Bindings, mobility of bindings, and the ∇-quantifier

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

Based on technical results in:

"A Proof Theory for Generic Judgments", LICS2003 & ACM ToCL
"A Proof Search Specification of the π-Calculus", Workshop on the Foundations of Global Ubiquitous Computing, 2004.

Papers and work are joint with Alwen Tiu
(PhD 2004; soon post doc at Loria, Nancy)

Outline

1. Abstract syntax for binders
2. Generic judgments and the ∇-quantification
3. Inference rules for non-logical constants and equality
4. Example: π-calculus

Two slogans

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

Thus: all variables will be bound somewhere.

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

Thus: we focus on bindings abstractly and not on how they are named.

Higher-Order Abstract Syntax

"If your object-level syntax (formulas, programs, types, etc) contain binders, then map these binders to binders in the meta-language."

Functional Programming & Constructive type theories: the binder available is the one for function spaces.

Proof Search (a modern update to logic programming): the binders available are typed λ-expressions with equality (and, hence, unification) 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.

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

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

Dynamics of binders during proof search

During computation, binders can be *instantiated*

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

or they can *move*.

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

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).

Note: The variables in Σ within $\Sigma : \Delta \longrightarrow C$ are eigenvariables and are *bound* over the sequent. Σ is the sequent's *signature*.

The collapse of eigenvariables

An attempt to build a cut-free proof of $\forall x \forall y.P\,x\,y$ 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 cut-free 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.

Quiz

Consider a simple "object-logic" with a pairing constructor $\langle x, y \rangle$.

If the formula $\forall u \forall v[q\ \langle u, t_1 \rangle\ \langle v, t_2 \rangle\ \langle v, t_3 \rangle]$ follows from the assumptions

$$\forall x \forall y[q\ x\ x\ y] \qquad \forall x \forall y[q\ x\ y\ x] \qquad \forall x \forall y[q\ y\ x\ x]$$

what can we say about the terms t_1, t_2, and t_3?

Quiz

Consider a simple "object-logic" with a pairing constructor $\langle x, y \rangle$.

If the formula $\forall u \forall v [q\ \langle u, t_1 \rangle\ \langle v, t_2 \rangle\ \langle v, t_3 \rangle]$ follows from the assumptions

$$\forall x \forall y [q\ x\ x\ y] \qquad \forall x \forall y [q\ x\ y\ x] \qquad \forall x \forall y [q\ y\ x\ x]$$

what can we say about the terms t_1, t_2, and t_3?

Answer: the terms t_2 and t_3 are equal.

Does not matter the domain of the quantifiers $\forall u \forall v$. This conclusion holds for *internal* reasons instead of *external* reasons.

Such an internal treatment does not seem possible if the binders named u and v move to the meta-level as eigenvariables.

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. We also need 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 $\forall$ and $\exists$.

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 ∇

The following are theorems: ∇ moves through all propositional connectives:

$$\nabla x \neg Bx \equiv \neg \nabla x Bx \qquad \nabla x(Bx \supset Cx) \equiv \nabla x Bx \supset \nabla x Cx$$
$$\nabla x.\top \equiv \top \qquad \nabla x(Bx \wedge Cx) \equiv \nabla x Bx \wedge \nabla x Cx$$
$$\nabla x.\bot \equiv \bot \qquad \nabla x(Bx \vee Cx) \equiv \nabla x Bx \vee \nabla x Cx$$

The ∇ moves through the quantifiers by *raising* them.

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

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

Non-theorems and not-yet-theorems

Some non-theorems:

$$\nabla x \nabla y B x y \supset \nabla z B z z \qquad \nabla x B x \supset \exists x B x$$
$$\nabla z B z z \supset \nabla x \nabla y B x y \qquad \forall x B x \supset \nabla x B x$$
$$\forall y \nabla x B x y \supset \nabla x \forall y B x y \qquad \exists x B x \supset \nabla x B x$$

Once we introduce inference rules for definitions and equality, the following can be proved.

$$\nabla x B x \supset \forall x B x \qquad \nabla x B \equiv B \qquad \nabla x \nabla y B\, x\, y \equiv \nabla y \nabla x B\, x\, y$$

A proof theoretic notion of definitions

Introduce non-logical constants (predicate) via *definitions*.

