Bindings, mobility of bindings, and the $\nabla$-quantifier

Dale Miller, INRIA-Saclay and LIX, École Polytechnique

This talk is based on papers with Tiu in LICS2003 & ACM ToCL, and experience using λProlog and Bedwyr (systems developed with Baelde, Gacek, Nadathur, and Tiu).
Outline

1. Bindings or Names?
2. Binder mobility
3. ∇-quantification
4. Example: π-calculus bisimulation
Some 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 (implemented).

This conclusion that “bindings are important and not names” is targeted at *meta-programming*. Names as concrete objects clearly have importance in a number of other tasks: distributed computing, operating systems, the *implementation* of theorem provers, etc.
Concrete syntax vs. parse trees

Church and Gödel did their meta-theory on strings.

For example, Church wrote about “well formed formulas” and “the head is the first non-bracket symbol on the left.”

Concrete syntax has too much concrete nonsense. Parsing allows us to move from strings to algebraic terms (parse trees).

Parse tree vs. λ-tree syntax

Parse tree still have too much concrete nonsense. In particular, the names of bound variables.

λ-tree syntax is an approach to HOAS where bound variable names cannot be accessed.
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 $\lambda$-expressions with equality (and, hence, unification) modulo $\alpha$, $\beta$, and $\eta$ conversions.

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

Functional Prog: (\*) is not always a theorem, since the identity and the constant valued function coincide on singleton domains.

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

$\lambda$-tree syntax is HOAS in the proof search setting.
Some developments in Proof Theory

Uniform (focused) proofs  Used to justify the proof-search paradigm (eg, λProlog). Different development path than for proof-normalization (functional programming).

**A proof theory for fixed points**  Allow unfoldings only. Finite-failure is given a symmetric treatment to finite success. One can capture some must-behavior (eg, bisimulation) and not just may-behavior (eg, reachability).

∇-quantification  The difference between ∇ and ∀ does not “appear” without negation-as-failure. Rich mobility of binders are possible in the resulting logic.

*Bedwyr implements the features above*

**A proof theory for induction/co-induction**  Tiu and Momigliano have approaches to induction, co-induction, and ∇. Baelde has a proof-theory of fixed points to support automation.
Dynamics of binders during proof search

During computation, binders can be *instantiated*

\[
\begin{align*}
\Sigma &: \Delta, \text{typeof } c \text{ (int }\rightarrow\text{ int)} &\rightarrow& C \\
\Sigma &: \Delta, \forall\alpha(\text{typeof } c (\alpha \rightarrow \alpha)) &\rightarrow& C
\end{align*}
\]

∀L

or they can *move.*

\[
\begin{align*}
\Sigma, x &: \Delta, \text{typeof } x \alpha &\rightarrow& \text{typeof } [B] \beta \\
\Sigma &: \Delta &\rightarrow& \forall x (\text{typeof } x \alpha \supset \text{typeof } [B] \beta) \\
\Sigma &: \Delta &\rightarrow& \text{typeof } [\lambda x.B] (\alpha \rightarrow \beta)
\end{align*}
\]

∀R

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

Note: The variables in Σ within Σ : Δ → 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 $\pi$-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

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

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

\[
\forall x \forall y [q \ x \ x \ y] \quad \forall x \forall y [q \ x \ y \ x] \quad \forall x \forall y [q \ y \ x \ x]
\]

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

**Answer:** The terms \( t_2 \) and \( t_3 \) are equal. We wish to prove

\[
\forall t_1 \forall t_2 \forall t_3 [prov (\forall u \forall v [q \langle u, t_1 \rangle \langle v, t_2 \rangle \langle v, t_3 \rangle]) \supset t_2 = t_3]
\]

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 \textit{extensional}: $\forall x$ mean a (possibly infinite) conjunction indexed by terms.

The quantifier $\nabla x. B x$ provides a more \textit{“intensional”}, \textit{“internal”}, or \textit{“generic”} reading. It employs a new local context in sequents.

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

$$\Downarrow$$

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

$\Sigma$ is a list of distinct eigenvariables, scoped over the sequent and $\sigma_i$ is a list of distinct variables, locally scoped over the formula $B_i$.

