The Curry-Howard Isomorphism

Akim Demaille `akim@lrde.epita.fr`

EPITA — École Pour l'Informatique et les Techniques Avancées

June 14, 2016

The Curry-Howard Isomorphism

Heyting's Semantics of Proofs

Functional Interpretation

[Girard, 2004, Chap. 5]

- Instead of "when is a sentence A true"
- ask "what is a proof of A"?

Also named BHK (Brouwer–Heyting–Kolmogorov) interpretation.

Functional Interpretation

[Girard et al., 1989, Sec. 1.2.2]

What is a proof π of A? ($\pi \vdash A$)

$A \wedge B$ A pair (π_A, π_B) s.t. $\pi_A \vdash A$ and $\pi_B \vdash B$

Functional Interpretation
[Girard et al., 1989, Sec. 1.2.2]

What is a proof π of A? ($\pi \vdash A$)

$A \wedge B$	A pair (π_A, π_B) s.t. $\pi_A \vdash A$ and $\pi_B \vdash B$
$A \vee B$	A pair (i, π) s.t.
	$i = 0$ $\pi \vdash A$
	$i = 1$ $\pi \vdash B$

Functional Interpretation
[Girard et al., 1989, Sec. 1.2.2]

What is a proof π of A? ($\pi \vdash A$)

$A \wedge B$	A pair (π_A, π_B) s.t. $\pi_A \vdash A$ and $\pi_B \vdash B$
$A \vee B$	A pair (i, π) s.t.
	$i = 0$ $\pi \vdash A$
	$i = 1$ $\pi \vdash B$
$A \Rightarrow B$	A function f s.t. if $\pi \vdash A$ then $f(\pi) \vdash B$

Functional Interpretation

[Girard et al., 1989, Sec. 1.2.2]

What is a proof π of A? ($\pi \vdash A$)

$A \wedge B$	A pair (π_A, π_B) s.t. $\pi_A \vdash A$ and $\pi_B \vdash B$
$A \vee B$	A pair (i, π) s.t. $i = 0 \quad \pi \vdash A$; $i = 1 \quad \pi \vdash B$
$A \Rightarrow B$	A function f s.t. if $\pi \vdash A$ then $f(\pi) \vdash B$
$\forall x \cdot A$	A function f s.t. $f(a) \vdash A[a/x]$

Functional Interpretation

[Girard et al., 1989, Sec. 1.2.2]

What is a proof π of A? ($\pi \vdash A$)

$A \wedge B$	A pair (π_A, π_B) s.t. $\pi_A \vdash A$ and $\pi_B \vdash B$
$A \vee B$	A pair (i, π) s.t. $i = 0$ $\pi \vdash A$; $i = 1$ $\pi \vdash B$
$A \Rightarrow B$	A function f s.t. if $\pi \vdash A$ then $f(\pi) \vdash B$
$\forall x \cdot A$	A function f s.t. $f(a) \vdash A[a/x]$
$\exists x \cdot A$	A pair (a, π) s.t. $\pi \vdash A[a/x]$

Functional Interpretation
[Girard et al., 1989, Sec. 1.2.2]

What is a proof π of A? ($\pi \vdash A$)

Atomic Values	Assume we know what a proof is
$A \wedge B$	A pair (π_A, π_B) s.t. $\pi_A \vdash A$ and $\pi_B \vdash B$
$A \vee B$	A pair (i, π) s.t. $i = 0$ $\pi \vdash A$; $i = 1$ $\pi \vdash B$
$A \Rightarrow B$	A function f s.t. if $\pi \vdash A$ then $f(\pi) \vdash B$
$\forall x \cdot A$	A function f s.t. $f(a) \vdash A[a/x]$
$\exists x \cdot A$	A pair (a, π) s.t. $\pi \vdash A[a/x]$

Heyting's Semantics of Proofs

- An informal interpretation

Heyting's Semantics of Proofs

- An informal interpretation
- The handling of $\vee$ and $\exists$ are similar to the disjunctive and existential properties

Heyting's Semantics of Proofs

- An informal interpretation
- The handling of $\vee$ and $\exists$ are similar to the disjunctive and existential properties
- Therefore refers to a cut-free proof

Heyting's Semantics of Proofs

- An informal interpretation
- The handling of $\vee$ and $\exists$ are similar to the disjunctive and existential properties
- Therefore refers to a cut-free proof
- For instance id is a proof of $A \Rightarrow A$

The Curry-Howard Isomorphism

A Striking Correspondence

Type Derivations

$$\dfrac{M : \sigma \to \tau \quad N : \sigma}{MN : \tau}\ \mathrm{app}$$

$$\dfrac{\begin{array}{c}[x : \sigma] \\ \vdots \\ M : \tau\end{array}}{\lambda x \cdot M : \sigma \to \tau}\ \mathrm{abs}$$