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

The introduction rules for defined atoms are immediate.

$$\frac{\Sigma : \Delta, \sigma \rhd B\bar{t} \longrightarrow C}{\Sigma : \Delta, \sigma \rhd p\bar{t} \longrightarrow C}\ def\mathcal{L} \qquad \frac{\Sigma : \Delta \longrightarrow \sigma \rhd B\bar{t}}{\Sigma : \Delta \longrightarrow \sigma \rhd p\bar{t}}\ def\mathcal{R}$$

Writing several clauses is sometimes more convenient than writing one. For example, the clauses

$$\forall \bar{u}.p\,t_1 \stackrel{\triangle}{=} B_1 \qquad \forall \bar{v}.p\,t_2 \stackrel{\triangle}{=} B_2 \qquad \ldots$$

denote the single clause

$$\forall x.p\,x \stackrel{\triangle}{=} (\exists \bar{u}.x = t_1 \wedge B_1) \vee (\exists \bar{v}.x = t_2 \wedge B_2) \vee \ldots$$

Introduction rules for equality

$$\frac{}{\Sigma : \Delta \longrightarrow \sigma \triangleright t = t} \qquad \frac{\{\Sigma\theta : \Gamma\theta \longrightarrow C\theta \mid \theta \in csu(\lambda\bar{x}.s, \lambda\bar{x}.t)\}}{\Sigma : \Delta, \sigma \triangleright s = t \longrightarrow C}$$

where *csu* stands for "complete set of unifiers." In many cases, this can be replaced by *mgu* (most general unifier).

The set of premises might be empty, finite, or infinite. Each member of this set is a premise.

The signature $\Sigma\theta$ is obtained from Σ by removing variables in the domain of θ, and adding free variables in the range of θ.

Notice that it is eigenvariables that get unified and instantiated.

If the premise set is empty, the proof search is complete. That is: a unification *failure* yields a proof search *success*.

Proof System with Definitions and Equality

This approach to definitions and equality in sequent calculus was first introduced independently by Schroeder-Heister, Girard (fixpoints), and R. Stärk (1990-92).

It was later elaborated by McDowell/Miller/Tiu (1996-2003).

Closely related to the "Clark completion" studied in logic programming.

Tiu has generalized definitions so that they are not just discussing "general fixpoints" but can be used to support induction (least) and coinduction (greatest).

By imposing certain restriction on definitions, we can prove cut-elimination. We must not allow, for example, $p \stackrel{\triangle}{=} \neg p$.

Examples: $1+2=3$ and $1+2\neq 1$

$$\begin{array}{rcl} sum\ z\ N\ N & \triangleq & \top. \\ sum\ (s\ N)\ M\ (s\ P) & \triangleq & sum\ N\ M\ P. \end{array}$$

$$\dfrac{\dfrac{\longrightarrow \top}{\longrightarrow sum\ z\ (s\ (s\ z))\ (s\ (s\ z))}\ def\mathcal{R}}{\longrightarrow sum\ (s\ z)\ (s\ (s\ z))\ (s\ (s\ (s\ z)))}\ def\mathcal{R}$$

$$\dfrac{\dfrac{}{sum\ z\ (s\ (s\ z))\ z \longrightarrow}\ def\mathcal{L}}{sum\ (s\ z)\ (s\ (s\ z))\ (s\ z) \longrightarrow}\ def\mathcal{L}$$

More generally:

$$\dfrac{\Sigma, n:\Gamma(n,n), \top \longrightarrow G(n,n)}{\Sigma, n, m:\Gamma(n,m), sum\ z\ n\ m \longrightarrow G(n,m)}\ def\mathcal{L}$$

Example: computing max

$$
\begin{array}{rl}
a\ (s\ z) & \triangleq \top. \\
a\ (s\ (s\ (s\ z))) & \triangleq \top. \\
a\ z & \triangleq \top. \\
maxa\ N & \triangleq (a\ N) \wedge \forall x(a\ x \supset x \leq N). \\
z \leq N & \triangleq true. \\
(s\ N) \leq (s\ M) & \triangleq N \leq M.
\end{array}
$$

$maxa\ N$ holds if and only if N is the maximum value for a.

$$
\dfrac{\dfrac{\longrightarrow \top}{\longrightarrow a\ 3}\ def\mathcal{R} \qquad \dfrac{\dfrac{\longrightarrow 1 \leq 3 \quad \longrightarrow 3 \leq 3 \quad \longrightarrow 0 \leq 3}{x : a\ x \longrightarrow x \leq 3}\ def\mathcal{L}}{\longrightarrow \forall x(a\ x \supset x \leq 3)}\ \forall\mathcal{R}, \supset \mathcal{R}}{\longrightarrow maxa\ 3}\ def\mathcal{R}
$$

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 coinduction are added.)

