Monads in ML

A monad in ML is given by type $\tau(\alpha)$ with a free type variable α together with expressions

$$unit: lpha o au(lpha) \ lift: (lpha o au(eta)) o au(lpha) o au(eta)$$

(writing $\tau(\beta)$ for the result of replacing α by β in τ) satisfying some equational properties [omitted].

PLC versus the Pure Type System $\lambda 2$

PTS signature:

$$2 \triangleq (\mathcal{S}_2, \mathcal{A}_2, \mathcal{R}_2) ext{ where } \left\{ egin{array}{ccc} \mathcal{S}_2 & riangle & \{*, \Box\} \ \mathcal{A}_2 & riangle & \{(*, \Box)\} \ \mathcal{R}_2 & riangle & \{(*, *, *), (\Box, *, *)\} \end{array}
ight.$$

Translation of PLC types and terms to $\lambda 2$ pseudo-terms:

$$\begin{bmatrix} \alpha \end{bmatrix} = \alpha$$

$$\begin{bmatrix} \tau \to \tau' \end{bmatrix} = \Pi x : \begin{bmatrix} \tau \end{bmatrix} (\begin{bmatrix} \tau' \end{bmatrix})$$

$$\begin{bmatrix} \forall \alpha (\tau) \end{bmatrix} = \Pi \alpha : * (\begin{bmatrix} \tau' \end{bmatrix})$$

$$\begin{bmatrix} x \end{bmatrix} = x$$

$$\begin{bmatrix} \lambda x : \tau (M) \end{bmatrix} = \lambda x : \begin{bmatrix} \tau \end{bmatrix} (\llbracket M \rrbracket)$$

$$\begin{bmatrix} M M' \end{bmatrix} = \llbracket M \rrbracket \llbracket M' \rrbracket$$

$$\begin{bmatrix} M \alpha (M) \end{bmatrix} = \lambda \alpha : * (\llbracket M \rrbracket)$$

$$\begin{bmatrix} M \tau \rrbracket = \llbracket M \rrbracket \llbracket \tau \rrbracket$$

PTS specification $\omega = (\mathcal{S}_{\omega}, \mathcal{A}_{\omega}, \mathcal{R}_{\omega})$:

$$egin{aligned} \mathcal{S}_{\omega} & & \triangleq \{*, \Box\} \ \mathcal{A} & & \triangleq \{(*, \Box)\} \ \mathcal{R} & & \triangleq \{(*, *, *), (\Box, *, *), (\Box, \Box, \Box)\} \end{aligned}$$

PTS specification $\omega = (S_{\omega}, A_{\omega}, \mathcal{R}_{\omega})$:

$$egin{aligned} \mathcal{S}_{\omega} & & \triangleq \{*, \Box\} \ \mathcal{A} & & \triangleq \{(*, \Box)\} \ \mathcal{R} & & \triangleq \{(*, *, *), (\Box, *, *), (\Box, \Box, \Box)\} \end{aligned}$$

As in $\lambda 2$, sort * is a universe of types; but in $\lambda \omega$, the rule (**prod**) for (\Box, \Box, \Box) means that $\diamond \vdash t : \Box$ holds for all the 'simple types' over the ground type * – the ts generated by the grammar $t := * \mid t \to t$

$$(\text{prob}) \xrightarrow{\Gamma + A: \Box} \quad (\overline{J}, J: A + B: \Box) \quad for (\overline{J}, \overline{J}, \overline{\Box}) \quad for (\overline{J}, \overline{\Box}, \overline{\Box})$$

PTS specification $\omega = (\mathcal{S}_{\omega}, \mathcal{A}_{\omega}, \mathcal{R}_{\omega})$:

$$egin{aligned} \mathcal{S}_{\omega} &\triangleq \{*, \Box\} \ \mathcal{A} &\triangleq \{(*, \Box)\} \ \mathcal{R} &\triangleq \{(*, *, *), (\Box, *, *), (\Box, \Box, \Box)\} \end{aligned}$$