Natural Deduction

$$\dfrac{A \Rightarrow B \quad A}{B} \Rightarrow\mathcal{E}$$

$$\dfrac{\begin{array}{c}[A] \\ \vdots \\ B\end{array}}{A \Rightarrow B} \Rightarrow\mathcal{I}$$

A Striking Correspondence

Type Derivations

$$\frac{M : \sigma \to \tau \quad N : \sigma}{MN : \tau}\,\mathrm{app}$$

$$\frac{\begin{array}{c}[x : \sigma]\\ \vdots \\ M : \tau\end{array}}{\lambda x \cdot M : \sigma \to \tau}\,\mathrm{abs}$$

Natural Deduction

$$\frac{A \Rightarrow B \quad A}{B} \Rightarrow\mathcal{E}$$

$$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \Rightarrow B} \Rightarrow\mathcal{I}$$

The Isomorphism

The Isomorphism

[Girard et al., 1989]

Curry-Howard Isomorphism

There is a perfect equivalence between two viewpoints:

Natural Deduction
: Formulas A, deductions of A, normalization in natural deduction

Typed λ-calculus
: Types A, terms of type A, normalization in λ-calculus.

Deductions and Terms: Conjuction

$$A^i$$

$$\dfrac{\begin{matrix}\vdots & \quad & \vdots \\ A & \quad & B\end{matrix}}{A \wedge B}\ \wedge\mathcal{I}$$

$$\vdots$$

$$\dfrac{A \wedge B}{A}\ \wedge 1\mathcal{E}$$

$$\vdots$$

$$\dfrac{A \wedge B}{B}\ \wedge 2\mathcal{E}$$

Deductions and Terms: Conjuction

$$x_i^A : A^i$$

$$\dfrac{\begin{array}{c}\vdots \\ u : A\end{array} \qquad \begin{array}{c}\vdots \\ v : B\end{array}}{\langle u, v\rangle : A \wedge B} \wedge\mathcal{I}$$

$$\dfrac{\begin{array}{c}\vdots \\ u : A \wedge B\end{array}}{\pi_1 u : A} \wedge 1\mathcal{E}$$

$$\dfrac{\begin{array}{c}\vdots \\ u : A \wedge B\end{array}}{\pi_2 u : B} \wedge 2\mathcal{E}$$

Reductions

$$\dfrac{\dfrac{\begin{matrix}\vdots & \qquad & \vdots\\ A & & B\end{matrix}}{A \wedge B}\wedge\mathcal{I}}{\begin{matrix}A\\ \vdots\end{matrix}}\wedge 1\mathcal{E} \qquad\rightsquigarrow\qquad \begin{matrix}\vdots\\ A\\ \vdots\end{matrix}$$

Reductions

$$\begin{matrix}\dfrac{\dfrac{\begin{matrix}\vdots\\ u : A\end{matrix} \qquad \begin{matrix}\vdots\\ v : B\end{matrix}}{\langle u, v\rangle : A \wedge B}\,\wedge\mathcal{I}}{\pi_1\langle u, v\rangle : A}\,\wedge 1\mathcal{E}\\ \vdots\end{matrix} \quad \leadsto \quad \begin{matrix}\vdots\\ u : A\\ \vdots\end{matrix}$$

Reductions

$$\begin{array}{c}\dfrac{\dfrac{\begin{array}{c}\vdots\\ u : A\end{array} \qquad \begin{array}{c}\vdots\\ v : B\end{array}}{\langle u, v\rangle : A \wedge B}\,\wedge\mathcal{I}}{\pi_1\langle u, v\rangle : A}\,\wedge 1\mathcal{E}\\ \vdots\end{array} \quad \rightsquigarrow \quad \begin{array}{c}\vdots\\ u : A\\ \vdots\end{array}$$

$$\begin{aligned}\pi_1\langle u, v\rangle &\rightsquigarrow u\\ \pi_2\langle u, v\rangle &\rightsquigarrow v\end{aligned}$$

Deductions and Terms: Implication

$$\dfrac{\begin{array}{c}[A]^i\\ \vdots\\ B\end{array}}{A \Rightarrow B}\Rightarrow\mathcal{I}_i \qquad \dfrac{\begin{array}{c}\vdots\\ A \Rightarrow B\end{array} \qquad \begin{array}{c}\vdots\\ A\end{array}}{B}\Rightarrow\mathcal{E}$$

Deductions and Terms: Implication

$$\dfrac{\begin{array}{c} x_i^A : [A]^i \\ \vdots \\ u : B \end{array}}{\lambda x_i^A \cdot u : A \Rightarrow B} \Rightarrow\mathcal{I}_i \qquad \dfrac{\begin{array}{c} \vdots \\ u : A \Rightarrow B \end{array} \qquad \begin{array}{c} \vdots \\ v : A \end{array}}{uv : B} \Rightarrow\mathcal{E}$$

