Focusing and higher-order abstract syntax

Noam Zeilberger

Carnegie Mellon University

12 January 2008
Programs $\leftrightarrow$ Proofs
Programs $\leftrightarrow$ Proofs

typed lambda-calculus $\leftrightarrow$ natural deduction
Programs $\leftrightarrow$ Proofs

typed lambda-calculus $\leftrightarrow$ natural deduction

? $\leftrightarrow$ sequent calculus
Programs $\leftrightarrow$ Proofs

typed lambda-calculus $\leftrightarrow$ natural deduction

? $\leftrightarrow$ sequent calculus

ML, Haskell, ... $\leftrightarrow$ ?
Programs $\leftrightarrow$ Proofs

typed lambda-calculus $\leftrightarrow$ natural deduction

? $\leftrightarrow$ sequent calculus

ML, Haskell, ... $\leftrightarrow$ ?

Practical features

1. explicit evaluation order
2. pattern-matching
Programs $\leftrightarrow$ Proofs

typed lambda-calculus $\leftrightarrow$ natural deduction

? $\leftrightarrow$ sequent calculus?

ML, Haskell, . . .

Practical features

1. explicit evaluation order
2. pattern-matching

Claim 1: focused sequent calculus provides both 1 and 2
Claim 2: these features actually simplify logic & language
The bureaucracy of syntax

Two equivalent $\lambda$-calculus terms:

\[
\text{case}(\pi_1 x, y_1.\text{case}(\pi_2 x, z_1.e_1, z_2.e_2), y_2.\text{case}(\pi_2 x, z_1.e_3, z_2.e_4))
\]

\[
\text{case}(\pi_2 x, z_1.\text{case}(\pi_1 x, y_1.e_1, y_2.e_3), z_2.\text{case}(\pi_1 x, y_1.e_2, y_2.e_4))
\]

Same term in ML:

\[
\text{case } x \text{ of }
\]

| (Inl y1, Inl z1) => e1 |
| (Inl y1, Inr z2) => e2 |
| (Inr y2, Inl z1) => e3 |
| (Inr y2, Inr z2) => e4 |
Bottom-up proof-search

Naive bottom-up search:

1. Find a rule with goal as conclusion
2. Apply rule (premises become the new goals)
3. Upon failure, backtrack and try applying a different rule

Fact: naive bottom-up search is wildly inefficient for LL

Andreoli’s solution: tame proof-search via inversion and focus
Inversion

A rule is *invertible* if its conclusion implies its premises.

Every connective of LL is invertible on either left or right, e.g.:

\[
\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B} \quad (&R) \quad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \quad (\otimes L)
\]
Inversion

A rule is *invertible* if its conclusion implies its premises.

Every connective of LL is invertible on either left or right, e.g.:

\[
\begin{align*}
\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B} (\&R) \\
\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} (\otimes L)
\end{align*}
\]

Inversion stage: chain application of invertible rules.
Inversion

A rule is *invertible* if its conclusion implies its premises.

Every connective of LL is invertible on either left or right, e.g.:

\[
\frac{\Gamma \vdash A \Gamma \vdash B}{\Gamma \vdash A \& B} \quad (\&R) \qquad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \quad (\otimes L)
\]

Inversion stage: chain application of invertible rules

\[
\Gamma; (A \oplus B) \otimes (C \oplus D) \vdash E
\]
Inversion

A rule is *invertible* if its conclusion implies its premises

Every connective of LL is invertible on either left or right, e.g.:

\[
\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \mathbin{\&} B} \quad \text{(\&R)} \quad \frac{\Gamma, A, B \vdash C}{\Gamma, A \mathbin{\otimes} B \vdash C} \quad \text{(\otimes L)}
\]

Inversion stage: chain application of invertible rules

\[
\frac{\Gamma; A \mathbin{\oplus} B, C \mathbin{\oplus} D \vdash E}{\Gamma; (A \mathbin{\oplus} B) \mathbin{\otimes} (C \mathbin{\oplus} D) \vdash E}
\]
Inversion

A rule is *invertible* if its conclusion implies its premises

Every connective of LL is invertible on either left or right, e.g.:

\[
\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B} \quad (\&R) \quad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \quad (\otimes L)
\]

Inversion stage: chain application of invertible rules

\[
\frac{\Gamma; A, C \oplus D \vdash E \quad \Gamma; B, C \oplus D \vdash E}{\Gamma; (A \oplus B) \otimes (C \oplus D) \vdash E}
\]
Inversion

A rule is *invertible* if its conclusion implies its premises.

