A Congruence Format for Name-passing Calculi

Axelle Ziegler, Dale Miller, Catuscia Palamidessi

A congruence format for name-passing calculi

Axelle Ziegler, Dale Miller, and Catuscia Palamidessi [1]

INRIA-Futurs and LIX, École Polytechnique, Palaiseau, France

Abstract

We define and use a SOS-based framework to specify the transition systems of calculi with name-passing properties. This setting uses proof-theoretic tools to take care of some of the difficulties specific to name-binding and make them easier to handle in proofs. The contribution of this paper is the presentation of a format that ensures that open bisimilarity is a congruence for calculi specified within this framework, extending the well-known tyft/tyxt format to the case of name-binding and name-passing. We apply this result to the π-calculus in both its late and early semantics.

Key words: Structural operational semantics, rule formats, name-binding, name-mobility, open bisimulation.

1 Introduction

Structured operational semantics (SOS) [Plo81] is well-suited for the specification of process calculi and allows for both a clear and convenient presentation of transition systems. Because it is desirable to reason about behavioral equivalences in a compositional fashion, various researchers have provided restrictions on the specification of operational semantics that guarantee that the derived notion of bisimilarity is a congruence. The first such restrictions, such as GSOS [BIM95], tyft/tyxt [GV92], and Panth [Ver95], were not well-suited to deal with processes involving *abstractions*. Examples of such process calculi are the higher-order process calculi, like CHOCS [Tho93], and the name-passing and name-binding process calculi, like the π-calculus [MPW92]. Recently, there have been proposals for extending rule formats to higher-order languages [How96] and in particular to higher-order process calculi [Ber98,MGR05]. However, [MGR05] considers neither name-passing nor name-binding features, while [Ber98] does handle some name-passing languages, but at the cost of a complete rewriting of the specification, which ends

[1] Email: ziegler [at] clipper.ens.fr, dale [at] lix.polytechnique.fr, and catuscia [at] lix.polytechnique.fr

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

up in representing the features of names in an indirect and low-level way. As far as we know, rule formats for the mechanisms related to names have not been investigated yet, at least not in a direct fashion.

The aim of this paper is to fill the above gap: we present a rule format for name-passing and name-binding process-calculi which guarantees that open bisimilarity is a congruence, and which handles name binding and passing in a direct fashion.

To obtain the result of congruence, we need to make precise the notion of congruence for name-passing and name-binding process-calculi. We propose a notion which lifts to our higher-order setting the standard first-order notion of congruence. Our definition is strongly inspired by similar definitions in the case of λ-calculus (for instance, logical relations [Sta85]). In the framework of process calculi, the only other proposal of this kind we know of is the notion of *agent congruence* for the π-calculus with abstraction and concretion [Mil99]. However, the spirit of our approach is different from the one of [Mil99]: we aim at preserving the meaning of the first-order notion of congruence, and, as a consequence, our definition is much stricter. For instance, strong bisimilarity is an agent congruence in [Mil99] but is not a congruence according to our definition.

The results presented here are based on the first author's master thesis [Zie04]. We refer to the full version of this paper [ZMP05] for details.

1.1 Plan of the paper

In Section 2 we describe the proof-theoretic tools that we use to encode transition system specifications and we illustrate briefly their use with the π-calculus. Sections 3 and 4 show how the usual notions of bisimulation and congruence can be lifted to our proof theoretic framework to accommodate name-binding. In Section 5, we use this framework to define a rule format that ensures that open bisimilarity is a congruence. We conclude in Section 6 with some future directions.

2 Transition System Specifications and Name Passing

2.1 Motivating a logical framework for binding

The main drawback of standard treatments of SOS when used to specify a name-passing and name-binding calculus, such as the π-calculus, is that the status of variables and names generally requires adding a number of side-conditions. Some might for instance require a name to be free or not in certain subexpressions, or a name to be "fresh" with respect to some context, etc. Those side conditions are generally not allowed in rule formats that guarantee congruence of bisimilarity. Moreover, the presence of such side conditions is generally acknowledged as inducing many complications into the theory of the specified process calculus.

A more declarative approach to specifying the semantics of processes with abstractions involves finding a logic that completely internalizes the complexities of bindings. If there is such a logic, then hopefully the side conditions are replaced by naturally occurring phenomenon within the logic. The specifiers of such a logic must, of course, deal with the complexities of bindings and substitutions.

In 1940, Church [Chu40] designed the Simple Theory of Types (STT) as a higher-order logic containing just one binder, the λ-binder, that could be used to capture all binders in formulas and in terms. The proof rules for STT were, indeed, complex in order to fully axiomatize how λ-binders interacted with logical inference rules. Decades later, experience with programming languages such as λProlog [MN87] (based on a subset of STT *without* extensionality) showed that bindings in operational semantic specifications could be treated declaratively in that logic.

The use of λ-terms within certain weak subsets of higher-order logic is referred to as the λ-*tree syntax approach* to encoding syntax [MP99]. The λ-tree syntax approach is a particular approach to *higher-order abstract syntax* (HOAS) [PE88]. In the latter, other approaches to the encoding of bindings are possible. For example, the HOAS encoding of the π-calculus in [Des00] makes use of a logic that is extensional and, hence, bindings in syntax are mapped to *functions* in higher-order logic. Most of the development of this paper requires that the logic interprets λ-bindings as abstract syntax and not functions.

While Church's logic provided strong methods for reasoning about λ-terms as functions, the viewing of λ-terms as syntax requires a more intensional approach. Recent papers of Miller and Tiu [MT03b,MT03a] provide such an approach by providing the logic $FO\lambda^{\Delta\nabla}$ (fold-nabla) that includes the new quantifier ∇ for defining *generic judgments*. It is possible in $FO\lambda^{\Delta\nabla}$ to specify labeled transitions, bisimulations, and modal logics for the π-calculus [TM04,Tiu05] completely declaratively and without side conditions.

2.2 Technical description of the system

The conventional approach to presenting SOS for labeled transition systems uses inference rules of the form

$$\frac{\left\{P_i \xrightarrow{A_i} Q_i \mid i \in I\right\}}{P \xrightarrow{A} Q}.$$

It is a trivial observation that such an inference rule can be seen as the Horn clause

$$\forall \bar{X}\left[\left(\bigwedge_{i\in I} P_i \xrightarrow{A_i} Q_i\right) \supset P \xrightarrow{A} Q\right]$$

in a first-order logic. Here, the variables in the list $\bar{X}$ are the variables free in the conclusion or in some premise of the rule. They have first-order type n, a, or p, which stand for *names*, *actions*, and *processes*, respectively. These variables are known as *meta-variables* or *schema variables* of the inference rule.