Disjunction

$$\dfrac{\begin{array}{c}\vdots\\ A\end{array}}{A \vee B}\,\vee l\mathcal{I}$$

$$\dfrac{\begin{array}{c}\vdots\\ B\end{array}}{A \vee B}\,\vee r\mathcal{I}$$

$$\dfrac{\begin{array}{c}\vdots\\ A \vee B\end{array}\qquad\begin{array}{c}[A]\\ \vdots\\ C\end{array}\qquad\begin{array}{c}[B]\\ \vdots\\ C\end{array}}{C}\,\vee\mathcal{E}$$

Disjunction

$$\frac{\begin{matrix}\vdots\\ u : A\end{matrix}}{\iota_l u : A \lor B}\lor l\mathcal{I}$$

$$\frac{\begin{matrix}\vdots\\ B\end{matrix}}{A \lor B}\lor r\mathcal{I}$$

$$\frac{\begin{matrix}\vdots\\ A \lor B\end{matrix}\qquad \begin{matrix}[A]\\ \vdots\\ C\end{matrix}\qquad \begin{matrix}[B]\\ \vdots\\ C\end{matrix}}{C}\lor\mathcal{E}$$

Disjunction

$$\dfrac{\begin{array}{c}\vdots\\ u : A\end{array}}{\iota_l u : A \vee B}\ \vee l\mathcal{I}$$

$$\dfrac{\begin{array}{c}\vdots\\ v : B\end{array}}{\iota_r v : A \vee B}\ \vee r\mathcal{I}$$

$$\dfrac{\begin{array}{c}\\ \vdots\\ A \vee B\end{array} \qquad \begin{array}{c}[A]\\ \vdots\\ C\end{array} \qquad \begin{array}{c}[B]\\ \vdots\\ C\end{array}}{C}\ \vee\mathcal{E}$$

Disjunction

$$\dfrac{\begin{array}{c}\vdots\\ u : A\end{array}}{\iota_l u : A \vee B}\ \vee l\mathcal{I}$$

$$\dfrac{\begin{array}{c}\vdots\\ v : B\end{array}}{\iota_r v : A \vee B}\ \vee r\mathcal{I}$$

$$\dfrac{\begin{array}{c}\\ \vdots\\ r : A \vee B\end{array} \qquad \begin{array}{c}[x : A]\\ \vdots\\ u : C\end{array} \qquad \begin{array}{c}[y : B]\\ \vdots\\ v : C\end{array}}{C}\ \vee\mathcal{E}$$

Disjunction

$$\dfrac{\begin{matrix}\vdots\\ u : A\end{matrix}}{\iota_l u : A \vee B}\,\vee l\mathcal{I}$$

$$\dfrac{\begin{matrix}\vdots\\ v : B\end{matrix}}{\iota_r v : A \vee B}\,\vee r\mathcal{I}$$

$$\dfrac{\begin{matrix}\vdots\\ r : A \vee B\end{matrix} \qquad \begin{matrix}[x : A]\\ \vdots\\ u : C\end{matrix} \qquad \begin{matrix}[y : B]\\ \vdots\\ v : C\end{matrix}}{\delta\ x \cdot u\ \ y \cdot v\ \ r : C}\,\vee\mathcal{E}$$

Reductions

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ r:A\end{array}}{A\vee B}\,\vee I\mathcal{I} \qquad \begin{array}{c}[A]\\ \vdots\\ C\end{array} \qquad \begin{array}{c}[B]\\ \vdots\\ C\end{array}}{\begin{array}{c}C\\ \vdots\end{array}}\,\vee\mathcal{E} \quad \rightsquigarrow \quad \begin{array}{c}\vdots\\ A\\ \vdots\\ C\\ \vdots\end{array}$$

Reductions

$$\cfrac{\cfrac{\begin{array}{c}\vdots\\ r : A\end{array}}{\iota_l r : A \vee B}\,\vee l\mathcal{I} \qquad \begin{array}{c}[x : A]\\ \vdots\\ u : C\end{array} \qquad \begin{array}{c}[y : B]\\ \vdots\\ v : C\end{array}}{\begin{array}{c}C\\ \vdots\end{array}}\,\vee\mathcal{E} \quad \rightsquigarrow \quad \begin{array}{c}\vdots\\ A\\ \vdots\\ C\\ \vdots\end{array}$$

Reductions

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ r : A\end{array}}{\iota_l r : A \vee B}\,\vee l\mathcal{I} \qquad \begin{array}{c}[x : A]\\ \vdots\\ u : C\end{array} \qquad \begin{array}{c}[y : B]\\ \vdots\\ v : C\end{array}}{\begin{array}{c}\delta\ x \cdot u\ y \cdot v\ (\iota_l r) : C\\ \vdots\end{array}}\,\vee\mathcal{E} \qquad \leadsto \qquad \begin{array}{c}\vdots\\ A\\ \vdots\\ C\\ \vdots\end{array}$$

