A Proof Search Specification of the π-calculus

Alwen Tiu
INRIA Lorraine

Joint work with Dale Miller (*INRIA Futurs/École polytechnique*)

TYPES 2004, 16 December 2004

Motivations

- To use logic, more precisely, *proof theory*, as a *framework* for specifying and reasoning about operational semantics.

- We focus on the presentation of operational semantics as *deductive systems*, e.g., structural operational semantics, and on computation as proof search.

- Reasoning about *mobility* of *names*. The kind of mobility we are considering: link mobility (as in π-calculus), which are basically the changing in scoping of names. Its operational aspects can (hopefully) be understood in the quantification theory of logic.

The λ-tree syntax encoding of operational semantics

- The usual presentations of SOS for name-passing calculi are not "syntax-directed" enough. Various side conditions on names and scoping of names need to be carefully stated along with the inference rules.

- We adopt an abstract syntax, called λ-tree syntax. λ-tree syntax is a logic programming approach to *higher-order abstract syntax*.

- It is based on simply typed λ-calculus: abstractions in process calculi are represented using λ-abstraction, and process constructors are represented as constants with appropriate types. Capture-avoiding substitution is modelled by β-reduction.

An example of λ-tree syntax encoding

An inference rule for (late) transition system of π-calculus:

$$\frac{\mathtt{P} \xrightarrow{\alpha} \mathtt{P}'}{(x)\mathtt{P} \xrightarrow{\alpha} (x)\mathtt{P}'}\, x \notin \mathrm{n}(\alpha) \qquad \frac{\forall x(Px \xrightarrow{A} P'x)}{\nu(\lambda x.Px) \xrightarrow{A} \nu(\lambda x.P'x)}$$

P, P' and A are *scheme variables*, subject to *capture avoiding* susbtitutions. The constants ν is of type $(n \to p) \to p$ where the types n and p denote names and processes.

Since substitution is capture-avoiding, the scheme variable A cannot be instantiated with term containing free occurrences of x.

Sequents and generic judgments

We use Gentzen's *sequent calculus* for presenting our logic. A sequent is an expression of the form

$$\Sigma\,;\, B_1, \ldots, B_n \vdash B_0$$

where $B_0, \ldots, B_n$ are formulas and Σ is the *signature* of the sequent, containing declarations of the free variables in each B_i (also called *eigenvariables*).

To capture properly the notion of name restriction in process calculi, we enrich sequents with locally bound variables within the formulas:

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

where σ_i is a list of variables locally scoped over B_i.

Variables in σ_i are binding occurrences, so the usual α-equivalence applies.

Overview of $FO\lambda^{\Delta\nabla}$: propositional fragment

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

$$\frac{}{\Sigma;\ \sigma\triangleright \bot, \Gamma \vdash \mathcal{B}}\ \bot\mathcal{L} \qquad \frac{}{\Sigma;\ \Gamma \vdash \sigma\triangleright \top}\ \top\mathcal{R}$$

$$\frac{\Sigma;\ \mathcal{B}, \mathcal{B}, \Gamma \vdash \mathcal{C}}{\Sigma;\ \mathcal{B}, \Gamma \vdash \mathcal{C}}\ c\mathcal{L} \qquad \frac{\Sigma;\ \Gamma \vdash \mathcal{C}}{\Sigma;\ \mathcal{B}, \Gamma \vdash \mathcal{C}}\ w\mathcal{L}$$

$$\frac{\Sigma;\ \sigma\triangleright B_i, \Gamma \vdash \mathcal{D}}{\Sigma;\ \sigma\triangleright B_1 \wedge B_2, \Gamma \vdash \mathcal{D}}\ \wedge\mathcal{L} \qquad \frac{\Sigma;\ \Gamma \vdash \sigma\triangleright B_1 \qquad \Sigma;\ \Gamma \vdash \sigma\triangleright B_2}{\Sigma;\ \Gamma \vdash \sigma\triangleright B_1 \wedge B_2}\ \wedge\mathcal{R}$$

$$\frac{\Sigma;\ \sigma\triangleright B_1, \Gamma \vdash \mathcal{D} \qquad \Sigma;\ \sigma\triangleright B_2, \Gamma \vdash \mathcal{D}}{\Sigma;\ \sigma\triangleright B_1 \vee B_2, \Gamma \vdash \mathcal{D}}\ \vee\mathcal{L} \qquad \frac{\Sigma;\ \Gamma \vdash \sigma\triangleright B_i}{\Sigma;\ \Gamma \vdash \sigma\triangleright B_1 \vee B_2}\ \vee\mathcal{R}$$

