A Proof Theoretic Approach to Operational Semantics
(Focus on binders)

Dale Miller, *INRIA-Futurs and LIX, École Polytechnique*

Based on technical results in:

- M & Tiu: "A Proof Theory for Generic Judgments", LICS03
- Tiu & M: "A Proof Search Specification of the π-Calculus", FGUC04
- Tiu: "Model Checking for π-Calculus Using Proof Search", CONCUR05
- Ziegler, M, Palamidessi: "A congruence format for name-passing calculi", SOS05

Two slogans about bindings

(I) From Alan Perlis's *Epigrams on Programming*: As Will Rogers would have said, "There is no such thing as a free variable."

(II) The *names* of binders are the same kind of fiction as *white space*: they are artifacts of how we write expressions and have *zero semantic content*.

To specify or implement a logic for dealing with bindings, one must, of course, deal with the complexity of names.

Church provided a specification of such a logic in 1940 with his paper on "A Formulation of the Simple Theory of Types." We shall work in this *Paradise of (the) Church*.

Example: Binding a variable in a proof

When proving a universal quantifier, one uses a "new" or "fresh" variable.

$$\frac{B_1, \ldots, B_n \longrightarrow Bv}{B_1, \ldots, B_n \longrightarrow \forall x_\tau . Bx}\ \forall\mathcal{R},$$

provided that v is a "new" variable (not free in the lower sequent). Such new variables are called *eigenvariables*.

But this violates the "Perlis principle." Instead, we write

$$\frac{\Sigma, v\colon \tau : B_1, \ldots, B_n \longrightarrow Bv}{\Sigma : B_1, \ldots, B_n \longrightarrow \forall x_\tau . Bx}\ \forall\mathcal{R},$$

Here, we assume that the variables in the new context (signature) are bindings over the sequent.

Eigenvariables are bound variables.

Higher-Order Abstract Syntax

"If your object-level syntax contain binders, then map these binders to binders in the meta-language."

Functional Programming: binders describe function spaces.

Logic Programming (aka proof search; eg, λProlog): binder are typed λ-expressions modulo α, β, and η conversions.

These approaches are different. Consider $\forall w_i.\ \lambda x.x \neq \lambda x.w \quad (*)$.

FP: $(*)$ is not a theorem, since the identity and the constant valued function coincide on singleton domains.

LP: $(*)$ is a theorem since no instance of $\lambda x.w$ can equal $\lambda x.x$.

λ-tree syntax: HOAS in the proof search setting.

Unification with binders

Binding is built into “higher-order unification” and “unification under a mixed prefix.”

The following are equivalent and fail to unify.

$$\exists w_i.\ \lambda x.x = \lambda x.w \qquad\qquad \exists w_i \forall x.\ x = w$$

Quantifier scope matters. The unification problem

$$\forall a_i \exists f_{i \to i}.(f a) = (g a a),$$

has four unifiers: $f \mapsto \lambda w.gww$, $\lambda w.gaw$, $\lambda w.gwa$, or $\lambda w.gaa$. Switching around the binders yields

$$\exists f_{i \to i} \forall a_i.(f a) = (g a a)$$

with a unique unifier: $f \mapsto \lambda w.gww$.

More generally $\forall x \exists y \forall z \exists u \ldots$.

Dynamics of binders during proof search

During computation, binders can be *instantiated*

$$\frac{\Sigma : \Delta, \mathit{typeof}\ c\ (\mathit{int} \rightarrow \mathit{int}) \longrightarrow C}{\Sigma : \Delta, \forall\alpha(\mathit{typeof}\ c\ (\alpha \rightarrow \alpha)) \longrightarrow C}\ \forall\mathcal{L}$$

or they can *move*.

$$\cfrac{\cfrac{\Sigma, x : \Delta, \mathit{typeof}\ x\ \alpha \longrightarrow \mathit{typeof}\ \lceil B \rceil\ \beta}{\Sigma : \Delta \longrightarrow \forall x(\mathit{typeof}\ x\ \alpha \supset \mathit{typeof}\ \lceil B \rceil\ \beta)}\ \forall\mathcal{R}}{\Sigma : \Delta \longrightarrow \mathit{typeof}\ \lceil \lambda x.B \rceil\ (\alpha \rightarrow \beta)}$$

In this case, the binder named x moves from *term-level* (λx) to *formula-level* ($\forall x$) to *proof-level* (as an eigenvariable in Σ, x).