Reductions

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ r : A\end{array}}{\iota_l r : A \vee B}\vee l\mathcal{I} \qquad \begin{array}{c}[x : A]\\ \vdots\\ u : C\end{array} \qquad \begin{array}{c}[y : B]\\ \vdots\\ v : C\end{array}}{\begin{array}{c}\delta\ x \cdot u\ y \cdot v\ (\iota_l r) : C\\ \vdots\end{array}}\vee\mathcal{E} \quad \rightsquigarrow \quad \begin{array}{c}\vdots\\ r : A\\ \vdots\\ u[r/x] : C\\ \vdots\end{array}$$

Reductions

$$\begin{matrix}\dfrac{\dfrac{\begin{matrix}\vdots\\ r : A\end{matrix}}{\iota_l r : A \vee B}\vee l\mathcal{I} \qquad \begin{matrix}[x : A]\\ \vdots\\ u : C\end{matrix} \qquad \begin{matrix}[y : B]\\ \vdots\\ v : C\end{matrix}}{\delta\ x \cdot u\ y \cdot v\ (\iota_l r) : C}\vee\mathcal{E}\\ \vdots\end{matrix} \quad \rightsquigarrow \quad \begin{matrix}\vdots\\ r : A\\ \vdots\\ u[r/x] : C\\ \vdots\end{matrix}$$

$$\begin{aligned}\delta\ x \cdot u\ y \cdot v\ (\iota_l r) &\rightsquigarrow u[r/x]\\ \delta\ x \cdot u\ y \cdot v\ (\iota_r s) &\rightsquigarrow v[s/y]\end{aligned}$$

Commutative Reductions

$$
\cfrac{\cfrac{\begin{array}{c}\vdots\\ A \lor B\end{array} \qquad \begin{array}{c}[A]\\ \vdots\\ C\end{array} \qquad \begin{array}{c}[B]\\ \vdots\\ C\end{array}}{C}\,\lor\mathcal{E} \qquad \vdots}{\begin{array}{c}D\\ \vdots\end{array}}\,R\mathcal{E}
$$

$$
\leadsto \qquad \cfrac{\begin{array}{c}\vdots\\ A \lor B\end{array} \qquad \cfrac{\begin{array}{c}[A]\\ \vdots\\ C\end{array} \qquad \vdots}{D}\,R\mathcal{E} \qquad \cfrac{\begin{array}{c}[B]\\ \vdots\\ C\end{array} \qquad \vdots}{D}\,R\mathcal{E}}{\begin{array}{c}D\\ \vdots\end{array}}\,\lor\mathcal{E}
$$

Commutative Reductions

$$\dfrac{\dfrac{\begin{matrix}\vdots \\ r : A \vee B\end{matrix} \qquad \begin{matrix}[x : A]\\ \vdots \\ u : C\end{matrix} \qquad \begin{matrix}[y : B]\\ \vdots \\ v : C\end{matrix}}{C}\,\vee\mathcal{E} \qquad \vdots}{\begin{matrix}D\\ \vdots\end{matrix}}\,R\mathcal{E}$$

$$\rightsquigarrow \qquad \dfrac{\begin{matrix}\vdots \\ A \vee B\end{matrix} \qquad \dfrac{\begin{matrix}[A]\\ \vdots \\ C\end{matrix} \qquad \vdots}{D}\,R\mathcal{E} \qquad \dfrac{\begin{matrix}[B]\\ \vdots \\ C\end{matrix} \qquad \vdots}{D}\,R\mathcal{E}}{\begin{matrix}D\\ \vdots\end{matrix}}\,\vee\mathcal{E}$$

Commutative Reductions

$$\begin{array}{c}\dfrac{\dfrac{\begin{array}{c}\vdots\\ r : A \vee B\end{array} \qquad \begin{array}{c}[x : A]\\ \vdots\\ u : C\end{array} \qquad \begin{array}{c}[y : B]\\ \vdots\\ v : C\end{array}}{\delta\ x \cdot u\ y \cdot v\ r : C}\vee\mathcal{E} \qquad \vdots}{D}R\mathcal{E}\\ \vdots\end{array}$$

$$\rightsquigarrow \qquad \begin{array}{c}\dfrac{\begin{array}{c}\vdots\\ A \vee B\end{array} \qquad \dfrac{\begin{array}{c}[A]\\ \vdots\\ C\end{array} \qquad \vdots}{D}R\mathcal{E} \qquad \dfrac{\begin{array}{c}[B]\\ \vdots\\ C\end{array} \qquad \vdots}{D}R\mathcal{E}}{D}\vee\mathcal{E}\\ \vdots\end{array}$$