In order to use this conventional approach to treat name-binding constructs in processes and their transitions, side-conditions on the inference rules are often employed. These side conditions generally state that a name is "fresh" or not free in a given term. There are two choices to formalizing such side conditions: one can axiomatize them in first-order logic, or, following the motivation in Section 2.1, one moves to a logic that directly supports bindings.

We take this second approach here. In particular, we use the $FO\lambda^{\Delta\nabla}$ logic that allows for λ-bindings in terms and use the two arrow types $n \to a$ and $n \to p$ that are constructed by this λ-binding. Being derived from STT, syntactic expressions involving λ-bindings satisfy α, β, and η conversion. As we shall see, in order to adequately be able to reason about the structure of λ-abstractions, we shall need the ∇-quantifier, and a distinction over types of n.

To encode a transition system, we shall introduce a new type system: As stated before, we have one type p for the syntactic category of processes in our language, but we actually need two types, n and v, for the syntactic category of names. Here, n denotes the usual notion of names and contains an infinite number of constants, and v denotes new names. There are no constants assumed of type v and v is considered to be a subtype of n, which means that we can use a value of type v wherever a value of type n is required (a new name is a name). We also have both $n \to p$ and $v \to p$ abstractions. Further, we have a type a for actions, which comes with the three constructors $\downarrow\colon n \to n \to a$, $\uparrow\colon n \to n \to a$, and $\tau : a$, denoting input, output, and internal actions. For example, $\downarrow xy$ denotes the action of inputting name y on the channel with name x. If needed, the system could be extended to handle more kinds of actions.

Transitions themselves are encoded using the symbol $\cdot \stackrel{\cdot}{\longrightarrow} \cdot$. This symbol denotes three different predicates: $\cdot \underset{p}{\stackrel{\cdot}{\longrightarrow}} \cdot$ taking arguments of type p, a and p, $\cdot \underset{n\to p}{\stackrel{\cdot}{\longrightarrow}} \cdot$ taking arguments of type p, $n \to a$ and $n \to p$, and $\cdot \underset{v\to p}{\stackrel{\cdot}{\longrightarrow}} \cdot$ taking arguments of type p, $v \to a$ and $v \to p$. The first predicate allows us to encode free transitions, whereas the two others allows us to encode bound transitions, abstracted on names or on fresh names. The intuition behind this distinction will be further discussed in the rest of the paper.

With this symbol, we can specify transition systems in the usual way. The major difference with usual practice is that we will allow both quantification of schema variables of higher type, such as $n \to a$ and $n \to p$, as well as ∇-quantifications over names in the premises of rules. In particular, rules will

be of the following form:

$$\frac{\left\{\mathcal{Q}_i\left(P_i \xrightarrow[\gamma_i]{A_i} Q_i\right) \mid i \in I\right\}}{P \xrightarrow[\gamma]{A} Q}$$

where I is a finite subset of $\mathbb{N}$, and γ and γ_i may be any type amongst $\{p, n \to p, v \to p\}$. Furthermore, the schema variables P and P_i are of type p, the variables Q, Q_i are of type γ and γ_i respectively, and the variables A, A_i are of type a, $n \to a$ or $v \to a$, depending on γ. Finally, the $\mathcal{Q}_i$ denotes a possibly empty sequence of ∇-quantified variables of type v. See Figure 1 for examples of such inference rules.

2.3 A logic for operational semantics

We overview here the proof theory of the ∇-quantifier in order to help make its role in the specification of name-binding clear. More details can be found in [MT03b,MT03a]. The proof theory for the logic $FO\lambda^{\Delta\nabla}$ is derived from the standard sequent calculus for intuitionistic logic by adding the quantifier ∇. This quantifier is used to declare that a new object has scope within a certain part of a computation and nowhere else. Given our focus here on name-binding in process calculi, ∇-quantified variables will always be names, and they will be in the type v.

We now outline how ∇-quantification is accommodated within proofs. Our sequents extend traditional (single-conclusion) sequent calculus by the addition of both *global* and *local* signatures. In particular, in the sequent

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

the set Σ, called the *global signature*, collects together the free (eigen)variables of the entire sequent, while the lists σ_i are *local signatures* and these contain variables scoped only over B_i. We consider both kinds of signatures as binding structures within sequents.

To illustrate the novel features $FO\lambda^{\Delta\nabla}$, we present the inference rules for ∇ and $\forall$. The interaction between those two quantifiers is central to our work.

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

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

Notice the right and left introduction rules for ∇ are essentially the same. Thus, ∇ is self-dual: that is, the equivalence $\neg\nabla x.Bx \equiv \nabla x.\neg B$ will hold. In comparing the right introduction rules for ∇ and $\forall$, we notice that ∇-bound variables move to local signatures (reading proof rules bottom up)

and $\forall$-bound variables move to global signatures. The new eigen-variable h introduced by the $\forall\mathcal{R}$ rule does *raising*: that is, instead of instantiating the quantifier with a new variable, it instantiates the quantifier with the "raised" variable expression $(h\ \sigma)$, which denotes the term $(hx_1 \dots x_n)$ if σ is the list $(x_1 \dots x_n)$. Thus, when a $\forall$-bound variable appears in the scope of a local variable, the $\forall$-bound variable is instantiated with an abstraction over the local variable.

A brief comparison of ∇ with the "new"quantifier proposed by Gabbay and Pitts [**?**] might be useful here. The "new" quantifier is provided with a set-theoretic semantics that contains a denumerably infinite set of names. The "new" quantifier is then able to select a fresh variable name (in particular, a name not present in the quantified formula) from this existing set of names and the underlying set theory provides laws that make the particular choice of the fresh name immaterial. The ∇ quantifier is described in terms of proof theory and can be used to quantify over any type (not necessarily one for names) and its meta-theory does not assume that that type is inhabited. The ∇ quantifier is essentially hypothetical. It essentially asks the question: "If you are given a new element of this type, can you prove the body of the quantifier for it." A concrete difference between these quantifiers is that the ∇ quantifier is not entailed by nor entails the $\forall$ or $\exists$ quantifier, whereas the "new" quantifier is implied by the $\forall$ quantifier and implies the $\exists$ quantifier.

Another aspect of $FO\lambda^{\Delta\nabla}$ is that it allows for the treatment of fixed points: that is, predicates can be defined inductively and co-inductively. For example, operational semantics are specified by an inductive definition while bisimilarity is specified by a co-inductive one. The specifics of how this are treated in the sequent calculus are not central to our presentation of our rule format definition nor to the statement of our main results. The formal proof of the main results are, however, based on the details of such fixed pointed constructions. The interested reader can find more details in [MT03c,Tiu04].