As in $\lambda 2$, sort * is a universe of types; but in $\lambda \omega$, the rule (**prod**) for (\Box, \Box, \Box) means that $\diamond \vdash t : \Box$ holds for all the 'simple types' over the ground type * – the ts generated by the grammar $t := * \mid t \to t$

$$(\text{prob}) \xrightarrow{\Gamma + A:\Box} \quad (\neg, \neg L:A + B:\Box) \quad \text{for} (\Box, \Box, \Box) \quad \text{for} (\Box, \Box, \Box) \quad \text{for} (\Box, \Box, \Box) \quad (A \rightarrow B \stackrel{2}{=} TT_{X:A}(B) \text{ with } x \notin f_{Y}(B))$$

PTS specification $\omega = (\mathcal{S}_{\omega}, \mathcal{A}_{\omega}, \mathcal{R}_{\omega})$:

$$egin{aligned} \mathcal{S}_{\omega} &\triangleq \{*, \Box\} \ \mathcal{A} &\triangleq \{(*, \Box)\} \ \mathcal{R} &\triangleq \{(*, *, *), (\Box, *, *), (\Box, \Box, \Box)\} \end{aligned}$$

As in $\lambda 2$, sort * is a universe of types; but in $\lambda \omega$, the rule (**prod**) for (\Box, \Box, \Box) means that $\diamond \vdash t : \Box$ holds for all the 'simple types' over the ground type * – the *t*s generated by the grammar $t ::= * \mid t \to t$ Hence rule (**prod**) for $(\Box, *, *)$ now gives many more legal pseudo-terms of type * in $\lambda \omega$ compared with $\lambda 2$ (PLC), such as

$$\begin{array}{l} \diamond \vdash (\Pi T : * \to * (\Pi \alpha : * (\alpha \to T \alpha))) : * \\ \diamond \vdash (\Pi T : * \to * (\Pi \alpha, \beta : * ((\alpha \to T \beta) \to T \alpha \to T \beta))) : * \\ types for unit & lift operations, making T a monal \\ \end{array}$$

► Monad transformer for state (using a type ◇ ⊢ S : * for states):

$$\mathbb{M} \triangleq \lambda T : * \to * (\lambda \alpha : * (S \to T(\mathbb{P} \alpha S))) \\ \diamond \vdash \mathbb{M} : (* \to *) \to * \to *$$

- Product types (cf. the PLC representation of product types):
 - $\mathsf{P} \triangleq \lambda \alpha, \beta : \ast (\Pi \gamma : \ast ((\alpha \rightarrow \beta \rightarrow \gamma) \rightarrow \gamma))$

$$\Diamond \vdash P : \ast \to \ast \to \ast$$

► Monad transformer for state (using a type ◇ ⊢ S : * for states):

$$\mathbb{M} \triangleq \lambda T : * \to * (\lambda \alpha : * (S \to T(P\alpha S))) \\ \diamond \vdash \mathbb{M} : (* \to *) \to * \to *$$

- Product types (cf. the PLC representation of product types):
 - $P \triangleq \lambda \alpha, \beta : * (\Pi \gamma : * ((\alpha \to \beta \to \gamma) \to \gamma)) \\ \diamond \vdash P : * \to * \to *$

 $T \times \tau' \stackrel{a}{=} \forall \gamma ((\tau \rightarrow \tau' \rightarrow \gamma) \rightarrow \gamma)$ where $\gamma \notin ftv(\tau, \tau')$ (one definition per each choice of types $\tau \& \tau'$)

• Product types (cf. the PLC representation of product types): $P \triangleq \lambda \alpha, \beta : * (\Pi \gamma : * ((\alpha \rightarrow \beta \rightarrow \gamma) \rightarrow \gamma))$ $\diamond \vdash P : * \rightarrow * \rightarrow *$

► Mon
state
$$\exists \alpha(\tau) \stackrel{A}{=} \forall \beta((\forall \alpha(\tau \rightarrow \beta)) \rightarrow \beta)$$

where $\beta \notin ftv(\tau)$

Existential types (cf. the PLC representation of existential types):