Commutative Reductions

$$\begin{array}{c}
\dfrac{\dfrac{\begin{array}{c}\vdots\\ r : A\vee B\end{array} \qquad \begin{array}{c}[x:A]\\ \vdots\\ u:C\end{array} \qquad \begin{array}{c}[y:B]\\ \vdots\\ v:C\end{array}}{\delta\ x\cdot u\ y\cdot v\ r : C}\vee\mathcal{E} \quad \vdots}{\cdots(\delta\ x\cdot u\ y\cdot v\ r)\cdots : D}R\mathcal{E}\\
\vdots
\end{array}$$

$$\rightsquigarrow \quad \begin{array}{c}
\dfrac{\begin{array}{c}\vdots\\ A\vee B\end{array} \qquad \dfrac{\begin{array}{c}[A]\\ \vdots\\ C\end{array} \quad \vdots}{D}R\mathcal{E} \qquad \dfrac{\begin{array}{c}[B]\\ \vdots\\ C\end{array} \quad \vdots}{D}R\mathcal{E}}{D}\vee\mathcal{E}\\
\vdots
\end{array}$$

Commutative Reductions

$$
\begin{matrix}
\dfrac{\dfrac{\begin{matrix} \vdots \\ r : A \vee B \end{matrix} \qquad \begin{matrix} [x : A] \\ \vdots \\ u : C \end{matrix} \qquad \begin{matrix} [y : B] \\ \vdots \\ v : C \end{matrix}}{\delta\ x \cdot u\ y \cdot v\ r : C}\ \vee\mathcal{E} \qquad \vdots}{\cdots(\delta\ x \cdot u\ y \cdot v\ r)\cdots : D}\ R\mathcal{E} \\
\vdots
\end{matrix}
$$

$$
\rightsquigarrow \qquad
\begin{matrix}
\dfrac{\begin{matrix} \vdots \\ r : A \vee B \end{matrix} \qquad \dfrac{\begin{matrix} [x : A] \\ \vdots \\ u : C \end{matrix} \qquad \vdots}{D}\ R\mathcal{E} \qquad \dfrac{\begin{matrix} [B] \\ \vdots \\ C \end{matrix} \qquad \vdots}{D}\ R\mathcal{E}}{D}\ \vee\mathcal{E} \\
\vdots
\end{matrix}
$$

Commutative Reductions

$$\begin{matrix}\dfrac{\dfrac{\begin{matrix}\vdots\\ r : A \vee B\end{matrix} \qquad \begin{matrix}[x : A]\\ \vdots\\ u : C\end{matrix} \qquad \begin{matrix}[y : B]\\ \vdots\\ v : C\end{matrix}}{\delta\ x \cdot u\ y \cdot v\ r : C}\vee\mathcal{E} \quad \vdots}{\cdots(\delta\ x \cdot u\ y \cdot v\ r)\cdots : D} R\mathcal{E}\\ \vdots\end{matrix}$$

$$\rightsquigarrow \quad \begin{matrix}\dfrac{\begin{matrix}\vdots\\ r : A \vee B\end{matrix} \qquad \dfrac{\begin{matrix}[x : A]\\ \vdots\\ u : C\end{matrix} \quad \vdots}{u' : D} R\mathcal{E} \qquad \dfrac{\begin{matrix}[y : B]\\ \vdots\\ v : C\end{matrix} \quad \vdots}{D} R\mathcal{E}}{D}\vee\mathcal{E}\\ \vdots\end{matrix}$$

Commutative Reductions

$$\begin{array}{c}\dfrac{\dfrac{\begin{array}{c}\vdots\\ r : A \vee B\end{array} \qquad \begin{array}{c}[x : A]\\ \vdots\\ u : C\end{array} \qquad \begin{array}{c}[y : B]\\ \vdots\\ v : C\end{array}}{\delta\ x \cdot u\ y \cdot v\ r : C}\vee\mathcal{E} \quad \vdots}{\cdots(\delta\ x \cdot u\ y \cdot v\ r)\cdots : D}R\mathcal{E}\\ \vdots\end{array}$$

$$\rightsquigarrow \quad \begin{array}{c}\dfrac{\begin{array}{c}\vdots\\ r : A \vee B\end{array} \qquad \dfrac{\begin{array}{c}[x : A]\\ \vdots\\ u : C\end{array} \quad \vdots}{u' : D}R\mathcal{E} \qquad \dfrac{\begin{array}{c}[y : B]\\ \vdots\\ v : C\end{array} \quad \vdots}{v' : D}R\mathcal{E}}{D}\vee\mathcal{E}\\ \vdots\end{array}$$

