x} Q \supset \nabla y.Qy \models [\![\mathtt{B}]\!]}\ \forall\mathcal{R}}{\cdot ; \mathcal{X} \vdash \bar{n} \triangleright [\![\mathtt{P} \models [\bar{x}(y)]\mathtt{B}]\!]}\ def R$$

where for each θ, we have that $(\bar{n} \triangleright [\![\mathtt{P}]\!] \xrightarrow{\uparrow x} M\bar{n})\theta$ is provable. Each successor $\mathtt{P}'$ of the process $\mathtt{P}$ under the bound output transition can be characterized by a substitution θ (e.g., take for example the substitution $\{(\lambda\bar{n}\lambda y.[\![\mathtt{P}']\!])/M\}$), such that (by adequacy of transitions in Proposition 4) $(\bar{n} \triangleright [\![\mathtt{P}]\!] \xrightarrow{\uparrow x} M\bar{n})\theta$ is provable. Hence by induction hypothesis on Π_θ, we have that $\mathtt{P}' \models \mathtt{B}$.

in: Suppose $\mathtt{A}$ is $[x(y)]^L\mathtt{B}$. We show that there exists a process $\mathtt{P}'$ such that $\mathtt{P} \xrightarrow{x(y)} \mathtt{P}'$ and for all name w, $\mathtt{P}'[w/y] \models \mathtt{B}[w/y]$. It is enough to consider the case where w is a name in fn$(\mathtt{P}, \mathtt{A})$ and the case where w is a new name not in fn$(\mathtt{P}, \mathtt{A})$. The proof Π in this case is of the form:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\Pi_1 \\ \cdot ; \mathcal{X} \vdash \bar{n} \triangleright \mathtt{P} \xrightarrow{\downarrow x} N\end{array} \quad \begin{array}{c}\Pi_2 \\ \cdot ; \mathcal{X} \vdash \bar{n} \triangleright \forall y.Ny \models [\![\mathtt{B}]\!]\end{array}}{\cdot ; \mathcal{X} \vdash \bar{n} \triangleright \mathtt{P} \xrightarrow{\downarrow x} N \wedge \forall y.Ny \models [\![\mathtt{B}]\!]}\ \wedge\mathcal{R}}{\cdot ; \mathcal{X} \vdash \bar{n} \triangleright \exists M.\mathtt{P} \xrightarrow{\downarrow x} M \wedge \forall y.My \models [\![\mathtt{B}]\!]}\ \exists\mathcal{R}}{\cdot ; \mathcal{X} \vdash \bar{n} \triangleright [\![[x(y)]^L\mathtt{B}]\!]}\ def R$$

where N is some closed term. By the adequacy result in Proposition 4, there exists a process $\mathtt{P}'$ such that $[\![\mathtt{P}']\!] = Ny$. By Proposition 11, we can instantiate y with any of the free names occurring in $\mathtt{P}$ or $\mathtt{A}$ (since they are all in the list $\bar{n}$), and hence for any name $w \in$ fn$(\mathtt{P}, \mathtt{A})$ by induction hypothesis we get $\mathtt{P}'[w/y] \models \mathtt{B}[w/z]$. By Proposition 9, there is a proof Π_2' of $\mathcal{X} \vdash \nabla y.Ny \models [\![\mathtt{B}]\!]$ with length less than or equal to that of Π_2, and hence by induction hypothesis we also cover the case where w is a new name.

□

To show completeness of the encoding of modal logics, we need the following lemma. We use the following abbreviation: given a list of names $\bar{x} = x_1, \ldots, x_n$

and a term t, we write $t \neq \bar{x}$ to denote the conjunction $t \neq x_1 \wedge \cdots \wedge t \neq x_n$ or $\top$ if $n = 0$.