 $\exists \triangleq \lambda T : * \to * (\Pi \beta : * ((\Pi \alpha : * (T \alpha \to \beta)) \to \beta)) \\ \diamond \vdash \exists : (* \to *) \to *$

Type-checking for \mathbf{F}_{ω} ($\lambda \omega$)

Fact (Girard): System F_{ω} is *strongly normalizing* in the sense that for any legal pseudo-term t, there is no infinite chain of beta-reductions $t \to t_1 \to t_2 \to \cdots$.

Type-checking for \mathbf{F}_{ω} ($\lambda_{\mathbb{N}}$)

 $(\times \omega)$

Fact (Girard): System F_{ω} is *strongly normalizing* in the sense that for any legal pseudo-term t, there is no infinite chain of beta-reductions $t \rightarrow t_1 \rightarrow t_2 \rightarrow \cdots$.

As as corollary we have that the type-checking and typeability problems for F_{ω} are decidable.

Propositions as Types (sect. 6 of notes)

Logic	\leftrightarrow	Type system
propositions ϕ	\leftrightarrow	types $ au$
proofs <i>p</i>	\leftrightarrow	expressions M
' p is a proof of ϕ '	\leftrightarrow	' M is an expression of type $oldsymbol{ au}$ '
simplification of proofs	\leftrightarrow	reduction of expressions
First anose for	Cor	nstructive logics

Constructive interpretation of logic

- Implication: a proof of $\varphi \rightarrow \psi$ is a construction that transforms proofs of φ into proofs of ψ .
- Negation: a proof of ¬φ is a construction that from any (hypothetical) proof of φ produces a contradiction (= proof of falsity ⊥)
- **Disjunction:** a proof of $\varphi \lor \psi$ is an object that manifestly is either a proof of φ , or a proof of ψ .
- For all: a proof of ∀x (φ(x)) is a construction that transforms the objects a over which x ranges into proofs of φ(a).
- There exists: a proof of $\exists x (\varphi(x))$ is given by a pair consisting of an object a and a proof of $\varphi(a)$.

Constructive interpretation of logic

- Implication: a proof of $\varphi \rightarrow \psi$ is a construction that transforms proofs of φ into proofs of ψ .
- ► Negation: a proof of ¬φ is a construction that from any (hypothetical) proof of φ produces a contradiction (= proof of falsity ⊥)
- **Disjunction:** a proof of $\varphi \lor \psi$ is an object that manifestly is either a proof of φ , or a proof of ψ .
- For all: a proof of ∀x (φ(x)) is a construction that transforms the objects a over which x ranges into proofs of φ(a).
- There exists: a proof of $\exists x (\varphi(x))$ is given by a pair consisting of an object a and a proof of $\varphi(a)$.

The *Law of Excluded Middle* (LEM) $\forall p (p \lor \neg p)$ is a classical tautology (has truth-value true), but is rejected by constructivists.

Theorem. There exist two irrational numbers a and b such that b^a is rational.

Theorem. There exist two irrational numbers a and b such that b^a is rational.

Proof. Either $\sqrt{2^{\sqrt{2}}}$ is rational, or it is not (LEM!).

Theorem. There exist two irrational numbers a and b such that b^a is rational.

Proof. Either $\sqrt{2^{\sqrt{2}}}$ is rational, or it is not (LEM!).

If it is, we can take $a = b = \sqrt{2}$, since $\sqrt{2}$ is irrational by a well-known theorem attributed to Euclid.

Theorem. There exist two irrational numbers a and b such that b^a is rational.

Proof. Either $\sqrt{2^{\sqrt{2}}}$ is rational, or it is not (LEM!).

If it is, we can take $a = b = \sqrt{2}$, since $\sqrt{2}$ is irrational by a well-known theorem attributed to Euclid.