The expression $\sigma_i \triangleright B_i$ is called a \textit{generic judgment}. Equality between judgments is defined up to renaming of local variables.
The $\nabla$-quantifier

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

$$\begin{align*}
\Sigma : (\sigma, x : \tau) \triangleright B, \Gamma \rightarrow C & \quad \Sigma : \Gamma \rightarrow (\sigma, x : \tau) \triangleright B \\
\Sigma : \sigma \triangleright \nabla \tau x. B, \Gamma \rightarrow C & \quad \Sigma : \Gamma \rightarrow \sigma \triangleright \nabla \tau x. B
\end{align*}$$

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 $\nabla$

The following are theorems: $\nabla$ moves through all propositional connectives:

\[
\begin{align*}
\nabla x \neg Bx & \equiv \neg \nabla x Bx \\
\nabla x (Bx \supset Cx) & \equiv \nabla x Bx \supset \nabla x Cx \\
\nabla x. \top & \equiv \top \\
\nabla x (Bx \land Cx) & \equiv \nabla x Bx \land \nabla x Cx \\
\nabla x. \bot & \equiv \bot \\
\nabla x (Bx \lor Cx) & \equiv \nabla x Bx \lor \nabla x Cx
\end{align*}
\]

The $\nabla$ moves through the quantifiers by *raising* them.

\[
\begin{align*}
\nabla x_\alpha \forall y_\beta . Bxy & \equiv \forall h_{\alpha \rightarrow \beta} \nabla x . Bx(hx) \\
\nabla x_\alpha \exists y_\beta . Bxy & \equiv \exists h_{\alpha \rightarrow \beta} \nabla x . Bx(hx)
\end{align*}
\]

Finally, with equality: $\nabla x . t = s \equiv \lambda x . t = \lambda x . s$.

Thus: $\nabla$ can be implemented for “free” in systems already solving equations involving $\lambda$-terms.
Non-theorems and not-yet-theorems

Some non-theorems:

\[ \nabla x \nabla y B x y \supset \nabla z B z z \quad \nabla x B x \supset \exists x B x \]
\[ \nabla z B z z \supset \nabla x \nabla y B x y \quad \forall x B x \supset \nabla x B x \]
\[ \forall y \nabla x B x y \supset \nabla x \forall y B x y \quad \exists x B x \supset \nabla x B x \]

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

\[ \nabla x B x \supset \forall x B x \quad \nabla x B \equiv B \quad \nabla x \nabla y B x y \equiv \nabla y \nabla x B x y \]

This quantifier seems to be a weaker version of the Pitts-Gabbay “fresh” quantifier.
\textbf{π-calculus: encoding (bi)simulation}

In the atomic formula $P \xrightarrow{A} P'$, the expressions $P$ and $P'$ are processes and $A$ is an action.

In the atomic formulas $P \xrightarrow{\downarrow X} P'$ and $P \xrightarrow{\uparrow X} P'$, the expression $P'$ is an name abstraction over processes and both $\downarrow X$ and $\uparrow X$ are name abstractions over actions ($X$ is a name).

\[
\text{sim } P Q \triangleq \forall A \forall P' \ [P \xrightarrow{A} P' \supset \exists Q'.Q \xrightarrow{A} Q' \land \text{sim } P' Q'] \land \\
\forall X \forall P' [P \xrightarrow{\downarrow X} P' \supset \exists Q'.Q \xrightarrow{\downarrow X} Q' \land \forall w.\text{sim}(P'w)(Q'w)] \land \\
\forall X \forall P' [P \xrightarrow{\uparrow X} P' \supset \exists Q'.Q \xrightarrow{\uparrow X} Q' \land \nabla w.\text{sim}(P'w)(Q'w)]
\]

Bisimulation (\textit{bisim}) is easy to write: it has 6 cases.
Learning something from our encoding

Theorem: Assume the finite $\pi$-calculus and the bisimulation definition.

$\vdash I \forall \bar{x}.\text{bisim} (P\bar{x}) (Q\bar{x})$ if and only if $P\bar{x}$ is open bisimilar to $Q\bar{x}$.

$\mathcal{X} \vdash I \nabla \bar{x}.\text{bisim} (P\bar{x}) (Q\bar{x})$ if and only if $P\bar{x}$ is late bisimilar to $Q\bar{x}$.

Here, $\mathcal{X}$ is a finite set of excluded middle assumptions of the form

\[ \nabla n_1 \ldots \nabla n_k \forall \forall w \forall y. (w = y \lor w \neq y) \quad (k \geq 0) \]

A straightforward application of proof search principles (as provided by, for example, Bedwyr) provides symbolic open bisimulation
Future Work

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

What are some natural strengthening of $\nabla$? How exactly does it compare to Pitts-Gabbay? (See Tiu’s talk.)

How to do induction in the presence of $\nabla$?

Develop an interactive theorem prover that incorporates induction and co-induction along with the model-checking behavior of Bedwyr.

Generalize format rules for operational semantic (such as tyft/tyxt) to mobility in process calculi so that one gets guarantees that (open) bisimulation is a congruence (joint work with Palamidessi and Ziegler).