2.4 Example: encoding of the pi-calculus

To encode the π-calculus syntax, we use the constructors

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

in the straightforward way: in particular, we encode $x(y).P$ as $\mathit{in}\ x\ (\lambda y.P)$, $\bar{x}y.P$ as $\mathit{out}\ x\ y\ P$, $\nu y.P$ as $\nu(\lambda y.P)$, and $[x = y].P$ as $\mathit{match}\ x\ y\ P$. Notice that π-calculus bindings are mapped to λ-bindings within terms. We shall usually write π-calculus expressions in their original form and only refer to this specific encoding when needed.

The rules for π-calculus are shown in Figure 1. On the surface, these inferences resemble the usual ones for the π-calculus. There are, however, several

$$\frac{}{\tau.P \xrightarrow[p]{\tau} P}\ (\tau) \qquad \frac{}{X(y).(My) \xrightarrow[n\to p]{\downarrow X} M}\ (\text{IN}) \qquad \frac{}{\bar{X}Y.P \xrightarrow[p]{\uparrow XY} P}\ (\text{OUT})$$

$$\frac{P \xrightarrow[\gamma]{A} R}{P + Q \xrightarrow[\gamma]{A} R}\ (\text{PLUS}) \qquad \frac{P \xrightarrow[\gamma]{A} Q}{[X = X].P \xrightarrow[\gamma]{A} Q}\ (\text{MATCH})$$

$$\frac{P \xrightarrow[p]{A} P'}{P \mid Q \xrightarrow[p]{A} P' \mid Q}\ (\text{PAR}) \qquad \frac{P \xrightarrow[\delta]{A} M}{P \mid Q \xrightarrow[\delta]{A} \lambda x(M\, x \mid Q)}\ (\text{PAR})$$

$$\frac{\nabla x(Px \xrightarrow[p]{A} Qx)}{\nu x.Px \xrightarrow[p]{A} \nu x.Qx}\ (\text{RES}) \qquad \frac{\nabla x(Px \xrightarrow[\delta]{A} P'x)}{\nu x.Px \xrightarrow[\delta]{A} \lambda y.\nu x.P'xy}\ (\text{RES})$$

$$\frac{\nabla y(Py \xrightarrow[p]{\uparrow Xy} Qy)}{\nu y.Py \xrightarrow[v\to p]{\uparrow X} Q}\ (\text{OPEN}) \qquad \frac{P \xrightarrow[n\to p]{\downarrow X} M \qquad Q \xrightarrow[v\to p]{\uparrow X} N}{P \mid Q \xrightarrow[p]{\tau} \nu y.(My \mid Ny)}\ (\text{CLOSE})$$

$$\frac{P \xrightarrow[n\to p]{\downarrow X} M \qquad Q \xrightarrow[p]{\uparrow XY} Q'}{P \mid Q \xrightarrow[p]{\tau} MY \mid Q'}\ (\text{COM}) \qquad \frac{P \xrightarrow[p]{\uparrow XY} P' \qquad Q \xrightarrow[n\to p]{\downarrow X} M}{P \mid Q \xrightarrow[p]{\tau} P' \mid MY}\ (\text{COM})$$

$$\frac{P \xrightarrow[p]{A} P'}{!P \xrightarrow[p]{A} P' \mid !P}\ (!\text{ACT}) \qquad \frac{P \xrightarrow[\delta]{A} P'}{!P \xrightarrow[\delta]{A} \lambda x(P'x \mid !P)}\ (!\text{ACT})$$

$$\frac{P \xrightarrow[p]{\uparrow XY} P' \qquad P \xrightarrow[n\to p]{\downarrow X} M}{!P \xrightarrow[p]{\tau} !P \mid (P' \mid MY)}\ (!\text{COM}) \qquad \frac{P \xrightarrow[v\to p]{\uparrow X} M \qquad P \xrightarrow[n\to p]{\downarrow X} N}{!P \xrightarrow[p]{\tau} \nu y.(\, !P \mid (Ny \mid My))}\ (!\text{CLOSE})$$

Fig. 1. Transition system for the late semantics of the π-calculus. The symmetric versions of (CLOSE) and both (PLUS) and (PAR) rules are also assumed. We use the convention that schema variables for a rule are written with a capitalized letter. The type variable γ ranges over the set $\{p, n \to p, v \to p\}$, while δ ranges over $\{n \to p, v \to p\}$.

subtle differences. One difference is that rules do not explicitly contain names but rather meta-variables of type v and n (these are implicitly universally bound and written with capital letters) or bound variables of type v or n (these are explicitly λ- or ∇-bound and are written with lowercase letters). The types of the variables can always be inferred from the context. In this paper, ∇-quantified variables will always be considered to be of type v.

A second difference is that these inference rules use higher-order variables of type $n \to a$, $n \to p$, $v \to a$, $v \to p$ and (in the second (RES) rule) $n \to n \to p$. Notice that the expression $X(y).(My)$ matches the π-calculus expression $n(w).\bar{w}m.0$ exactly when the meta-variables X and M are instantiated with

the terms n and $\lambda w.\bar{w}m.0$, respectively. These matching substitutions are unique up to α-conversion. Notice further that the expression $X(y).M$ fails to match this same π-calculus expression since there is no capture-avoiding substitution for M that will yield that expression.

A third difference is the use of the ∇-quantification: there are not explicit rules for introducing ∇-quantification in Figure 1. Instead, the logical inference rules outlined in Section 2.3 are responsible for dealing with this quantifier. The interplay between higher-order variables and ∇-quantification means that the common side conditions on names and their occurrences are not necessary. For example, the usual restriction for the (res) rule that states that x is not free in A is implicit in the quantifier scoping rules of that rule: the ∇-quantifier for x is in the scope of the $\forall$-quantification for A, thus no substitution instance of this inference rule will have an occurrence of x in the substitution of A. The logic guarantees, of course, that the application of a substitution avoids variable-capture.

It is straightforward to modify those rules to encode the early semantics of the π-calculus [Par01]: in particular, we need to add the rule

$$\frac{}{X(y)(My) \xrightarrow[p]{\downarrow XU} (M\ U)}\ (\text{IN-E})$$

and replace the two (com) rules with the rules

$$\frac{P \xrightarrow[p]{\uparrow XY} P' \qquad Q \xrightarrow[p]{\downarrow XY} Q'}{P \mid Q \xrightarrow[p]{\tau} P' \mid Q'}\ (\text{COM-E}) \qquad \frac{P \xrightarrow[p]{\downarrow XY} P' \qquad Q \xrightarrow[p]{\uparrow XY} Q'}{P \mid Q \xrightarrow[p]{\tau} P' \mid Q'}\ (\text{COM-E})$$

and change the (!com) and (open) rules to the rules

$$\frac{P \xrightarrow[p]{\uparrow XY} P' \qquad P \xrightarrow[p]{\downarrow XY} Q'}{!P \xrightarrow[p]{\tau} !P \mid P' \mid Q'}\ (!\text{COM-E}) \qquad \frac{\nabla y(Py \xrightarrow[p]{Ay} Qy)}{\nu y.Py \xrightarrow[\delta]{A} Q}\ (\text{OPEN})$$