Every connective of LL is invertible on either left or right, e.g.:

\[
\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B} \quad (\&R) \quad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \quad (\otimes L)
\]

Inversion stage: chain application of invertible rules

\[
\frac{\Gamma; A, C \vdash E \quad \Gamma; A, D \vdash E}{\Gamma; A, C \oplus D \vdash E} \quad \frac{\Gamma; B, C \vdash E \quad \Gamma; B, C \vdash E}{\Gamma; B, C \oplus D \vdash E} \quad \frac{\Gamma; A \oplus B, C \oplus D \vdash E}{\Gamma; (A \oplus B) \otimes (C \oplus D) \vdash E}
\]
Inversion

A rule is *invertible* if its conclusion implies its premises

Every connective of LL is invertible on either left or right, e.g.:

\[
\frac{\Gamma \vdash A}{\Gamma \vdash A \& B} \quad (\&R) \quad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \quad (\otimes L)
\]

Inversion stage: chain application of invertible rules

... and treat as one big step:

\[
\begin{align*}
\Gamma, A, C & \vdash E \\
\Gamma, A, D & \vdash E \\
\Gamma, B, C & \vdash E \\
\Gamma, B, D & \vdash E \\
\hline
\Gamma; (A \oplus B) \otimes (C \oplus D) & \vdash E
\end{align*}
\]
Flashback

Two equivalent λ-calculus terms:

\[
\text{case}\left(\pi_1 x, y_1.\text{case}\left(\pi_2 x, z_1.e_1, z_2.e_2\right), y_2.\text{case}\left(\pi_2 x, z_1.e_3, z_2.e_4\right)\right)
\]

\[
\text{case}\left(\pi_2 x, z_1.\text{case}\left(\pi_1 x, y_1.e_1, y_2.e_3\right), z_2.\text{case}\left(\pi_1 x, y_1.e_2, y_2.e_4\right)\right)
\]

Same term in ML:

\[
\text{case } x \text{ of }
\begin{align*}
  \text{(Inl } y_1, \text{ Inl } z_1) & \Rightarrow e_1 \\
  \text{(Inl } y_1, \text{ Inr } z_2) & \Rightarrow e_2 \\
  \text{(Inr } y_2, \text{ Inl } z_1) & \Rightarrow e_3 \\
  \text{(Inr } y_2, \text{ Inr } z_2) & \Rightarrow e_4
\end{align*}
\]
Focus

Focus stage: chain application of non-invertible rules
Focus

Focus stage: chain application of non-invertible rules

\[ \Gamma \vdash [(A \oplus B) \otimes (C \oplus D)] \]
Focus

Focus stage: chain application of non-invertible rules

\[
\Gamma_1 \vdash [A \oplus B] \quad \Gamma_2 \vdash [C \oplus D] \\
\Gamma_1, \Gamma_2 \vdash [(A \oplus B) \otimes (C \oplus D)]
\]
Focus

Focus stage: chain application of non-invertible rules

\[
\Gamma_1 \vdash [B] \\
\Gamma_1 \vdash [A \oplus B] \quad \Gamma_2 \vdash [C \oplus D] \\
\Gamma_1, \Gamma_2 \vdash [(A \oplus B) \otimes (C \oplus D)]
\]
Focus

Focus stage: chain application of non-invertible rules

\[
\Gamma_1 \vdash [B] \\
\Gamma_1 \vdash [A \oplus B] \\
\Gamma_2 \vdash [C] \\
\Gamma_2 \vdash [C \oplus D] \\
\Gamma_1, \Gamma_2 \vdash [(A \oplus B) \otimes (C \oplus D)]
\]
Focus

Focus stage: chain application of non-invertible rules
... and treat as one big (nondeterministic) step:

\[
\Gamma_1 \vdash [B] \quad \Gamma_2 \vdash [C] \\
\Gamma_1, \Gamma_2 \vdash [(A \oplus B) \otimes (C \oplus D)]
\]
Focus

Focus stage: chain application of non-invertible rules
...and treat as one big (nondeterministic) step:

\[
\Gamma_1 \vdash [B] \quad \Gamma_2 \vdash [C]
\]

\[
\Gamma_1, \Gamma_2 \vdash [(A \oplus B) \otimes (C \oplus D)]
\]