Lemma 14. *If $\mathcal{X} \supset \nabla\bar{n}\nabla x.[\![\mathtt{P} \models \mathtt{A}]\!]$ is provable, where $\mathcal{X} \subseteq_f \mathcal{E}$, then the formula $\mathcal{X}' \supset \nabla\bar{n}\forall x.x \neq \bar{n} \supset [\![\mathtt{P} \models \mathtt{A}]\!]$ is provable.*

Proof. We prove a stronger statement: given a proof Π of the sequent

$$\Sigma\,;\,\mathcal{X}, \bar{n}x\bar{m}_1 \triangleright A_1, \ldots, \bar{n}x\bar{m}_k \triangleright A_k \vdash \bar{n}x\bar{m}_0 \triangleright A_0$$

there is a proof of the sequent

$$\Sigma, H\,;\,\mathcal{X}', \bar{n}\triangleright H\bar{n} \neq \bar{n}, \bar{n}\bar{m}_1\triangleright A_1[H\bar{n}/x], \ldots, \bar{n}x\bar{m}_k\triangleright A_k[H\bar{n}/x] \vdash \bar{n}x\bar{m}_0\triangleright A_0[H\bar{n}/x]$$

provided that H is an eigenvariable not appearing in Σ, and that for every eigenvariable F in Σ, if F has type $\tau_1 \to \cdots \to \tau_l \to n$ then each τ_j is the type n. The excluded middles $\mathcal{X}'$ are just those of $\mathcal{X}$ with one local-variable less.

The proof is by induction on the length of proof of Π. The proof Π' is constructed by imitating the rules used in Π. Most cases follow straightforwardly from induction hypothesis, except for the case involving $\mathit{def}\mathcal{L}$, since in this case unification may produce premises that cannot be derived from induction hypothesis. We need to show that those premises can be proved using the inequality assumption $\bar{n} \triangleright H\bar{n} \neq \bar{n}$. So suppose that the proof Π ends with a $\mathit{def}\mathcal{L}$ applied to the judgment $\bar{n}x\bar{m}_i \triangleright A_i$. We imitate this step in constructing Π' by applying $\mathit{def}\mathcal{L}$ to $\bar{n}\bar{m}_i \triangleright A_i[H\bar{n}/x]$. That is, we unify this judgment with the head of some definition clause $\bar{n}\bar{m}_i \triangleright A \stackrel{\triangle}{=} \bar{n}\bar{m}_i \triangleright B$. Suppose that θ is a unifier of the two terms, that is,

$$(\lambda\bar{n}\lambda\bar{m}_i.A_i[H\bar{n}/x])\theta = (\lambda\bar{n}\lambda\bar{m}_i.A)\theta.$$

There are three cases to consider:

1. $\theta(H) = \lambda\bar{n}.x$, where x is in $\bar{n}$,
2. $\theta(H) = H$,
3. $\theta(H) = \lambda\bar{n}.u$, for some non-variable term u.

In the first case, the resulting premise can be proved immediately using the inequality $(\bar{n} \triangleright H\bar{n} \neq x)\theta$. In the second case, a proof for the resulting premise can be constructed using induction hypothesis. In the third case, since we do not assume any constant of type $\cdots \to n$, u can only be a term of the form: $(H't_1 \cdots t_p)$ where H' is an eigenvariable. It is easy to see that the occurrence of this term inside any other term is unaffected by λ-conversion. Therefore, we can replace any occurrence of the subterm $(H't_1 \cdots t_p)$ in the range of θ with any term and the resulting substitution is still a unifier. Let us denote with $t^\bullet$ the normal form of the term t with every occurrence of the subterm $(H't_1 \cdots t_p)$ replaced with $H\bar{n}$. We construct the following substitution:

$$\theta'(X) = \begin{cases} t^\bullet, \text{ if } \theta(X) = t \text{ and } t \neq X, \\ X, \text{ otherwise.} \end{cases}$$

Then θ' is a unifier of $\lambda\bar{n}\lambda\bar{m}_i.A_i[H\bar{n}/x]$ and $\lambda\bar{n}\lambda\bar{m}_i.A$. Moreover, it is more general than θ, that is,

$$\theta' = \theta \circ [\lambda\bar{n}.u/H].$$

Therefore we apply the construction in the second case, and apply the substitution $[\lambda\bar{n}.u/H]$ to the resulting proof. □