Theorem: Given a *noetherian* definition and 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 definitions* (no implication or negations in the body of the definitions) then

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

LINC

LINC stands for a logic with Lambdas, Induction, Nabla, and Coinduction.

Also: LINC Is Not Coq.

Extends $FO\lambda^{\Delta\mathbb{N}}$ of McDowell/Miller.

It is a big logic, providing a framework for proving properties about logic specifications (current target: operational semantics).

Allows induction and coinduction on the λ-tree approach of HOAS.

Example: encoding π calculus

We write the concrete syntax of π-calculus processes as:

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

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

We assume 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 $\cdot \stackrel{\cdot}{\longrightarrow} \cdot$ relates p and a and p.

The "bound action" arrow $\cdot \stackrel{\cdot}{\longrightarrow\!\!\!\!\searrow} \cdot$ relates p and $n \to a$ and $n \to p$.

$P \xrightarrow{A} Q$	free actions, $A : a$ $(\tau, \downarrow xy, \uparrow xy)$
$P \xrightarrow{\downarrow x} M$	bound input action, $\downarrow x : n \to a$, $M : n \to p$
$P \xrightarrow{\uparrow x} M$	bound output action, $\uparrow x : n \to a$, $M : n \to p$

Consider encoding a few one-step rules as definition clauses.

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

Should that last $\forall$ be a ∇? To know, we must leave Horn clauses.

The process $(y)[x = y]\bar{x}z.0$ cannot make any transition. Thus the following statement should be provable.

$$\forall x \forall z \forall Q \forall \alpha.[((y)[x = y](\bar{x}z.0) \xrightarrow{\alpha} Q) \supset \bot]$$

Given the encoding of restriction using $\forall$, this reduces to proving the sequent

$$\{x, z, Q', \alpha\} : \forall y.([x = y](\bar{x}z.0) \xrightarrow{\alpha} Q'y) \longrightarrow \bot$$

No matter what is used to instantiate the $\forall y$, the eigenvariable x can instantiated to the same thing (say, w), and this case leads to the non-provable sequent

$$\{z\} : ([w = w](\bar{w}z.0) \xrightarrow{\bar{w}z} 0) \longrightarrow \bot$$

The universal quantifier was not the correct choice.

Scoping is captured precisely by ∇. Change RES to use ∇.

$$\text{RES}: \quad (x)P \xrightarrow{\alpha} (x)Q \quad \triangleq \quad \nabla x.(P \xrightarrow{\alpha} Q)$$

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

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

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

π-calculus: encoding (bi)simulation

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

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: Assume the finite π-calculus and the bisimulation definition.

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

$\vdash_C \nabla \bar{x}.bisim\ P\ Q$ if and only if P is *late bisimilar* to Q.

The gap can be isolated to be the following instance of the excluded middle:

$$\forall w \forall y.(w = y \vee w \neq y)$$

A straightforward application of proof search principles provides *symbolic open bisimulation*. A prototype built by Tiu uses L_λ-unification (a.k.a. higher-order pattern unification), which has MGUs.

Conclusions

Extended the sequent calculus with the notion of generic judgment and the ∇ quantifier. Proof rules used standard aspects of λ-calculus.

Provided a completely declarative presentation of the operational semantics of the π-calculus without side conditions.

That presentation was so good that we learned something from it:

- a new characterization of open and late bisimulation, and
- a new approach to symbolic open bisimulation using standard (higher-order) logic programming techniques.

Future Work

Generalize the results concerning GSOS and congruence of bisimulation for mobile and for higher-order process 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.

Clearly, the π-calculus is just one application. What can we do with other topics?

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