Theorem (Andreoli): focusing proof-search is complete.
1. Inversion and focus are dual stages
2. They divide the connectives into two groups

Together these dualities form a *square of opposition*:

<table>
<thead>
<tr>
<th>polarity</th>
<th>inversion</th>
<th>focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>positive</td>
<td>$\otimes, \oplus, 1, 0$</td>
<td>left</td>
</tr>
<tr>
<td>negative</td>
<td>$&amp;, \emptyset, \top, \bot$</td>
<td>right</td>
</tr>
</tbody>
</table>
1. Inversion and focus are dual stages
2. They divide the connectives into two groups

Together these dualities form a *square of opposition*:

<table>
<thead>
<tr>
<th>polarity</th>
<th>inversion</th>
<th>focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>positive</td>
<td>$\otimes, \oplus, 1, 0$</td>
<td>left</td>
</tr>
<tr>
<td>negative</td>
<td>$&amp;, \otimes, \top, \bot$</td>
<td>right</td>
</tr>
</tbody>
</table>

Girard ’93: dualities extend beyond LL by *polarization*. . .
Duality$^2$

1. Inversion and focus are dual stages
2. They divide the connectives into two groups

Together these dualities form a square of opposition:

<table>
<thead>
<tr>
<th>polarity</th>
<th>inversion</th>
<th>focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>positive</td>
<td>$\otimes, \oplus, 1, 0$</td>
<td>left</td>
</tr>
<tr>
<td>negative</td>
<td>$&amp;, \exists, \top, \bot$</td>
<td>right</td>
</tr>
</tbody>
</table>

Girard '93: dualities extend beyond LL by polarization. . .
. . . i.e., by fixing evaluation order
What you’ll find in the paper

A new formulation of focusing for intuitionistic logic with . . .

• big-step inversion and focus through higher-order rules
• A “connective-blind” proof of cut-elimination!

A Curry-Howard interpretation incorporating . . .

• CBV functions, strict products and sums, recursive types
• Pattern-matching “for free” through a HO encoding:
  \[ \text{fnc} = \text{pat} \rightarrow \text{exp} \quad \text{val} = \text{pat} \times \text{sub} \]
• A connective-blind proof of type safety

A formalization in Coq
A very quick sketch
Connectives of the day

Polarized intuitionistic connectives

Positive: $\otimes$, $\oplus$, 1, 0 (strict products and sums)
Negative: $\rightarrow$ (CBV function space)
(Same entailments as IL, different proofs.)

Notation for polarized formulas:

\[
P, Q ::= X \mid P \otimes Q \mid P \oplus Q \mid 0 \mid 1 \mid N
\]

\[
N ::= P \rightarrow Q
\]

In paper: $P ::= \cdots \mid \mu X.P$

“Hidden” slides (not in paper): $N, M ::= \cdots \mid N \rightarrow M \mid N & M \mid \cdots$
Linear right rules \( \cong \) Pattern typing

Linear contexts \[ \Delta ::= \cdot | X, \Delta | N, \Delta \]

\[ \Delta \Rightarrow P \]

\[
\begin{array}{ll}
X \Rightarrow X & N \Rightarrow N \\
\Delta_1 \Rightarrow P & \Delta_2 \Rightarrow Q \\
\cdot \Rightarrow 1 & \Delta_1, \Delta_2 \Rightarrow P \otimes Q \\
\end{array}
\]

(no rule for 0)

\[
\begin{array}{ll}
\Delta \Rightarrow P & \Delta \Rightarrow Q \\
\Delta \Rightarrow P \oplus Q & \Delta \Rightarrow P \oplus Q \\
\end{array}
\]
Linear right rules $\simeq$ Pattern typing

Linear contexts $\Delta ::= \cdot \mid x : X, \Delta \mid f : N, \Delta$

\[
\Delta \Rightarrow P
\]

\[
\begin{align*}
X & \Rightarrow X \\
N & \Rightarrow N \\
\Delta_1 \Rightarrow P & \quad \Delta_2 \Rightarrow Q \\
\cdot \Rightarrow 1 & \quad \Delta_1, \Delta_2 \Rightarrow P \otimes Q
\end{align*}
\]