If it is not, we can take $a = \sqrt{2}$ and $b = \sqrt{2^{\sqrt{2}}}$, since then $b^a = (\sqrt{2^{\sqrt{2}}})^{\sqrt{2}} = \sqrt{2^{\sqrt{2} \cdot \sqrt{2}}} = \sqrt{2^2} = 2$.

QED

Theorem. There exist two irrational numbers a and b such that b^a is rational.

Proof. $\sqrt{2}$ is irrational by a well-known constructive proof attributed to Euclid.

 $2\log_{2} 3 \text{ is irrational, by an easy constructive proof (exercise).}$ $(IF 2\log_{2} 3 = m_{h}, then$ $3^{2n} = 2^{2n\log_{2} 3} = 2^{m} \times)$

Theorem. There exist two irrational numbers a and b such that b^a is rational.

Proof. $\sqrt{2}$ is irrational by a well-known constructive proof attributed to Euclid.

 $2\log_2 3$ is irrational, by an easy constructive proof (exercise).

So we can take $a = 2\log_2 3$ and $b = \sqrt{2}$, for which we have that $b^a = (\sqrt{2})^{2\log_2 3} = (\sqrt{2}^2)^{\log_2 3} = 2^{\log_2 3} = 3$ is rational.

QED

Curry-Howard correspondence				
	$\langle - \rangle$	PL C		
Logic	\leftrightarrow	Type system		
propositions ϕ	\leftrightarrow	types $ au$		
proofs <i>p</i>	\leftrightarrow	expressions M		
' p is a proof of ϕ '	\leftrightarrow	' M is an expression of type $ au$ '		
simplification of proofs	\leftrightarrow	reduction of expressions		
	E.g.			
2IPC	\leftrightarrow	PLC		

Second-order intuitionistic propositional calculus (2IPC)

2IPC propositions: $\phi ::= p | \phi \rightarrow \phi | \forall p (\phi)$ where p ranges over an infinite set of propositional variables.

2IPC sequents: $\Phi \vdash \phi$ where Φ is a finite multiset (= unordered list) of 2IPC propositions and ϕ is a 2IPC proposition.

Second-order intuitionistic propositional calculus (2IPC)

2IPC propositions: $\phi := p | \phi \rightarrow \phi | \forall p(\phi)$ where p ranges over an infinite set of propositional variables.

2IPC sequents: $\Phi \vdash \phi$ where Φ is a finite multiset (= unordered list) of 2IPC propositions and ϕ is a 2IPC proposition.

 $\Phi \vdash \phi$ is *provable* if it is in the set of sequents inductively generated by:

 $(Id) \Phi \vdash \phi \quad \text{if } \phi \in \Phi$ $(\rightarrow I) \frac{\Phi, \phi \vdash \phi'}{\Phi \vdash \phi \rightarrow \phi'} \qquad (\rightarrow E) \frac{\Phi \vdash \phi \rightarrow \phi' \quad \Phi \vdash \phi}{\Gamma \vdash \phi'}$ $(\forall I) \frac{\Phi \vdash \phi}{\Phi \vdash \forall p(\phi)} \quad \text{if } p \notin fv(\Phi) \qquad (\forall E) \frac{\Phi \vdash \forall p(\phi)}{\Phi \vdash \phi[\phi'/p]}$

- Truth $\top \triangleq \forall p \ (p \rightarrow p)$
- Falsity $\bot \triangleq \forall p(p)$

- Truth $\top \triangleq \forall p \ (p \rightarrow p)$
- Falsity $\bot \triangleq \forall p(p)$
- Conjunction $\phi \land \psi \triangleq \forall p ((\phi \rightarrow \psi \rightarrow p) \rightarrow p)$ (where $p \notin fv(\phi, \psi)$)

- Truth $\top \triangleq \forall p \ (p \rightarrow p)$
- Falsity $\bot \triangleq \forall p(p)$
- Conjunction $\phi \land \psi \triangleq \forall p ((\phi \rightarrow \psi \rightarrow p) \rightarrow p)$ (where $p \notin fv(\phi, \psi)$)
- Disjunction $\phi \lor \psi \triangleq \forall p ((\phi \rightarrow p) \rightarrow (\psi \rightarrow p) \rightarrow p)$ (where $p \notin fv(\phi, \psi)$)