3 Open bisimulation

The co-inductive predicate defined in Figure 2 defines our notion of open bisimulation in the sense that two processes P and Q are said to be open bisimilar when there is a proof (possibly involving the co-induction inference rule) of $\forall \bar{x}\ \mathit{bisim}\ P\ Q$, where $\bar{x}$ denotes the set of all free name variables in P and Q. In the case of the π-calculus, this definition exactly coincides with the usual definition of open-bisimulation [SW01,TM04]. In other calculi however, this relation could be named otherwise. What we will call open bisimulation in the rest of this paper is bisimulation up to substitutions of variables of type n and *distinctions* of variables of type v [SW01].

$$\begin{aligned}
&\mathit{bisim}\ P\ Q \stackrel{\nu}{=} \\
&\quad \forall A \forall P' [P \xrightarrow[p]{A} P' \supset \exists Q'.Q \xrightarrow[p]{A} Q' \wedge \mathit{bisim}\ P'\ Q'] \wedge \\
&\quad \forall A \forall Q' [Q \xrightarrow[p]{A} Q' \supset \exists P'.P \xrightarrow[p]{A} P' \wedge \mathit{bisim}\ Q'\ P'] \wedge \\
&\quad \forall A \forall P' [P \xrightarrow[n\to p]{A} P' \supset \exists Q'.Q \xrightarrow[n\to p]{A} Q' \wedge \forall w.\mathit{bisim}\ (P'w)\ (Q'w)] \wedge \\
&\quad \forall A \forall Q' [Q \xrightarrow[n\to p]{A} Q' \supset \exists P'.P \xrightarrow[n\to p]{A} P' \wedge \forall w.\mathit{bisim}\ (Q'w)\ (P'w)] \wedge \\
&\quad \forall A \forall P' [P \xrightarrow[v\to p]{A} P' \supset \exists Q'.Q \xrightarrow[v\to p]{A} Q' \wedge \nabla w.\mathit{bisim}\ (P'w)\ (Q'w)] \wedge \\
&\quad \forall A \forall Q' [Q \xrightarrow[v\to p]{A} Q' \supset \exists P'.P \xrightarrow[v\to p]{A} P' \wedge \nabla w.\mathit{bisim}\ (Q'w)\ (P'w)]
\end{aligned}$$

Fig. 2. Specification of the *bisim* predicate. The $\stackrel{\nu}{=}$ symbol is used to declare that this definition can be used with a co-inductive inference rule.

Note that if we considered proving the formula $\nabla \bar{x}\ \mathit{bisim}\ P\ Q$ instead, we would be treating the free names of P and Q as different. Thus, ∇ quantification can help encode distinction in the case of open bisimulation. As shown in [TM04], assuming the excluded middle assumption on the equality of names, namely, the formula

$$\forall w \forall x [x = w \vee x \neq w],$$

allows us to capture strong bisimulation instead.

We now define three binary relations Obisim_p, $\mathit{Obisim}_{n\to p}$, and $\mathit{Obisim}_{v\to p}$ with respect to provability involving the definition in Figure 2 and any given SOS description following the lines described in Section 2.2.

Let $\langle P, Q\rangle$ be a pair of open terms of the same type and let $\bar{x}$ a the list the variables free in either P or in Q. If P and Q have type p then the set Obisim_p contains $\langle P, Q\rangle$ if and only if $\vdash \forall \bar{x}.\mathit{bisim}\ P\ Q$. If P and Q have type $n \to p$ then the set $\mathit{Obisim}_{n\to p}$ contains $\langle P, Q\rangle$ if and only if $\vdash \forall \bar{x} \forall y.\mathit{bisim}\ (Py)\ (Qy)$. Finally, if P and Q have type $v \to p$ then the set $\mathit{Obisim}_{v\to p}$ contains $\langle P, Q\rangle$ if and only if $\vdash \forall \bar{x} \nabla y.\mathit{bisim}\ (Py)\ (Qy)$.

4 Congruence

We now propose a notion of congruence in our framework.

To be a congruence, a relation should obviously be constructor preserving (taking equality as a relation on names) and be an equivalence relation on each set. Furthermore the addition of abstracted types in our framework requires adding conditions on the interaction between primitive types and abstracted types.

Definition 4.1 The triple $\langle R_p, R_{n\to p}, R_{v\to p}\rangle$ is said to be a *congruence* if for each $\gamma \in \{p, n \to p, v \to p\}$, R_γ is a binary relation of open terms of type γ

and if the following properties hold.

- *(Equ):* The three binary relations are equivalence relations.
- (λ)*:* If $\langle P, Q\rangle \in R_p$ then $\langle \lambda x.P, \lambda x.Q\rangle \in R_{n\to p}$ and $\langle \lambda x.P, \lambda x.Q\rangle \in R_{v\to p}$, depending on the type of x.
- *(App):* If $\langle M, N\rangle \in R_{n\to p}$ and t is a term of type n, then $\langle Mx, Nx\rangle \in R_p$. If $\langle M, N\rangle \in R_{v\to p}$ and M and N are closed and t is a constructor of type n that appears in neither N nor M, then $\langle Mt, Nt\rangle \in R_p$.
- *(Cons):* If c is a constructor of type $t_1 \to t_2 \to \cdots \to t_n \to p$, and if $\langle P_1, Q_1\rangle, \ldots, \langle P_n, Q_n\rangle$ are in $R_{t_1}, \ldots, R_{t_n}$, $\langle c(P_1, \ldots, P_n), c(Q_1, \ldots, Q_n)\rangle$ is in R_p.

The properties in Definition 4.1 are the minimum requirements for a relation to be compositional in our setting: in addition to the standard conditions, one needs the (λ) property to ensure that constructor with abstraction in their types will behave as expected according to the standard notion of congruence. In fact, consider for instance the case of the *in* constructor of the π-calculus. Without the (λ) property, one could have $\langle P, Q\rangle \in R_p$ but not $\langle (\lambda x.P), (\lambda x.Q)\rangle \in R_{n\to p}$ and the $(Cons)$ property would not ensure $\langle in\ y\ x\ P,\ in\ y\ x\ Q\rangle \in R_p$. This is exactly the reason why strong bisimilarity is not a congruence (for the π-calculus) in the standard sense. Note that in [Mil99] congruence is formulated using a much weaker requirement than (λ): $\langle (\lambda x.P)$ and $(\lambda x.Q)\rangle$ are forced to be related only if all the instances of P and Q are related. As a consequence, in [Mil99] strong bisimilarity is a congruence.

Notice that the first three properties defining a congruence hold for

$$\langle Obisim_p, Obisim_{n\to p}, Obisim_{v\to p}\rangle,$$

no matter what SOS rules are used, by merit of the proof-theoretic origin of their definition. These properties are inherent to the definition of the relation more than to the system: It is the "constructor preserving" condition that requires restrictions on the transition system in order to hold.