Example: encoding finite π calculus

Concrete syntax of π-calculus processes:

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

Three syntactic types: n for names, a for actions, and p for processes. The type n may or may not be inhabited.

Three constructors for actions: $\tau : a$ and $\downarrow$ and $\uparrow$ (for input and output actions, resp), both of type $n \to n \to a$.

Abstract syntax for processes is the usual. Restriction: $(y)Py$ is coded using a constant $nu : (n \to p) \to p$ as $nu(\lambda y.Py)$ or as just $nu\ P$. Input prefix $x(y).Py$ is encoded using a constant $in : n \to (n \to p) \to p$ as $in\ x\ (\lambda y.Py)$ or just $in\ x\ P$. Other constructors are done similarly.

π-calculus: one step transitions

The "free action" arrow $\cdot \xrightarrow{\cdot} \cdot$ relates p and a and p.

The "bound action" arrow $\cdot \xrightharpoonup{\cdot} \cdot$ relates p and $n \to a$ and $n \to p$.

$P \xrightarrow{A} Q$ free actions, $A : a$ (τ, $\downarrow xy$, $\uparrow xy$)

$P \xrightharpoonup{\downarrow x} M$ bound input action, $\downarrow x : n \to a$, $M : n \to p$

$P \xrightharpoonup{\uparrow x} M$ bound output action, $\uparrow x : n \to a$, $M : n \to p$

Some SOS rules presented as quantified "reverse" implications.

OUTPUT–ACT:	$\forall x, y, P.$	$\bar{x}y.P \xrightarrow{\uparrow xy} P$	$\subset$	$\top$
INPUT–ACT:	$\forall x, M.$	$x(y).My \xrightharpoonup{\downarrow x} M$	$\subset$	$\top$
MATCH:	$\forall x, P, Q.$	$[x = x]P \xrightarrow{\alpha} Q$	$\subset$	$P \xrightarrow{\alpha} Q$
RES:	$\forall P, Q.$	$(x)Px \xrightarrow{\alpha} (x)Qx$	$\subset$	$\forall x(Px \xrightarrow{\alpha} Qx)$

Proving positives but not negatives

The following can be proved.

Adequacy Theorem: The following are provable from the specification of the π-calculus

$$P \xrightarrow{A} P' \qquad P \xrightharpoonup{\uparrow X} M \qquad P \xrightharpoonup{\downarrow X} M$$

if and only if the "corresponding" transition holds in the π-calculus.

But: If you turn the specification into a "bi-conditional" in the usual way, you still cannot prove interesting negations. For example, there is no proof of

$$\forall x \forall A \forall P.\ \neg[(y)[x = y].\bar{x}x.0 \xrightarrow{A} P]$$

Say good-bye to proving bisimulation.

The fault is in the use of eigenvariables at the meta-level.

Problem: eigenvariables collapse

An attempt to prove $\forall x \forall y. P\, x\, y$ first introduces two new and different eigenvariables c and d and then attempts to prove $P\, c\, d$.

Eigenvariables have been used to encode names in π-calculus [Miller93], nonces in security protocols [Cervesato, et.al. 99], reference locations in imperative programming [Chirimar95], etc.

Since $\forall x \forall y. P\ x\ y \supset \forall z. P\ z\ z$ is provable, it follows that the provability of $\forall x \forall y. P\, x\, y$ implies the provability of $\forall z. P\, z\, z$. That is, there is also a proof where the eigenvariables c and d are identified.

Thus, eigenvariables are unlikely to capture the proper logic behind things like nonces, references, names, etc.