(no rule for 0)

\[
\begin{align*}
\Delta \Rightarrow P & \quad \Delta \Rightarrow Q \\
\Delta \Rightarrow P \oplus Q & \quad \Delta \Rightarrow P \oplus Q
\end{align*}
\]
Linear right rules $\approx$ Pattern typing

Linear contexts $\Delta ::= \cdot | x : X, \Delta | f : N, \Delta$

\[\Delta \Rightarrow P\]

\[
\begin{align*}
X & \Rightarrow X \quad N \Rightarrow N \\
\cdot & \Rightarrow 1 \\
\Delta_1 \Rightarrow P, \Delta_2 \Rightarrow Q & \quad \Rightarrow P \otimes Q \\
\Delta_1, \Delta_2 \Rightarrow P \oplus Q \\
\Delta \Rightarrow P & \quad \Delta \Rightarrow Q \\
\Delta \Rightarrow P \oplus Q & \quad \Delta \Rightarrow P \oplus Q
\end{align*}
\]

(no rule for 0)
Linear right rules \(\approx\) Pattern typing

Linear contexts

\[
\Delta ::= \cdot \mid x : X, \Delta \mid f : N, \Delta
\]

\[\Delta \Rightarrow p : P\]

\[
x : X \Rightarrow x : X \quad f : N \Rightarrow f : N
\]

\[
\cdot \Rightarrow () : 1
\]

\[
\Delta_1 \Rightarrow p_1 : P \quad \Delta_2 \Rightarrow p_2 : Q
\]

\[
\Delta_1, \Delta_2 \Rightarrow (p_1, p_2) : P \otimes Q
\]

(no rule for 0)

\[
\Delta \Rightarrow p : P
\]

\[
\Delta \Rightarrow \text{inl } p : P \oplus Q
\]

\[
\Delta \Rightarrow \text{inr } p : P \oplus Q
\]

Usual pattern restrictions emerge!
Focused [sequent/lambda]-calculus

Unrestricted contexts  \[ \Gamma ::= \cdot \mid \Gamma, \Delta \]

Judgments:

\[ \Gamma \vdash [P] \quad \text{right-focus} \]
\[ \Gamma; P \vdash Q \quad \text{left-inversion} \]
\[ \Gamma \vdash P \quad \text{unfocused} \]
\[ \Gamma \vdash \Delta \quad \text{multiple (conjoined) conclusions} \]
Focused [sequent/lambda]-calculus

Unrestricted contexts \( \Gamma ::= \cdot \mid \Gamma, \Delta \)

Judgments:

\( \Gamma \vdash [P] \)
\( \Gamma; P \vdash Q \)
\( \Gamma \vdash P \)
\( \Gamma \vdash \Delta \)
Focused [sequent/lambda]-calculus

Unrestricted contexts \( \Gamma ::= \cdot \mid \Gamma, \Delta \)

Judgments:

\[\begin{align*}
\Gamma &\vdash V : [P] & \text{value} \\
\Gamma; P &\vdash Q \\
\Gamma &\vdash P \\
\Gamma &\vdash \Delta
\end{align*}\]
Focused [sequent/lambda]-calculus

Unrestricted contexts  \[ \Gamma ::= \cdot \mid \Gamma, \Delta \]

Judgments:
\[ \Gamma \vdash V : [P] \quad \text{value} \]
\[ \Gamma \vdash F : P > Q \quad \text{(CBV) function} \]
\[ \Gamma \vdash P \]
\[ \Gamma \vdash \Delta \]
Focused [sequent/lambda]-calculus

Unrestricted contexts

\[ \Gamma ::= \cdot \mid \Gamma, \Delta \]

Judgments:

\[ \Gamma \vdash V : [P] \quad \text{value} \]
\[ \Gamma \vdash F : P > Q \quad \text{(CBV) function} \]
\[ \Gamma \vdash E : P \quad \text{expression} \]
\[ \Gamma \vdash \Delta \]
Focused [sequent/lambda]-calculus

Unrestricted contexts  \[ \Gamma ::= \cdot \mid \Gamma, \Delta \]

Judgments:

\[ \Gamma \vdash V : [P] \quad \text{value} \]
\[ \Gamma \vdash F : P > Q \quad \text{(CBV) function} \]
\[ \Gamma \vdash E : P \quad \text{expression} \]
\[ \Gamma \vdash \sigma : \Delta \quad \text{substitution} \]
Untyped syntax