5 The rule format

In [GV92], two restrictions on the format of rules where introduced to ensure that bisimilarity is a congruence. We provide generalizations of these two restrictions so that we can claim the same theorem in the extended setting.

Definition 5.1 The tyft/tyxt format for SOS restricts all inference rules to be either of the form

$$\frac{\left\{ \mathcal{Q}_i(P_i \xrightarrow[\gamma_i]{A_i} \hat{Y}_i) \mid i \in I \right\}}{(f\ X_1\ \ldots\ X_n) \xrightarrow[\gamma]{A} Q}\ ,$$

where all the schema variables X_j and Y_i are distinct, except for the name variables (i.e. variables of type n)
or

$$\frac{\left\{\{\mathcal{Q}_i(P_i \xrightarrow[\gamma]{A_i} \hat{Y}_i) \mid i \in I\right\}}{X \xrightarrow[\gamma]{A} Q}, \quad \text{where } X \text{ and all the schema variables } Y_i \text{ are distinct.}$$

Here, as before, $\mathcal{Q}_i$ denotes a possibly empty sequence of ∇-quantified variables. The notation $\hat{Y}_i$ denotes a term of the form $(Y_i\ u_1 \ldots u_k)$ where $\mathcal{Q}_i$ is $\nabla u_1 \ldots \nabla u_k$. The type variable γ ranges over the set $\{p, n \to p, v \to p\}$

In the standard definition of tyft/tyxt format, a premise of a rule is of the form $P_i \xrightarrow{A_i} Y_i$, where Y_i is a variable (of type p). This requirement that a variable appears as the continuation of the transition implies that an inference rule can not be selected by first probing the structure of the result of a transition. For example, a premise of the form $P_i \xrightarrow{A_i} 0$ is explicitly ruled out. When there are binding surrounding such a labeled transition judgment, it is necessary to replace that variable by a variable that is applied to all those ∇-bindings: that is, the premise format becomes

$$\nabla u_1 \ldots \nabla u_k \left(P_i \xrightarrow[\gamma_i]{A_i} (Y_i\ u_1 \ldots u_k) \right)$$

Notice that instances of this premise will result in continuations that may or may not contain the variables $u_1, \ldots, u_k$. If fewer ∇-bound variables were in this application, however, the rule could be used to probe some of the structure of the continuation. For example, the premise of the (open) rule in Figure 1 is $\nabla y(Py \xrightarrow[p]{\uparrow Xy} Qy)$. If instead that premise was $\nabla y(Py \xrightarrow[p]{\uparrow Xy} Q')$ then the premise would match only those output transitions in which the ∇-bound variable y was not bound (recall that substitution is capture-avoiding in logic). Such ability to probe the structure of a continuation can result in a lost of congruence for bisimilarity.

Definition 5.2 The *dependency graph* of a rule with premise set

$$\{\mathcal{Q}_i(P_i \xrightarrow[\gamma]{A_i} \hat{Y}_i) \mid i \in I\}$$

has I as its nodes and arrows from i to j if Y_i is free in P_j. A rule is *without circular dependencies* if this graph is acyclic.

We also, of course, require that all inference rules are properly typed. Such a typing restriction adds another way in which rules are constrained. For example, consider an inference rule that contains a meta-variable M of type $v \to p$ and contains a premise or conclusion containing the formula $P \xrightarrow[n \to p]{A} \lambda x.Qx$. Proper typing will imply that the variable M cannot be applied to x in the expression Qx: in essence, the expression M, which requires a new name, cannot be applied to the variable x.

We now state our main result.

Theorem 5.3 *If the specification of labeled transitions in tyft/tyxt format and all rules have no circular dependences, then the relation induced by bisim for this system is a congruence according to Definition 4.1.*

The structure of this proof follows in part the one in [GV92]. There are novelties related to the treatment of names, which are mainly handled by our proof-theoretic setting. We only illustrate here the structure of the proof. The full proof of the main lemma appears in the appendix of [ZMP05].

First, consider the function $\mathcal{C}(t, s, \gamma)$, where t and s are terms of type γ and $\gamma \in \{n, p, v \to p, n \to p\}$. It then takes values as follows.

$$\mathcal{C}(t, s, n) = (t = s) \qquad \mathcal{C}(t, s, v \to p) = \nabla y.congr\ (ty)\ (sy)$$
$$\mathcal{C}(t, s, p) = congr\ t\ s \qquad \mathcal{C}(t, s, n \to p) = \forall y.congr\ (ty)\ (sy)$$

Second, we give an inductive definition for *congr* that contains the clause

$$congr\ P\ Q \stackrel{\mu}{=} bisim\ P\ Q$$

as well as the clauses

$$congr\ (f\ T_1 \ldots T_n)\ (f\ S_1 \ldots S_n) \stackrel{\mu}{=} \bigwedge_{i=1}^{n} \mathcal{C}(T_i, S_i, \gamma_i),$$

one for each constructor $f : \gamma_1 \to \cdots \to \gamma_n \to p$ of process expressions. Thus, if there are j constructors for type p then there are $j + 1$ clauses defining *congr*. Despite its appearance, the symbol $\stackrel{\mu}{=}$ is not literally an equivalence. Instead, it is used to indicate that *congr* is inductively defined by mutual recursion using *all clauses* marked by the $\stackrel{\mu}{=}$ symbol. To illustrate the second kind of definition clause above, the following are the corresponding clauses for $+ : p \to p \to p$ and for $in : n \to (n \to p) \to p$.

$$congr\ (T_1 + T_2)\ (S_1 + S_2) \stackrel{\mu}{=} congr\ T_1\ S_1 \wedge congr\ T_2\ S_2$$
$$congr\ (in\ X\ R)\ (in\ Y\ S) \stackrel{\mu}{=} X = Y \wedge \forall w.\ congr\ (Rw)\ (Sw)$$

Next, we wish to show that the predicate *congr* equals *bisim*. Clearly the latter is a subset of the former. To prove the converse, we simply need to show that *congr* is a post-fixed point of the definition of *bisim*. Since *bisim* is defined co-inductively, this would imply that the interpretation of *congr* is included in *bisim*. To proceed, we prove the following lemma.

Lemma 5.4 *Let B be the formula defined in Figure 3. The formula*

$$\forall P \forall Q.\ congr\ P\ Q \supset B\ congr\ P\ Q$$

is provable.