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

Overview of $FO\lambda^{\Delta\nabla}$: quantifiers

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

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

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

Overview of $FO\lambda^{\Delta\nabla}$: raising of types

Dependency between eigenvariables and local variables is encoded using the technique of $\forall$-lifting [Paulson] or *raising* [Miller92] of the types of the eigenvariables. Example:

$$\dfrac{\dfrac{\{x_\alpha, h_{\tau\to\gamma\to\beta}\};\ \Gamma \vdash (a_\tau, b_\gamma)\triangleright B\ (h\, a\, b)\ b}{\{x_\alpha\};\ \Gamma \vdash (a_\tau, b_\gamma)\triangleright \forall_\beta y.B\ y\ b}\,\nabla\mathcal{R}}{\{x_\alpha\};\ \Gamma \vdash (a_\tau)\triangleright \nabla_\gamma z.\forall_\beta y.B\ y\ z}\,\forall\mathcal{L}$$

Overview of $FO\lambda^{\Delta\nabla}$: a notion of definitions

We extend the logic further by allowing a non-logical constants (predicate) to be introduced. To each predicate, we associate some *definition clauses*. We write

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

to denote a definition clause for predicate p. This notion of definition has been previously studied by Schroeder-Heister, Girard, Miller and McDowell.

Definitions and Equality

The usual form of definitions allows matching on the head of definition, e.g.,

$$\mathit{nat}\ z \stackrel{\triangle}{=} \top.$$
$$\mathit{nat}\ (sX) \stackrel{\triangle}{=} \mathit{nat}\ X.$$

In our setting, pattern matching is encoded explicitly as equations, e.g.,

$$\mathit{nat}\ X \stackrel{\triangle}{=} (X = z) \vee \exists Y.(X = (sY) \wedge \mathit{nat}\ Y).$$

These two presentations are equivalent. We use the former in presenting examples and the latter for meta-theoretic discussions.

Introduction rules for definitions and equality

$$\frac{\Gamma \vdash \sigma \rhd B\,\bar{t}}{\Gamma \vdash \sigma \rhd p\,\bar{t}}\ \mathit{defR} \qquad \frac{\Gamma, \sigma \rhd B\,\bar{t} \vdash \mathcal{C}}{\Gamma, \sigma \rhd p\,\bar{t} \vdash \mathcal{C}}\ \mathit{defR}$$

where $\forall \bar{x}.[p\,\bar{x} \stackrel{\triangle}{=} B\,\bar{x}]$

$$\frac{\{\Gamma\theta \vdash C\theta \mid \theta \in CSU(\lambda\bar{x}.s, \lambda\bar{x}.t)\}}{\bar{x} \rhd s = t, \Gamma \vdash C}\ \mathrm{eq}\mathcal{L} \qquad \frac{}{\Gamma \vdash \sigma \rhd t = t}$$

Specification of π-calculus

We consider the finite fragment of π-calculus (no replication or recursion):

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

To encode the process expressions in λ-tree syntax, we introduce the types n (names), p (processes), and the following constants

$$0 : p, \quad \tau : p \to p, \quad out : n \to n \to p \to p, \quad in : n \to (n \to p) \to p,$$
$$+ : p \to p \to p, \quad \mid : p \to p \to p, \quad match : n \to n \to p \to p, \quad \nu : (n \to p) \to p.$$

Translation to λ-tree syntax: process expressions

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

Translation to λ-tree syntax: one-step transitions

We introduce the type a to denote *actions*. One-step transition judgments are encoded as follows.

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

where $\downarrow\colon n \to n \to a$ encodes *input action* and $\uparrow\colon n \to n \to a$ encodes *output action*. Transitions involving *bound actions* are encoded as

$$. \xrightarrow{\cdot} .\ : p \to a \to (n \to p)$$

Specification of (late) transition system of π-calculus

Some sample clauses:

$$\frac{P \xrightarrow{A} Q}{(x)P \xrightarrow{\alpha} (x)Q}\ x \notin \mathrm{fn}(\alpha)$$

$$\nu x.Mx \xrightarrow{A} \nu x.Nx \stackrel{\triangle}{=} \nabla x(Mx \xrightarrow{A} Nx).$$

$$\nu x.Mx \xrightarrow{A} \lambda y\ \nu x.Nxy \stackrel{\triangle}{=} \nabla x(Mx \xrightarrow{A} Nx).$$

$$\frac{P \xrightarrow{\alpha} Q}{[x = x]P \xrightarrow{\alpha} Q}$$

$$match\ x\ x\ P \xrightarrow{A} Q \stackrel{\triangle}{=} P \xrightarrow{A} Q.$$

$$match\ x\ x\ P \xrightarrow{A} M \stackrel{\triangle}{=} P \xrightarrow{A} M.$$

$$\frac{P \xrightarrow{\bar{x}w} P'}{(y)P \xrightarrow{\bar{x}(w)} P'[w/y]}\ \begin{array}{l} y \neq x, \\ w \notin \mathrm{fn}((y)P') \end{array}$$

$$\nu y.My \xrightarrow{\uparrow X} M' \stackrel{\triangle}{=} \nabla y(My \xrightarrow{\uparrow Xy} M'y).$$

Example: one-step transition

Consider the process $(y)[x = y]\bar{x}z.0$. It cannot make any transition, since y has to be "fresh".