Canonical fragment ($\beta$-reduced, $\eta$-long):

\[ V ::= p[\sigma] \]
\[ F ::= \lambda \phi \]
\[ \text{where } \phi ::= (p_1 \mapsto E_1 \mid \cdots \mid p_n \mapsto E_n) \]
\[ E ::= V \mid F(g(V)) \]
\[ \sigma ::= \cdot \mid (y/x, \sigma) \mid (F/f, \sigma) \]

Identity and cut principles:

\[ F ::= \cdots \mid \text{id} \mid f \]
\[ E ::= \cdots \mid F(V) \mid F(E) \]
Right-focus $\approx$ Value typing

$\Gamma \vdash [P]$

$\Delta \Rightarrow P \quad \Gamma \vdash \Delta$

$\Gamma \vdash [P]$
Right-focus $\approx$ Value typing

$\Gamma \vdash V : [P]$

$\Delta \Rightarrow P \quad \Gamma \vdash \Delta$

$\Gamma \vdash [P]$
Right-focus $\approx$ Value typing

$$\Gamma \vdash V : [P]$$

$$\Delta \Rightarrow p : P \quad \Gamma \vdash \sigma : \Delta$$

$$\Gamma \vdash p[\sigma] : [P]$$
Right-focus $\approx$ Value typing

$\Gamma \vdash V : [P]$

$\begin{array}{c}
\Delta \Rightarrow p : P \\
\Gamma \vdash \sigma : \Delta
\end{array}$

$\Gamma \vdash p[\sigma] : [P]$

Yes, this is really a “value” in the ML sense…

$(\text{fn } x \Rightarrow x \times x, \text{fn } x \Rightarrow x - 3)$
Right-focus $\approx$ Value typing

\[ \Gamma \vdash V : [P] \]

\[ \frac{\Delta \Rightarrow p : P \quad \Gamma \vdash \sigma : \Delta}{\Gamma \vdash p[\sigma] : [P]} \]

Yes, this is really a “value” in the ML sense…

\[ (f, g)[(fn \ x \Rightarrow x*x)/f, (fn \ x \Rightarrow x-3)/g] \]
Left-inversion \approx \text{CBV function typing}

\[ \Gamma; P \vdash Q \]

\[ \forall (\Delta \Rightarrow P) : \Gamma, \Delta \vdash Q \]

\[ \Gamma; P \vdash Q \]
Left-inversion $\approx$ CBV function typing

\[
\Gamma \vdash F : P > Q
\]

\[
\forall (\Delta \Rightarrow P) : \Gamma, \Delta \vdash Q \\
\therefore \Gamma; P \vdash Q
\]
Left-inversion \approx \text{CBV function typing}

\[ \Gamma \vdash F : P > Q \]

\[ \forall (\Delta \Rightarrow p : P) : \quad \Gamma, \Delta \vdash \phi(p) : Q \]

\[ \Gamma \vdash (\lambda \phi) : P > Q \]
Left-inversion $\approx$ CBV function typing

$$\Gamma \vdash F : P > Q$$

$$\forall (\Delta \Rightarrow p : P) : \Gamma, \Delta \vdash \phi(p) : Q$$

$$\Gamma \vdash (\lambda \phi) : P > Q$$

$\phi$ a (partial) map from patterns to expressions!

In Coq, define $\text{Lam} : (\text{pat} \rightarrow \text{exp}) \rightarrow \text{fnc}$
Left-inversion $\approx$ CBV function typing

$$\forall (\Delta \Rightarrow p : P) : \Gamma, \Delta \vdash \phi(p) : Q$$

$$\Gamma \vdash (\lambda \phi) : P > Q$$

$\phi$ a (partial) map from patterns to expressions!

In Coq, define $\text{Lam} : (\text{pat} \rightarrow \text{exp}) \rightarrow \text{fnc}$

Hmm...?!
A more abstract syntax

The basic premise of this higher-order encoding is that the static and dynamic semantics of a language shouldn’t care so much about the details of how a function is written down, but just get to the point about what expression is evaluated for each pattern. (Note this is not the same as only caring about the mapping from values to results!)