This part of the proof is similar to the standard case in its structure. The

$B = \lambda r \lambda P \lambda Q.$

$$\begin{array}{l}
\forall A \forall P' \, [P \xrightarrow[p]{A} P' \supset \exists Q'.Q \xrightarrow[p]{A} Q' \wedge r \; P'Q'] \wedge \\
\forall A \forall Q' \, [Q \xrightarrow[p]{A} Q' \supset \exists P'.P \xrightarrow[p]{A} P' \wedge r \; Q'P'] \wedge \\
\forall A \forall P' \, [P \xrightarrow[n \to p]{A} P' \supset \exists Q'.Q \xrightarrow[n \to p]{A} Q' \wedge \forall w.r \; (P'w)(Q'w)] \wedge \\
\forall A \forall Q' \, [Q \xrightarrow[n \to p]{A} Q' \supset \exists P'.P \xrightarrow[n \to p]{A} P' \wedge \forall w.r \; (Q'w)(P'w)] \wedge \\
\forall A \forall P' \, [P \xrightarrow[v \to p]{A} P' \supset \exists Q'.Q \xrightarrow[v \to p]{A} Q' \wedge \nabla w.r \; (P'w)(Q'w)] \wedge \\
\forall A \forall Q' \, [Q \xrightarrow[v \to p]{A} Q' \supset \exists P'.P \xrightarrow[v \to p]{A} P' \wedge \nabla w.r \; (Q'w)(P'w)]
\end{array}$$

Fig. 3. A higher-order λ-expression used to prove a co-inductive property.

interested reader can refer to the appendix of [ZMP05] for more details and to [GV92] to compare our proof to the standard case.

From this lemma, we can deduce that the relation induced by the two predicates *congr* and *obisim* coincides.

We can now prove that $\langle Obisim_p, Obisim_{n \to p}, Obisim_{v \to p} \rangle$ is a congruence according to our definition. As we commented at the end of Section 4, the first three conditions in the definition of congruence hold automatically given the logic used to describe this triple of relations. The fourth condition holds immediately for *Congr* and since that relation coincides with $Obisim_p$, it holds for $Obisim_p$ as well.

We have now proved that our format is enough to ensure that open bisimilarity is a congruence. The use of quantification on names and its interaction with binders allows our proofs to be almost as simple as in the first order case.

It is easy to see that π-calculus as presented earlier satisfies our extended tyft/tyxt format, for both the late and early transition systems. We have thus provided another proof that open bisimilarity is a congruence in these cases.

6 Conclusion and future work

We have presented a format to ensure that open bisimilarity is a congruence for name-passing calculi. To obtain this result, we used a new presentation of transition system specifications, better suited for reasoning about name-passing and name-binding, based on an extension of SOS exploiting a logic-based approaches to binding. In fact, by using an enriched logic able to handle bindings internally, we were able to naturally re-use standard first-order techniques to get our result.

This work opens several interesting fields of extension. First of all, the fact that all our work is done within proof theory permits using standard

higher-order logic programming technique to provide executable specifications of SOS rules as well as symbolic bisimulation for (finite) process expressions [TM04,Tiu05]. The recent Level0/1 system [TNM05] has been used to provide just such implementations for the π-calculus.

Besides, it would be both useful and insightful to express other name-passing calculi in our framework. The distinction made between the different kind of bound transition should allow us to model fusion calculus' hyperequivalence ([PV98]), and even seems well-suited for specifying the operational semantic of calculi based on D-fusion ([BBM04]), and maybe solve the problem of congruence of bisimilarity in this case. Handling extension of the π-calculus, like Abadi and Fournet's Applied Pi-Calculus ([AF01]), would also be very interesting. Here the hope is that our fully proof-theoretical framework will provide us with convenient tools to handle unification of terms. Finally, it would be very interesting to extend our work to process-passing calculi, as done in [MGR05], since it would provide us with a united framework to handle name-binding and higher-order.

Relating this work to other work on operational semantics based on model theoretic semantics, such as [**?**], is certainly an interesting direction to pursue.

Acknowledgments. We are grateful for the support of the ACI grants GEOCAL and Rossignol and for comments on an earlier draft of this paper from Alwen Tiu. Work of the first author was partially supported by ENS, Paris.

References

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *POPL '01: Proc. of the 28th Symposium on Principles of programming languages*, pages 104–115. ACM Press, 2001.

[BBM04] M. Boreale, M. Buscemi, and U. Montanari. D-fusion: a distinctive fusion calculus. In *Proc. of APLAS04*, volume 3302 of *LNCS*. Springer, 2004.

[Ber98] Karen L. Bernstein. A congruence theorem for structured operational semantics of higher-order languages. In John Mitchell, editor, *Proceedings of LICS98*, pages 153–163. IEEE, June 1998.

[BIM95] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, January 1995. Preliminary version appeared in POPL 1988: 229-239.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[Des00] Joëlle Despeyroux. A higher-order specification of the π-calculus. In *Proc. of the IFIP International Conference on Theoretical Computer Science, Sendai, Japan, August 17-19, 2000.*, August 2000.

[GV92] Jan Friso Groote and Frits Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100:202–260, 1992.

[How96] Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.

[MGR05] Mohammad Reza Mousavi, Murdoch J. Gabbay, and Michel A. Reniers. SOS for higher order precesses. In *CONCUR'05: Concurrency Theory*, 2005. To appear.

[Mil99] Robin Milner. *Communicating and Mobile Systems : The π-calculus*. Cambridge University Press, 1999.

[MMP03] Raymond McDowell, Dale Miller, and Catuscia Palamidessi. Encoding transition systems in sequent calculus. *Theoretical Computer Science*, 294(3):411–437, 2003.

[MN87] Dale Miller and Gopalan Nadathur. A logic programming approach to manipulating formulas and programs. In Seif Haridi, editor, *IEEE Symposium on Logic Programming*, pages 379–388, San Francisco, September 1987.

[MP99] Dale Miller and Catuscia Palamidessi. Foundational aspects of syntax. In P. Degano, R. Gorrieri, A. Marchetti-Spaccamela, and P. Wegner, editors, *ACM Computing Surveys Symposium on Theoretical Computer Science: A Perspective*, volume 31. ACM, September 1999.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, Part I. *Information and Computation*, 100(1):1–40, September 1992.

[MT03a] Dale Miller and Alwen Tiu. A proof theory for generic judgments. ACM Trans. on Computational Logic, ed. Phokion Kolaitis. To appear., November 2003.

[MT03b] Dale Miller and Alwen Tiu. A proof theory for generic judgments: An extended abstract. In *Proc. 18th IEEE Symposium on Logic in Computer Science (LICS 2003)*, pages 118–127. IEEE, June 2003.

[MT03c] Alberto Momigliano and Alwen Tiu. Induction and co-induction in sequent calculus. In Mario Coppo Stefano Berardi and Ferruccio Damiani, editors, *Post-proceedings of TYPES 2003*, number 3085 in LNCS, pages 293 – 308, January 2003.

[Par01] Joachim Parrow. An introduction to the pi-calculus. In Bergstra, Ponse, and Smolka, editors, *Handbook of Process Algebra*, pages 479–543. Elsevier, 2001.