Generic judgments and a new quantifier

Gentzen's introduction rule for $\forall$ on the left is *extensional:* $\forall x$ mean a (possibly infinite) conjunction indexed by terms.

The quantifier $\nabla x. B\, x$ provides a more *"intensional"*, *"internal"*, or *"generic"* reading. It uses a new local context in sequents.

$$\Sigma : B_1, \ldots, B_n \longrightarrow B_0$$

$$\Downarrow$$

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

Σ is a list of distinct eigenvariables, scoped over the sequent and σ_i is a list of distinct variables, locally scoped over the formula B_i.

The expression $\sigma_i \triangleright B_i$ is called a *generic judgment*. Equality between judgments is defined up to renaming of local variables.

The ∇-quantifier

The left and right introductions for ∇ (nabla) are the same.

$$\frac{\Sigma : (\sigma, x : \tau) \triangleright B, \Gamma \longrightarrow \mathcal{C}}{\Sigma : \sigma \triangleright \nabla_\tau x.B, \Gamma \longrightarrow \mathcal{C}} \qquad \frac{\Sigma : \Gamma \longrightarrow (\sigma, x : \tau) \triangleright B}{\Sigma : \Gamma \longrightarrow \sigma \triangleright \nabla_\tau x.B}$$

Standard proof theory design: Enrich context and add connectives dealing with these context.

Quantification Logic: Add the eigenvariable context; add ∀ and ∃.

Linear Logic: Add multiset context; add multiplicative connectives.

Also: hyper-sequents, calculus of structures, etc.

Such a design, augmented with cut-elimination, provides modularity of the resulting logic.

Properties of ∇

This quantifier moves through all propositional connectives:

$$\nabla x \neg Bx \equiv \neg \nabla x Bx \qquad \nabla x (Bx \supset Cx) \equiv \nabla x Bx \supset \nabla x Cx$$

$$\nabla x. \top \equiv \top \qquad \nabla x (Bx \wedge Cx) \equiv \nabla x Bx \wedge \nabla x Cx$$

$$\nabla x. \bot \equiv \bot \qquad \nabla x (Bx \vee Cx) \equiv \nabla x Bx \vee \nabla x Cx$$

It moves through the quantifiers by *raising* them.

$$\nabla x_\alpha \forall y_\beta . Bxy \quad \equiv \quad \forall h_{\alpha \to \beta} \nabla x_\alpha . Bx(hx)$$

$$\nabla x_\alpha \exists y_\beta . Bxy \quad \equiv \quad \exists h_{\alpha \to \beta} \nabla x_\alpha . Bx(hx)$$

Consequence: ∇ can always be given atomic scope within formulas, at the "cost" of raising quantifiers.

Non-theorems

$$\nabla x \nabla y Bxy \supset \nabla z Bzz \qquad \nabla x Bx \supset \exists x Bx^{\dagger}$$

$$\nabla z Bzz \supset \nabla x \nabla y Bxy \qquad \forall x Bx \supset \nabla x Bx^{\dagger}$$

$$\forall y \nabla x Bxy \supset \nabla x \forall y Bxy \qquad \exists x Bx \supset \nabla x Bx$$

† These are theorems using the Pitts new quantifier. (More comparisons later.)

Meta theorems

Theorem: *Cut-elimination.* Given a fixed stratified definition, a sequent has a proof if and only if it has a cut-free proof. (Tiu 2003: also when induction and co-induction are added.)

Theorem: For a fixed formula B,

$$\vdash \nabla x \nabla y. B\, x\, y \equiv \nabla y \nabla x. B\, x\, y.$$

Theorem: If we restrict to *Horn specification* (no implication or negations in the body of the clauses) then

1. $\forall$ and ∇ are interchangeable in specifications.
2. For a fixed B, $\vdash \nabla x. B\, x \supset \forall x. B\, x$.