Several people have complained that calling this sort of encoding “higher-order abstract syntax” is misleading, since HOAS has a well-established usage referring to meta-level substitution functions used to encode object-level binders. They are probably right that my choice of terminology was confusing—sorry! Maybe it should be “abstract higher-order syntax”?…
Small-step semantics as cut-elimination

\[
\text{id}(V) \leadsto V
\]

\[
\frac{\phi(p) \text{ defined}}{(\lambda \phi)(p[\sigma]) \leadsto \phi(p)[\sigma]}
\]

\[
\frac{E \leadsto E'}{F(E) \leadsto F(E')}
\]

Theorem: Type Safety
Proof does not mention (positive) connectives!

Cf. Gentzen/Prawitz/Dummett proof-theoretic semantics
Coq encoding

“Curry-style” (untyped syntax, type system, opsem, type safety):

http://www.cs.cmu.edu/~noam/research/focusing.tar

“Church-style” (typed syntax only):

http://www.cs.cmu.edu/~noam/research/focus-church.v

“Church v2.0” (with uniform treatment of negative types):

http://www.cs.cmu.edu/~noam/research/focusing2.v

Define functions using Coq’s built-in pattern-matching!

(But use de Bruijn indices for binding, alas...)
Coq example: plus

Definition plus : fnc :=
  Fix (Lam (fun p => match p with
       | Pair m Ze => Return (Val m (Sub nil))
       | Pair m (Su n) =>
          EComp
            (Lam (fun n’ =>
               Return (Val (Su n’)
                           (Sub nil))))
               (1,0) (Val (Pair m n) (Sub nil))
       | _ => EFail
   end)).
Conclusions
Focusing is awesome!
Focusing is awesome!

Why?

1. Pattern-matching $\approx$ focus and inversion
2. Explicit evaluation order $\approx$ polarity
Future Work

The computational world of polarized logic & type theory

- Mixed polarities = mixed evaluation strategies (Haskell?)
- Polymorphism & intersection/union types
- Dependent types

Higher-order encodings

- What is the relationship between AHOS and HOAS?
- Can they be combined in one logical framework?
- See Brigitte’s talk! And paper with Licata & Harper…
Thanks!
Recursion and recursive types

Recursion:

\[ \Gamma, f : P \rightarrow Q \vdash F : P > Q \]

\[ \Gamma \vdash \text{fix} f.F : P > Q \]

\[ (\text{fix} f.F)(V) \rightsquigarrow ([\text{fix} f.F/f]F)(V) \]

Recursive types:

\[ \Delta \Rightarrow p : P[\mu X.P/X] \]

\[ \Delta \Rightarrow \text{fold}(p) : \mu X.P \]
A uniform treatment of negative types

Make coercions explicit: \( \downarrow N \) positive, \( \uparrow P \) negative

\[
P \rightarrow Q = P \rightarrow \uparrow Q \quad N \rightarrow m \rightarrow M = \downarrow N \rightarrow M
\]

Define negative connectives through linear left rules:

\[
\Delta; N \Rightarrow \gamma
\]

\[
\frac{\Delta_1 \Rightarrow P \quad \Delta_2; N \Rightarrow \gamma}{\Delta_1, \Delta_2; P \rightarrow N \Rightarrow \gamma}
\]

\[
\frac{\Delta; N \Rightarrow \gamma \quad \Delta; M \Rightarrow \gamma}{\Delta; N \& M \Rightarrow \gamma}
\]

Curry-Howard: destructor patterns
General intuitionistic focusing

Right-focus and left-inversion:

\[ \frac{\Delta \Rightarrow P}{\Gamma \vdash \Delta} \quad \frac{\forall (\Delta \Rightarrow P)}{\Gamma, \Delta \vdash \gamma} \]

\[ \frac{\Gamma \vdash [P]}{\Gamma \vdash P} \quad \frac{\Gamma; P \vdash \gamma}{\Gamma; [P] \vdash \gamma} \]

Right-inversion and left-focus:

\[ \frac{\forall (\Delta; N \Rightarrow \gamma)}{\Gamma, \Delta \vdash \gamma} \quad \frac{\Delta; N \Rightarrow \gamma_0}{\Gamma \vdash \Delta} \quad \frac{\Gamma; \gamma_0 \vdash \gamma}{\Gamma; [N] \vdash \gamma} \]

Unfocused sequents:

\[ \frac{\Gamma \vdash [P]}{\Gamma \vdash P} \quad \frac{N \in \Gamma}{\Gamma; [N] \vdash \gamma} \quad \frac{\Gamma; \gamma \vdash \gamma}{\Gamma \vdash \gamma} \]