[PE88] Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *Proc. of the Conference on Programming Language Design and Implementation*, pages 199–208. ACM Press, June 1988.

[Plo81] G. Plotkin. A structural approach to operational semantics. DAIMI FN-19, Aarhus University, Aarhus, Denmark, September 1981.

[PV98] Joachim Parrow and Bjorn Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Logic in Computer Science*, pages 176–185, 1998.

[Sta85] Richard Statman. Logical relations and the typed λ-calculus. *Information and Control*, 65:85–97, 1985.

[SW01] Davide Sangiorgi and David Walker. *π-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[Tho93] Bent Thomsen. Plain chocs: A second generation calculus for higher order processes. *Acta Inf.*, 30(1):1–59, 1993.

[Tiu04] Alwen Tiu. *A Logical Framework for Reasoning about Logical Specifications*. PhD thesis, Pennsylvania State University, May 2004.

[Tiu05] Alwen Tiu. Model checking for π-calculus using proof search. In *CONCUR'05: Concurrency Theory*, 2005. To appear.

[TM04] Alwen Tiu and Dale Miller. A proof search specification of the π-calculus. In *3rd Workshop on the Foundations of Global Ubiquitous Computing*, September 2004.

[TNM05] Alwen Tiu, Gopalan Nadathur, and Dale Miller. Mixing finite success and finite failure in an automated prover. Submitted, May 2005.

[Ver95] Chris Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. *Nordic J. of Computing*, 2(2):274–302, 1995.

[Zie04] Axelle Ziegler. Un format pour que la bisimulation soit une congruence dans les langages de processus avec mobilité. Technical report, ENS, 2004. Master Thesis.

[ZMP05] Axelle Ziegler, Dale Miller, and Catuscia Palamidessi. A congruence format for name-passing calculi. Technical report, LIX, Ecole Polytechnique, 2005. Available at `http://www.lix.polytechnique.fr/parsifal/ziegler05report.pdf`.

A Proof of congruence result

We'll now describe the proof of the Lemma 5.4. We shall assume that the reader is familiar with basic ideas behind sequent calculus and the main ideas about the use of definitions to provide fixed points in such proofs: see, for example, [MMP03,MT03a].

In order to prove the sequent

$$P : p, Q : p \,;\, congr\ P\ Q \vdash B\ congr\ P\ Q$$

we may assume that it is the result of using the definition left introduction rule for the *congr* predicate, which then yields the following two sequents to prove:

$$P\colon p,\ Q\colon p \,;\, bisim\ P\ Q \vdash B\ congr\ P\ Q$$

$$T_1\colon \gamma_1, S_1\colon \gamma_1, \ldots, T_n\colon \gamma_n, S_n\colon \gamma_n \,;\, \bigwedge \mathcal{C}(T_i,\ S_i,\ \gamma_i) \vdash B\ congr\ f(\bar{T})\ f(\bar{S}).$$

In the second case, P and Q unified with $f(T_1, \ldots, T_n)$ and $f(S_1, \ldots, S_n)$, respectively, where f is a constructor of type $\gamma_1 \to \cdots \to \gamma_n \to p$.

In the first case, one unfolding of the definition for *bisim* yields (via the left-introduction rule for definitions) the sequent

$$P\colon p,\ Q\colon p \,;\, B\ bisim\ P\ Q \vdash B\ congr\ P\ Q.$$

Given the definition of *congr*, this sequent has a simple proof.

In the second case, things are slightly more complicated. One can write $B\ congr\ P\ Q$ as the following formula:

$$\bigwedge\nolimits_\gamma \left[\forall A \forall P' \left(P \xrightarrow[\gamma]{A} P' \supset \exists Q'[Q \xrightarrow[\gamma]{A} Q' \wedge \mathcal{C}(P', Q', \gamma)]\right)\right] \wedge$$
$$\bigwedge\nolimits_\gamma \left[\forall A \forall Q' \left(Q \xrightarrow[\gamma]{A} Q' \supset \exists P'[P \xrightarrow[\gamma]{A} P' \wedge \mathcal{C}(P', Q', \gamma)]\right)\right],$$

where γ ranges over $\{p, n \to p, v \to p\}$. Application of the conjunction right introduction rules yields six sequents, consisting of two groups of three symmetrical sequents. Since the problem is symmetrical, we prove the following three sequents (one for each $\gamma \in \{p, n \to p, v \to p\}$):

$$\Sigma \,;\, \bigwedge \mathcal{C}(T_i, S_i, \gamma_i) \vdash \forall A \forall P' (f(\bar{T}) \xrightarrow[\gamma]{A} P') \supset \exists Q'[(f(\bar{S}) \xrightarrow[\gamma]{A} Q') \wedge \mathcal{C}(P', Q', \gamma)],$$

where Σ is $T_1\colon \gamma_1, S_1\colon \gamma_1, \ldots, T_n\colon \gamma_n, S_n\colon \gamma_n$. Application of standard inference rules yields instead the following sequent to prove:

$$\Sigma, A, P'\colon \gamma \,;\, \bigwedge \mathcal{C}(T_i, S_i, \gamma_i), f(\bar{T}) \xrightarrow[\gamma]{A} P' \vdash \exists Q'[f(\bar{S}) \xrightarrow[\gamma]{A} Q' \wedge \mathcal{C}(P', Q', \gamma)]$$

One can now use one of the rule of our specification to unify with the transition predicate on the left-hand side. If there is no such unification, the sequent is proved. Otherwise, we have a sequent for each rule and each unifier. Since they are all of the same form, we shall consider just one of those cases where the rule used is

$$\frac{\mathcal{Q}_j\left(R_j \xrightarrow[\delta_j]{B_j} \hat{Y_j}\right)}{f(X_1,\ldots,X_n) \xrightarrow[\gamma]{B} R} \qquad (*)$$

and the substitution is θ, such that $\theta(X_i) = T_i$, $\theta(B) = A$ and $\theta(R) = P'$. We may assume that the variables free in the range of θ are distinct from any ∇-bound variables in any premise of this rule. This leaves us with the following sequent to prove:

$$\Sigma, \theta(B), \theta(R)\,;\, \bigwedge \mathcal{C}(T_i, S_i, \gamma_i),\ \bigwedge_j \mathcal{Q}_j[\theta(R_j) \xrightarrow[\delta_j]{\theta(B_j)} \theta(\hat{Y_j})] \vdash$$
$$\exists Q'\left(f(\bar{S}) \xrightarrow[\gamma]{\theta(B)} Q' \wedge \mathcal{C}(\theta(R), Q', \gamma)\right)$$

By the expression $\Sigma, \theta(B), \theta(R)$ we mean the signature resulting from collecting together the variables in Σ with any free variables in $\theta(B)$ or in $\theta(R)$.