Proof of Proposition 6. *Completeness:* if $\mathtt{P} \models \mathtt{A}$ then for some $\mathcal{X} \subseteq_f \mathcal{E}$ and some names $\bar{n} \supseteq \mathrm{fn}(\mathtt{P}, \mathtt{A})$, the sequent $.\,; \mathcal{X} \vdash \nabla\bar{n}.[\![\mathtt{P} \models \mathtt{A}]\!]$ is provable. This is proved by induction on the size of $\mathtt{A}$. We show the non-trivial case which involves universal quantification over names. That is, suppose $\mathtt{A} = \langle x(y)\rangle^L\mathtt{B}$ and $\mathtt{P} \models \mathtt{A}$. This means that there is a process $\mathtt{P}'$ such that $\mathtt{P} \xrightarrow{x(y)} \mathtt{P}'$ and for all name w, $\mathtt{P}'[w/y] \models \mathtt{B}[w/y]$. It is enough to consider the cases where $w \in \mathrm{fn}(\mathtt{P}, \mathtt{A})$ and the case where w is a new name. Thus from induction hypothesis (after applying some weakening and extensions on ∇ using Proposition 8), we have the following proofs:

$$\begin{array}{c} \Pi_1 \\ \cdot\,; \mathcal{X} \vdash \nabla\bar{n}.[\![\mathtt{P}[x_1/y] \models \mathtt{B}[x_1/y]]\!] \end{array} \cdots \begin{array}{c} \Pi_k \\ \cdot\,; \mathcal{X} \vdash \nabla\bar{n}.[\![\mathtt{P}[x_k/y] \models \mathtt{B}[x_k/y]]\!] \end{array}$$

where $\bar{n} = x_1, \ldots, x_k$, and

$$\begin{array}{c} \Pi_{k+1} \\ \cdot\,; \mathcal{X} \vdash \nabla\bar{n}\nabla y.[\![\mathtt{P} \models \mathtt{B}]\!]. \end{array}$$

From these proofs, we shall construct a proof of

$$\cdot\,; \mathcal{X} \vdash \nabla\bar{n}\forall y.[\![\mathtt{P} \models \mathtt{B}]\!].$$

We first apply the introduction rules for the quantifiers, thus arriving at the sequent:

$$H\,; \mathcal{X} \vdash \bar{n} \triangleright .[\![\mathtt{P}]\!][H\bar{n}/y] \models [\![\mathtt{B}]\!][H\bar{n}/y].$$

We do this by repeatedly applying the *ex*-rule on the excluded middle

$$\bar{n} \triangleright \forall x \forall w. x = w \vee x \neq w$$

which we assume (without loss of generality) is among the $\mathcal{X}$. That is, we instantiate x with $H\bar{n}$ and w with x_i. This will give us $k+1$ sequents to prove,

$$\cdot\,; \mathcal{X} \vdash \bar{n} \triangleright [\![\mathtt{P}[x_i/y] \models \mathtt{B}[x_i/y]]\!] \qquad (1)$$

for each $i \in \{1, \ldots, k\}$ and

$$H\,; \mathcal{X}, \bar{n} \triangleright H\bar{n} \neq \bar{n} \vdash \bar{n} \triangleright [\![\mathtt{P}]\!][H\bar{n}/y] \models [\![\mathtt{B}]\!][H\bar{n}/y]. \qquad (2)$$

The proof for sequent (1) can be constructed immediately from Π_i. The proof for sequent (2) is obtained by applying Lemma 14 to Π_{k+1}. □

D. Implementation

The complete concrete specification of modal logics for π-calculus is given in Figure 6.

(a) Propositional connectives and *basic* modality:

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

(b) *Late* modality:

$$
\begin{array}{lll}
P \models_L \langle \downarrow X \rangle^l A & \triangleq & \exists P'(P \xrightarrow{\downarrow X} P' \wedge \nabla z \forall y. y \in (z :: L) \supset P'y \models_{(z::L)} Ay). \\
P \models_L [\downarrow X]^l A & \triangleq & \forall P'(P \xrightarrow{\downarrow X} P' \supset \nabla z \exists y. y \in (z :: L) \wedge P'y \models_{(z::L)} Ay).
\end{array}
$$

(c) *Early* modality:

$$
\begin{array}{lll}
P \models_L \langle \downarrow X \rangle^e A & \triangleq & \nabla z \forall y. y \in (z :: L) \supset \exists P'(P \xrightarrow{\downarrow X} P' \wedge P'y \models_{(z::L)} Ay). \\
P \models_L [\downarrow X]^e A & \triangleq & \nabla z \exists y. y \in (z :: L) \wedge \forall P'(P \xrightarrow{\downarrow X} P' \supset P'y \models_{(z::L)} Ay).
\end{array}
$$

Fig. 6. A more concrete specification of modal logics for π-calculus