Commutative Reductions

$$\begin{matrix}\dfrac{\dfrac{\begin{matrix}\vdots\\ r : A \vee B\end{matrix} \qquad \begin{matrix}[x : A]\\ \vdots\\ u : C\end{matrix} \qquad \begin{matrix}[y : B]\\ \vdots\\ v : C\end{matrix}}{\delta\ x \cdot u\ y \cdot v\ r : C}\,\vee\mathcal{E} \qquad \vdots}{\cdots(\delta\ x \cdot u\ y \cdot v\ r)\cdots : D}\,R\mathcal{E}\\ \vdots\end{matrix}$$

$$\rightsquigarrow \qquad \begin{matrix}\dfrac{\begin{matrix}\vdots\\ r : A \vee B\end{matrix} \qquad \dfrac{\begin{matrix}[x : A]\\ \vdots\\ u : C\end{matrix} \qquad \vdots}{u' : D}\,R\mathcal{E} \qquad \dfrac{\begin{matrix}[y : B]\\ \vdots\\ v : C\end{matrix} \qquad \vdots}{v' : D}\,R\mathcal{E}}{\delta\ x \cdot u'\ y \cdot v'\ r : D}\,\vee\mathcal{E}\\ \vdots\end{matrix}$$

Commutative Conversions: Disjunction vs. Disjunction

$$\dfrac{\dfrac{\overset{\vdots}{A \vee B} \qquad \overset{[A]}{\overset{\vdots}{C \vee D}} \qquad \overset{[B]}{\overset{\vdots}{C \vee D}}}{C \vee D}\vee\mathcal{E} \qquad \overset{[C]}{\overset{\vdots}{E}} \qquad \overset{[D]}{\overset{\vdots}{E}}}{\underset{\vdots}{E}}\vee\mathcal{E}$$

$$\dfrac{\overset{\vdots}{A \vee B} \qquad \dfrac{\overset{[A]}{\overset{\vdots}{C \vee D}} \qquad \overset{[C]}{\overset{\vdots}{E}} \qquad \overset{[D]}{\overset{\vdots}{E}}}{E}\vee\mathcal{E} \qquad \dfrac{\overset{[B]}{\overset{\vdots}{C \vee D}} \qquad \overset{[C]}{\overset{\vdots}{E}} \qquad \overset{[D]}{\overset{\vdots}{E}}}{E}\vee\mathcal{E}}{\underset{\vdots}{E}}\vee\mathcal{E}$$

Commutative Conversions: Disjunction vs. Disjunction

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ t : A \vee B\end{matrix} \quad \begin{matrix}[x : A]\\ \vdots\\ u : C \vee D\end{matrix} \quad \begin{matrix}[y : B]\\ \vdots\\ v : C \vee D\end{matrix}}{C \vee D}\,\vee\mathcal{E} \quad \begin{matrix}[C]\\ \vdots\\ E\end{matrix} \quad \begin{matrix}[D]\\ \vdots\\ E\end{matrix}}{\begin{matrix}E\\ \vdots\end{matrix}}\,\vee\mathcal{E}$$

$$\dfrac{\begin{matrix}\vdots\\ A \vee B\end{matrix} \quad \dfrac{\begin{matrix}[A]\\ \vdots\\ C \vee D\end{matrix} \quad \begin{matrix}[C]\\ \vdots\\ E\end{matrix} \quad \begin{matrix}[D]\\ \vdots\\ E\end{matrix}}{E}\,\vee\mathcal{E} \quad \dfrac{\begin{matrix}[B]\\ \vdots\\ C \vee D\end{matrix} \quad \begin{matrix}[C]\\ \vdots\\ E\end{matrix} \quad \begin{matrix}[D]\\ \vdots\\ E\end{matrix}}{E}\,\vee\mathcal{E}}{\begin{matrix}E\\ \vdots\end{matrix}}\,\vee\mathcal{E}$$

Commutative Conversions: Disjunction vs. Disjunction

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ t : A \vee B\end{matrix} \quad \begin{matrix}[x : A]\\ \vdots\\ u : C \vee D\end{matrix} \quad \begin{matrix}[y : B]\\ \vdots\\ v : C \vee D\end{matrix}}{\delta\ x \cdot u\ y \cdot v\ t : C \vee D}\,\vee\mathcal{E} \qquad \begin{matrix}[x' : C]\\ \vdots\\ u' : E\end{matrix} \quad \begin{matrix}[y' : D]\\ \vdots\\ v' : E\end{matrix}}{\begin{matrix}E\\ \vdots\end{matrix}}\,\vee\mathcal{E}$$

$$\dfrac{\begin{matrix}\vdots\\ A \vee B\end{matrix} \quad \dfrac{\begin{matrix}[A]\\ \vdots\\ C \vee D\end{matrix} \quad \begin{matrix}[C]\\ \vdots\\ E\end{matrix} \quad \begin{matrix}[D]\\ \vdots\\ E\end{matrix}}{E}\,\vee\mathcal{E} \quad \dfrac{\begin{matrix}[B]\\ \vdots\\ C \vee D\end{matrix} \quad \begin{matrix}[C]\\ \vdots\\ E\end{matrix} \quad \begin{matrix}[D]\\ \vdots\\ E\end{matrix}}{E}\,\vee\mathcal{E}}{\begin{matrix}E\\ \vdots\end{matrix}}\,\vee\mathcal{E}$$