We next need to build a instantiation for the $\exists Q'$ quantifier and we do this by constructing a substitution θ' such that this existential quantifier is instantiated with $\theta'(R)$.

Let Z be the set of all free (meta) variables in the rule $(*)$. For each variable $z \in Z$ we define its *rank*, $rk(z)$ as follows: if $z = Y_i, i \in I$, $rk(z)$ is the length of the greatest path in the dependency graph starting from i. Otherwise, $rk(z) = 0$. The set Z is thus partitioned into sets $Z_i = \{z \mid rk(z) = i\}$. We define θ' inductively over these sets. First, we define θ' on the set Z_0. We have two cases:

- If the meta variable X_i in $(*)$ is of rank 0, then set $\theta'(X_i) = S_i$.
- For all other name variables process variables of rank 0, set $\theta'(z) = \theta(z)$.

For the remainder of the variables, we'll define θ' by induction on the rank i of the variable. The induction invariant is for all variables of rank smaller than i, we can prove $\mathcal{C}(\theta(z), \theta'(z), \delta)$, where δ is the type of z. This property is respected at rank 0: T_i and S_i are defined as belonging to the proper sets, and all of our relations are guaranteed to be reflexive. To build θ' we need the following lemma.

Lemma A.1 *Let T be a sub-term of type γ of either the conclusion or any premises of $(*)$. If for every free variable x of T, the expression $\mathcal{C}(\theta(x), \theta'(x), \delta_x)$ is provable (where δ_x is the type of x) in a given context then $\mathcal{C}(\theta(T), \theta'(T), \delta)$ is provable in the same context.*

Proof. We proceed by structural induction on T.

- If T is a variable, the result is immediate.
- If T is of the form $f(T_1, \ldots, T_n)$, then by the induction hypothesis, we can build proofs of $\mathcal{C}(\theta(T_i), \theta'(T_i), \gamma_i)$ (where γ_i is the type of T_i). By the definition of *congr*, we thus have a proof of

$$\mathcal{C}(f(\theta(T_1), \ldots, \theta(T_n)), f(\theta'(T_1), \ldots, \theta'(T_n)), \delta),$$

  which is also the formula $\mathcal{C}(\theta(T), \theta'(T), \delta)$.
- If T is of the form $T = Mx$, where M is of type $v \to p$, then x is a ∇-bound variable because the rule type checks, and, by induction hypothesis $\mathcal{C}(\theta(M), \theta'(M), v \to p)$ is provable. Since x is ∇-bound, $\mathcal{C}(\theta(M), \theta'(M), p)$ is provable.
- if T is of the form $T = MT'$ where M is of type $n \to p$ and T is a term of type n, then $\mathcal{C}(\theta(M), \theta'(M), n \to p)$ is provable and $\theta(T') = \theta'(T')$ by induction hypothesis. It's immediate that $\mathcal{C}(\theta(T), \theta'(T), p)$ is provable.
- If T is of the form $\lambda x.M$ with M of type p, the result is again immediate.

□

We can now continue building our substitution θ'. Pick $z \in Z_{i+1}$. Notice that z necessarily is one of the Y_j and as such, there is a transition of the form $\nabla x_1, \ldots x_n R_j \xrightarrow[\delta_j]{B_j} z(x_1, \ldots, x_n)$ in the premises of the rule we are applying, and $\forall z' \in fv(R_j),\ rk(z') \leq i$. From the Lemma A.1 and the induction hypothesis, $\mathcal{C}(\theta(R_j), \theta'(R_j), \delta_j)$ is provable in the context of $(**)$, but such a proof necessarily contains a proof of

$$\theta(R_j) \xrightarrow[\delta_j]{A} \theta(z) \supset \exists R'.\theta'(R_j) \xrightarrow[\delta_j]{A} R' \wedge \mathcal{C}(\theta(z), R', \delta_j).$$

From there, if $\theta(R_j) \xrightarrow[\delta_j]{A} \theta(z)$ unifies to nothing our original sequent is proved. Otherwise, since the sequent above is provable, the proof will contain a term R that unifies with R' and which makes both the right-hand side of our sequent provable. One can set $\theta'(z) = \lambda\ x_1 \ldots \lambda x_n\ R'$. This method allows to build θ' on every free variable. Moreover, the substitution θ' respects the following properties:

(i) $\theta'(f(X_1, \ldots, X_n)) = f(S_1, \ldots, S_n)$

(ii) $\theta'(B) = \theta(B)$

(iii) For all free variable z of type γ in our rule $\mathcal{C}(\theta(z), \theta'(z), \gamma)$ is provable.

Let's have a look at the sequent we wanted to prove:

$$\Sigma, \theta(B), \theta(R)\,;\, \bigwedge \mathcal{C}(T_i, S_i, \gamma_i),\ \bigwedge_j \mathcal{Q}_j[\theta(R_j) \xrightarrow[\delta_j]{\theta(B_j)} \theta(\hat{Y_j})] \vdash$$
$$\exists Q'.f(\bar{S}) \xrightarrow[\gamma]{\theta(B)} Q' \wedge \mathcal{C}(\theta(R), Q', \gamma)$$

We will now instantiate Q' with $\theta'(R)$. We obtain the following sequents:

$$\Sigma, \theta(B), \theta(R), \theta'(R)\,;\, \bigwedge \mathcal{C}(T_i, S_i, \gamma_i),\ \bigwedge_j \mathcal{Q}_j[\theta(R_j) \xrightarrow[\delta_j]{\theta(B_j)} \theta(\hat{Y}_j)] \vdash \theta'(f(\bar{S})) \xrightarrow[\gamma]{\theta(B)} \theta'(R)$$

$$\Sigma, \theta(B), \theta(R), \theta'(R)\,;\, \bigwedge \mathcal{C}(T_i, S_i, \gamma_i),\ \bigwedge_j \mathcal{Q}_j[\theta(R_j) \xrightarrow[\delta_j]{\theta(B_j)} \theta(\hat{Y}_j)] \vdash \mathcal{C}(\theta(R), \theta'(R), \gamma)$$

To prove the first sequent, one can use the same deduction rule that was used on the left earlier. We now need to provide a proof for one such sequent for each j in J:

$$\Sigma, \theta(B), \theta(R), \theta'(R)\,;\, \bigwedge \mathcal{C}(T_i, S_i, \gamma_i),\ \theta(R_j) \xrightarrow[\delta_j]{\theta(B_j)} \theta(\hat{Y}_j) \vdash \theta'(R_j) \xrightarrow[\gamma]{\theta'(B_j)} \theta'(\hat{Y}_j)$$

But they are provable by construction of θ'.

As for the second sequent, one can apply the Lemma A.1 to $(\theta(R))$ and $(\theta'(R))$.

We have thus completed the proof of Lemma 5.4.