- Truth $\top \triangleq \forall p \ (p \rightarrow p)$
- ► Falsity $\bot \triangleq \forall p(p)$
- Conjunction $\phi \land \psi \triangleq \forall p ((\phi \rightarrow \psi \rightarrow p) \rightarrow p)$ (where $p \notin fv(\phi, \psi)$)
- *Disjunction* $\phi \lor \psi \triangleq \forall p ((\phi \rightarrow p) \rightarrow (\psi \rightarrow p) \rightarrow p)$ (where $p \notin fv(\phi, \psi)$)
- Negation $\neg \phi \triangleq \phi \rightarrow \bot$
- ► Bi-implication $\phi \leftrightarrow \psi \triangleq (\phi \rightarrow \psi) \land (\psi \rightarrow \phi)$

- Truth $\top \triangleq \forall p \ (p \rightarrow p)$
- ► Falsity $\bot \triangleq \forall p(p)$
- Conjunction $\phi \land \psi \triangleq \forall p ((\phi \rightarrow \psi \rightarrow p) \rightarrow p)$ (where $p \notin fv(\phi, \psi)$)
- *Disjunction* $\phi \lor \psi \triangleq \forall p ((\phi \rightarrow p) \rightarrow (\psi \rightarrow p) \rightarrow p)$ (where $p \notin fv(\phi, \psi)$)
- Negation $\neg \phi \triangleq \phi \rightarrow \bot$
- ► Bi-implication $\phi \leftrightarrow \psi \triangleq (\phi \rightarrow \psi) \land (\psi \rightarrow \phi)$
- Existential quantification $\exists p(\phi) \triangleq \forall q (\forall p(\phi \rightarrow q) \rightarrow q)$ (where $q \notin fv(\phi, p)$)

A 2IPC proof

Writing $p \wedge q$ as an abbreviation for $\forall r ((p \rightarrow q \rightarrow r) \rightarrow r)$, the sequent

 $\{\} \vdash \forall p (\forall q ((p \land q) \rightarrow p))$

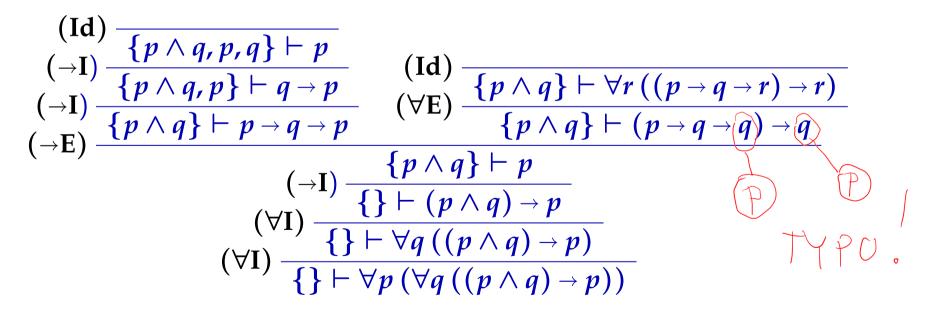
is provable in 2IPC:

A 2IPC proof

Writing $p \land q$ as an abbreviation for $\forall r ((p \rightarrow q \rightarrow r) \rightarrow r)$, the sequent

$$\{\} \vdash \forall p (\forall q ((p \land q) \to p))$$

is provable in 2IPC:

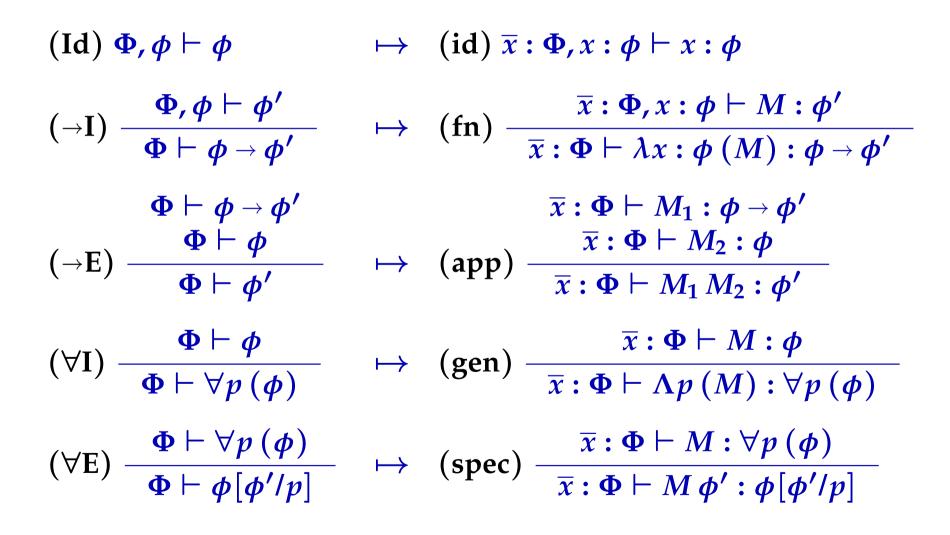






2IPC		PLC
Logic	\leftrightarrow	Type system
propositions ϕ	\leftrightarrow	types $ au$
proofs <i>p</i>	\leftrightarrow	expressions M
' p is a proof of ϕ '	\leftrightarrow	' M is an expression of type $ au$ '

Mapping 2IPC proofs to PLC expressions



The proof of the 2IPC sequent

$\{\} \vdash \forall p (\forall q ((p \land q) \rightarrow p))$

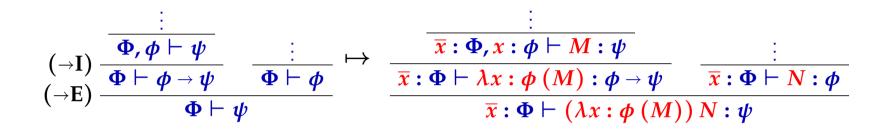
given before is transformed by the mapping of 2IPC proofs to PLC expressions to

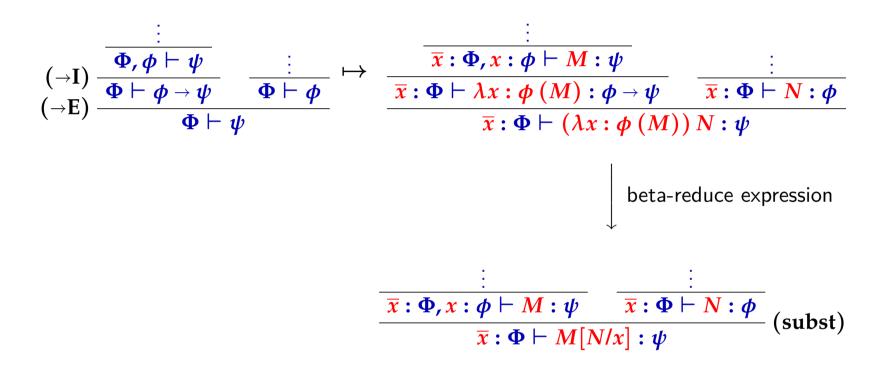
$$\{\} \vdash \Lambda p, q (\lambda z : p \land q (z p (\lambda x : p, y : q (x)))) \\ : \forall p (\forall q ((p \land q) \rightarrow p))$$

with typing derivation:

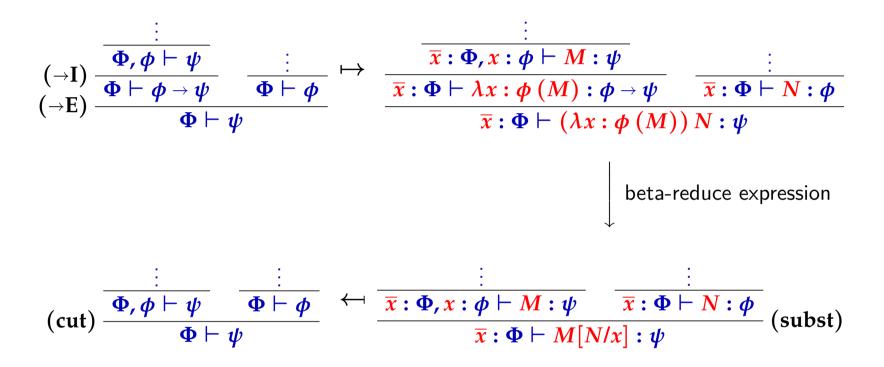
$$(id) \frac{\overline{\{z:p \land q, x: p, y:q\} \vdash x:p}}{\{z:p \land q, x:p\} \vdash \lambda y:q(x):q \rightarrow p} \quad (id) \frac{\overline{\{z:p \land q\} \vdash z: \forall r((p \rightarrow q \rightarrow r) \rightarrow r)}}{\{z:p \land q\} \vdash \lambda x:p, y:q(x):p \rightarrow q \rightarrow p} \quad (spec) \frac{(id)}{\{z:p \land q\} \vdash z:\forall r((p \rightarrow q \rightarrow r) \rightarrow r)}}{\{z:p \land q\} \vdash zp:(p \rightarrow q \rightarrow p) \rightarrow p} \\ (app) \frac{(fn)}{\{\} \vdash \lambda z:p \land q\} \vdash zp(\lambda x:p, y:q(x)):p}}{\{gen) \frac{(fn)}{\{\} \vdash \Lambda q(\lambda z:p \land q(zp(\lambda x:p, y:q(x))):(p \land q) \rightarrow p)}}{\{\} \vdash \Lambda p, q(\lambda z:p \land q(zp(\lambda x:p, y:q(x)))):\forall p, q((p \land q) \rightarrow p)}}$$

Logic	\leftrightarrow	Type system
propositions ϕ	\leftrightarrow	types $ au$
proofs <i>p</i>	\leftrightarrow	expressions M
' <i>p</i> is a proof of <i>φ</i> '	\leftrightarrow	' M is an expression of type $ au$ '
simplification of proofs	\leftrightarrow	reduction of expressions

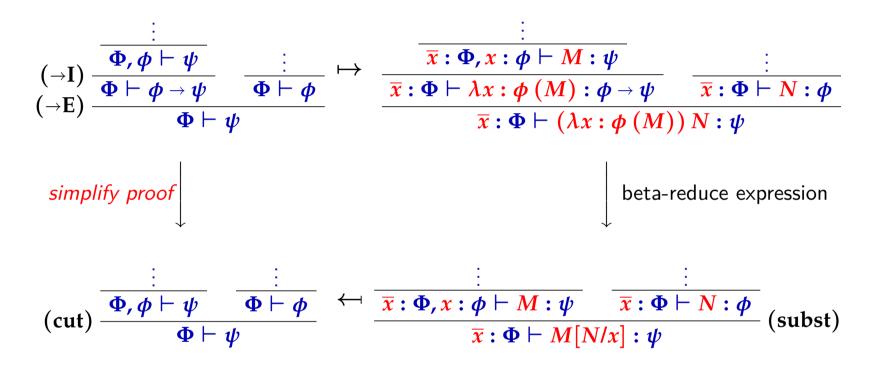




The rule (**subst**) for PLC is *admissible*: if its hypotheses are valid PLC typing judgements, then so is its conclusion.



The rule (**subst**) for PLC is *admissible*: if its hypotheses are valid PLC typing judgements, then so is its conclusion.



The rule (**subst**) for PLC is *admissible*: if its hypotheses are valid PLC typing judgements, then so is its conclusion.

Hence, the rule (cut) is admissible for 2IPC.