Commutative Conversions: Disjunction vs. Disjunction

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ t : A \lor B\end{matrix} \quad \begin{matrix}[x : A]\\ \vdots\\ u : C \lor D\end{matrix} \quad \begin{matrix}[y : B]\\ \vdots\\ v : C \lor D\end{matrix}}{\delta\ x \cdot u\ y \cdot v\ t : C \lor D}\lor\mathcal{E} \qquad \begin{matrix}[x' : C]\\ \vdots\\ u' : E\end{matrix} \quad \begin{matrix}[y' : D]\\ \vdots\\ v' : E\end{matrix}}{\begin{matrix}\delta\ x' \cdot u'\ y' \cdot v'\ (\delta\ x \cdot u\ y \cdot v\ t) : E\\ \vdots\end{matrix}}\lor\mathcal{E}$$

$$\dfrac{\begin{matrix}\vdots\\ A \lor B\end{matrix} \quad \dfrac{\begin{matrix}[A]\\ \vdots\\ C \lor D\end{matrix} \quad \begin{matrix}[C]\\ \vdots\\ E\end{matrix} \quad \begin{matrix}[D]\\ \vdots\\ E\end{matrix}}{E}\lor\mathcal{E} \quad \dfrac{\begin{matrix}[B]\\ \vdots\\ C \lor D\end{matrix} \quad \begin{matrix}[C]\\ \vdots\\ E\end{matrix} \quad \begin{matrix}[D]\\ \vdots\\ E\end{matrix}}{E}\lor\mathcal{E}}{\begin{matrix}E\\ \vdots\end{matrix}}\lor\mathcal{E}$$

Commutative Conversions: Disjunction vs. Disjunction

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ t : A \vee B\end{matrix} \quad \begin{matrix}[x : A]\\ \vdots\\ u : C \vee D\end{matrix} \quad \begin{matrix}[y : B]\\ \vdots\\ v : C \vee D\end{matrix}}{\delta\ x \cdot u\ y \cdot v\ t : C \vee D}\vee\mathcal{E} \quad \begin{matrix}[x' : C]\\ \vdots\\ u' : E\end{matrix} \quad \begin{matrix}[y' : D]\\ \vdots\\ v' : E\end{matrix}}{\begin{matrix}\delta\ x' \cdot u'\ y' \cdot v'\ (\delta\ x \cdot u\ y \cdot v\ t) : E\\ \vdots\end{matrix}}\vee\mathcal{E}$$

$$\dfrac{\begin{matrix}\vdots\\ t : A \vee B\end{matrix} \quad \dfrac{\begin{matrix}[x : A]\\ \vdots\\ u : C \vee D\end{matrix} \quad \begin{matrix}[x' : C]\\ \vdots\\ u' : E\end{matrix} \quad \begin{matrix}[y' : D]\\ \vdots\\ v' : E\end{matrix}}{\delta\ x' \cdot u'\ y' \cdot v'\ u : E}\vee\mathcal{E} \quad \dfrac{\begin{matrix}[y : B]\\ \vdots\\ v : C \vee D\end{matrix} \quad \begin{matrix}[x' : C]\\ \vdots\\ u' : E\end{matrix} \quad \begin{matrix}[y' : D]\\ \vdots\\ v' : E\end{matrix}}{\delta\ x' \cdot u'\ y' \cdot v'\ v : E}\vee\mathcal{E}}{\begin{matrix}E\\ \vdots\end{matrix}}\vee\mathcal{E}$$

Commutative Conversions: Disjunction vs. Disjunction

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ t : A \vee B\end{matrix} \quad \begin{matrix}[x : A]\\ \vdots\\ u : C \vee D\end{matrix} \quad \begin{matrix}[y : B]\\ \vdots\\ v : C \vee D\end{matrix}}{\delta\ x \cdot u\ y \cdot v\ t : C \vee D}\,\vee\mathcal{E} \quad \begin{matrix}[x' : C]\\ \vdots\\ u' : E\end{matrix} \quad \begin{matrix}[y' : D]\\ \vdots\\ v' : E\end{matrix}}{\begin{matrix}\delta\ x' \cdot u'\ y' \cdot v'\ (\delta\ x \cdot u\ y \cdot v\ t) : E\\ \vdots\end{matrix}}\,\vee\mathcal{E}$$