Returning to the π-calculus

We can now prove

$$\forall w \forall A \forall P.\ \neg.(x)[w = x].\bar{w}w.0 \xrightarrow{A} P$$

This proof requires observing that the equation

$$\lambda x.w = \lambda x.x.$$

has no solution for any instance of w (unification failure).

π-calculus: encoding (bi)simulation

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

This definition clause is not Horn and helps to illustrate the differences between $\forall$ and ∇.

Bisimulation (*bisim*) is easy to write: it has 6 cases.

The early version of bisimulation is a change in quantifier scope.

Learning something from our encoding

Theorem: For the finite π-calculus we have:

P is *open bisimilar* to Q if and only if $\vdash_I \forall \bar{x}.\mathit{bisim}\ P\ Q$.

P is *late bisimilar* to Q if and only if

$$\forall w_n \forall y_n (w = y \vee w \neq y) \vdash_I \nabla \bar{x}.\mathit{bisim}\ P\ Q.$$

Should one assume this instance of *excluded middle*?

Alwen Tiu has built a prototype prover for this logic, restricted to L_λ-unification (higher-order pattern unification). When provided with the above specification of *bisim*, it provides a *symbolic open bisimulation checker*.

Format rules

As Axelle Ziegler illustrated on Monday, specifications of bindings in process calculus can be done declaratively enough to allow for generalization of the tyft/tyxt format rule property.

$$\frac{\cdots \quad \nabla u_1 \ldots \nabla u_k [P \xrightarrow{A} (Y u_1 \ldots u_n)] \quad \cdots}{(f\ X_1\ \ldots\ X_n) \xrightarrow{A} Q}$$

$$\frac{\cdots \quad \nabla u_1 \ldots \nabla u_k [P \xrightarrow{A} (Y u_1 \ldots u_n)] \quad \cdots}{X \xrightarrow{A} Q}$$

That result is essentially the same as the first-order result except that bindings are handled directly (λ-tree syntax, ∇, and mixing of quantifier scopes).

Nothing fundamentally “higher-order” is happening here.

Modal logics

Alwen Tiu recently showed how to specify modal logics for the π-calculus (CONCUR05).

$$P \models \langle\uparrow X\rangle A \quad\subset\quad \exists P'(P \xrightarrow{\uparrow X} P' \wedge \nabla y.P'y \models Ay).$$
$$P \models [\uparrow X] A \quad\subset\quad \forall P'(P \xrightarrow{\uparrow X} P' \supset \nabla y.P'y \models Ay).$$

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

$$P \models [\downarrow X] A \quad\subset\quad \forall P'(P \xrightarrow{\downarrow X} P' \supset \forall y.P'y \models Ay).$$
$$P \models [\downarrow X]^l A \quad\subset\quad \forall P'(P \xrightarrow{\downarrow X} P' \supset \exists y.P'y \models Ay).$$
$$P \models [\downarrow X]^e A \quad\subset\quad \exists y \forall P'(P \xrightarrow{\downarrow X} P' \supset P'y \models Ay).$$

Comparison with Pitts/Gabbay New Quantifier

Fresh Logic:

- Semantics is primary (FM set theory); classical logic basis
- designed for names: an infinite heap of names assumed
- $\mathsf{N}x.Bx$ is analyzed by acquiring a "fresh" name n from the heap and considering Bn.

"Stale" Logic:

- Proof theory is primary (sequent calculus); intuitionistic logic basis (but classical and linear versions are immediate).
- ∇ works for all types; types not assumed to be inhabited
- $\nabla x_\tau.Bx$ is analyzed by hypothesizing a object c of type τ (as in a stack) and considering Bc.

Future Work

Clearly, the π-calculus is just one application. Applied π-calculus? spi-calculus?

Is there a "logical framework" for process calculus here? Do proof search implementations provide means to animate such calculi? Does the meta-theory of the meta-logic help in understanding formal aspects of the calculi?

How to implement *late bisimulation*? How to automate effectively the instances of the excluded middle for equality? Hint: unification failures can tell us which instances we should use.

What is a good model theoretic semantics for ∇? In classical and/or intuitionistic logic?