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{x,z,Q,\alpha\}\,;\, y \triangleright ([x=y](\bar{x}z.0) \xrightarrow{\alpha} Q) \vdash \bot}\, def\mathcal{L}}{\{x,z,Q,\alpha\}\,;\, . \triangleright \nabla y.([x=y](\bar{x}z.0) \xrightarrow{\alpha} Q) \vdash \bot}\, \nabla\mathcal{L}}{\{x,z,Q,\alpha\}\,;\, . \triangleright ((y)[x=y](\bar{x}z.0) \xrightarrow{\alpha} Q) \vdash \bot}\, def\mathcal{L}}{\{x,z,Q,\alpha\}\,;\, \vdash . \triangleright ((y)[x=y](\bar{x}z.0) \xrightarrow{\alpha} Q) \supset \bot}\, \supset\mathcal{R}$$

The success of *def*$\mathcal{L}$ depends on the failure of unification problem

$$\lambda y.x = \lambda y.y.$$

Specification of late bisimulation

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

The need for case split on names

The previous specification is *not* complete. Consider $P = x(u).(\tau.\tau + \tau)$ and $Q = x(u).(\tau.\tau + \tau + \tau.[u = y]\tau)$ and checking $P \sim Q$.

$$Q \xrightarrow{x(u)} \tau.\tau + \tau + \tau.[u = y]\tau \xrightarrow{\tau} [u = y]\tau$$

$$P \xrightarrow{x(u)} \tau.\tau + \tau \longrightarrow ?$$

Here the names u and y are encoded as eigenvariables in logic. The possible matching moves of P:

(1) $\tau.\tau + \tau \xrightarrow{\tau} \tau$ *if* u *is equal to* y

(2) $\tau.\tau + \tau \xrightarrow{\tau} 0.$, *if* u *is not equal to* y

Since name is not inductively defined, this case split cannot be captured intuitionistically.

Adequacy of lbisim

To get a complete encoding of late bisimulation we need to add explicitly excluded middle on names, i.e., the set

$$\begin{array}{rl} \mathcal{E} = & \{ \quad \forall x \forall y (x = y \vee x \neq y), \\ & \quad \nabla n_1 \forall x \forall y (x = y \vee x \neq y), \\ & \quad \nabla n_1 \nabla n_2 \forall x \forall y (x = y \vee x \neq y), \\ & \quad \vdots \\ & \} \end{array}$$

Theorem 1. *Let* P *and* Q *be two processes and let* $\bar{n}$ *be the free names in* P *and* Q*. Then* $\mathtt{P} \sim_l \mathtt{Q}$ *if and only if the sequent*

$$. \,;\, \mathcal{X} \vdash \nabla \bar{n}.\mathit{lbisim}\ \mathtt{P}\ \mathtt{Q}$$

is provable for some $\mathcal{X} \subseteq_f \mathcal{E}$*.*

Open bisimulation

What does *lbisim* P Q correspond to (without the excluded middle assumption)?

It turns out that if we drop the excluded middle axiom and $\forall$-quantify all the free names in P and Q we get a sound and complete encoding of *open bisimulation* [Sangiorgi].

Names in open bisimulation are instantiated lazily, and this bisimulation is closed under arbitrary substitution of free names.

Automation of proof search

- Proof search strategy for open bisimulation: apply left-introduction rules if applicable, otherwise apply right introduction rules. This proof strategy is *complete*.

- Proof search strategy for late bisimulation: need to know when to use the excluded middle on names. A complete proof strategy would be to do case split early, e.g., doing $n + 1$ case split for processes with n distinct free names.

Implementation

- Implementation is rather straightforward: it is based on known old technology, e.g., HOAS based on Church's simply typed λ-calculus, higher-order pattern unification, and other related implementation techniques for logic programming.

- We currently have a working prototype implementation for (automated) checking of open bisimulation. Available from the web: http://www.loria.fr/~tiu

Related work

- Formalization of π-calculus has been done in different frameworks, e.g., in Coq [Hirschkoff TPHOL'97, Despeyroux IFIP-TCS'00, Honsell et. al. TCS'01], in nominal logic [Gabbay'03].

- In those works, formalizing freshness of names, scoping, substitution, etc., is considerable work. In our framework, the same work has to be done, but it is done as a part of the description of the logic. Formalization of open bisimulation, as far as we know, has not been done prior to our work.

- Formalization of π-calculus with replication has been done in [Tiu'04].

Future work

- Relation with *symbolic bisimulation*. Find correspondence between decision procedure in, e.g., [Hennessy and Lin, TCS'95] and [Boreale and Nicola, IC. 96], to proof search.

- Formalizing other systems. Current work in progress: spi-calculus and framed bisimulation, modal logics for π-calculus.

- Proof procedures for bisimulation checking for non-terminationg processes, e.g., via *circular proofs*, tabled deduction, domain specialized proof procedures, etc.