$$\dfrac{\begin{matrix}\vdots\\ t : A \vee B\end{matrix} \quad \dfrac{\begin{matrix}[x : A]\\ \vdots\\ u : C \vee D\end{matrix} \quad \begin{matrix}[x' : C]\\ \vdots\\ u' : E\end{matrix} \quad \begin{matrix}[y' : D]\\ \vdots\\ v' : E\end{matrix}}{\delta\ x' \cdot u'\ y' \cdot v'\ u : E}\,\vee\mathcal{E} \quad \dfrac{\begin{matrix}[y : B]\\ \vdots\\ v : C \vee D\end{matrix} \quad \begin{matrix}[x' : C]\\ \vdots\\ u' : E\end{matrix} \quad \begin{matrix}[y' : D]\\ \vdots\\ v' : E\end{matrix}}{\delta\ x' \cdot u'\ y' \cdot v'\ v : E}\,\vee\mathcal{E}}{\begin{matrix}\delta\ x \cdot (\delta\ x' \cdot u'\ y' \cdot v'\ u)\ y \cdot (\delta\ x' \cdot u'\ y' \cdot v'\ v)\ t : E\\ \vdots\end{matrix}}\,\vee\mathcal{E}$$

Commutative Reductions

$$\begin{array}{l}\delta\ x' \cdot u'\ y' \cdot v'\ (\delta\ x \cdot u\ y \cdot v\ \ t) \\ \quad \leadsto \quad \delta\ x \cdot (\delta\ x' \cdot u'\ y' \cdot v'\ u)\ y \cdot (\delta\ x' \cdot u'\ y' \cdot v'\ v)\ t\end{array}$$

Commutative Reductions

$$\begin{array}{l}\delta\ x' \cdot u'\ y' \cdot v'\ (\delta\ x \cdot u\ y \cdot v\ t) \\ \quad \rightsquigarrow \quad \delta\ x \cdot (\delta\ x' \cdot u'\ y' \cdot v'\ u)\ y \cdot (\delta\ x' \cdot u'\ y' \cdot v'\ v)\ t\end{array}$$

$$\begin{array}{l}\pi_1(\delta\ x \cdot u\ y \cdot v\ t) \\ \quad \rightsquigarrow \quad \delta\ x \cdot (\pi_1 u)\ y \cdot (\pi_1 v)\ t\end{array}$$

Commutative Reductions

$$\begin{aligned}
&\delta\ x' \cdot u'\ y' \cdot v'\ (\delta\ x \cdot u\ y \cdot v\ t) \\
&\quad \leadsto \quad \delta\ x \cdot (\delta\ x' \cdot u'\ y' \cdot v'\ u)\ y \cdot (\delta\ x' \cdot u'\ y' \cdot v'\ v)\ t
\end{aligned}$$

$$\begin{aligned}
&\pi_1(\delta\ x \cdot u\ y \cdot v\ t) \\
&\quad \leadsto \quad \delta\ x \cdot (\pi_1 u)\ y \cdot (\pi_1 v)\ t
\end{aligned}$$

$$\begin{aligned}
&\pi_2(\delta\ x \cdot u\ y \cdot v\ t) \\
&\quad \leadsto \quad \delta\ x \cdot (\pi_2 u)\ y \cdot (\pi_2 v)\ t
\end{aligned}$$

Commutative Reductions

$$\begin{aligned}
&\delta\ x' \cdot u'\ y' \cdot v'\ (\delta\ x \cdot u\ y \cdot v\ t) \\
&\quad \rightsquigarrow \quad \delta\ x \cdot (\delta\ x' \cdot u'\ y' \cdot v'\ u)\ y \cdot (\delta\ x' \cdot u'\ y' \cdot v'\ v)\ t
\end{aligned}$$

$$\begin{aligned}
&\pi_1(\delta\ x \cdot u\ y \cdot v\ t) \\
&\quad \rightsquigarrow \quad \delta\ x \cdot (\pi_1 u)\ y \cdot (\pi_1 v)\ t
\end{aligned}$$

$$\begin{aligned}
&\pi_2(\delta\ x \cdot u\ y \cdot v\ t) \\
&\quad \rightsquigarrow \quad \delta\ x \cdot (\pi_2 u)\ y \cdot (\pi_2 v)\ t
\end{aligned}$$

$$\begin{aligned}
&(\delta\ x \cdot u\ y \cdot v\ t)w \\
&\quad \rightsquigarrow \quad \delta\ x \cdot (uw)\ y \cdot (vw)\ t
\end{aligned}$$

Consequences of the Isomorphism

Correspondences

Logic	**λ-calculus**	**Programs**
proof	strongly normalizable term	halting program
cut	redex	function call etc.
cut elimination	reduction	execution step
cut-free proof	normal form	value
formula	type	interface
conjunction	Cartesian product	record
disjunction	direct sum	variant
implication	functional type	function type
universal q.	dependent products	
existential q.	dependent sums	
contradiction	empty type	

Correspondences

Logic	λ-calculus	Programs
proof	strongly normalizable term	halting program
cut	redex	function call etc.
cut elimination	reduction	execution step
cut-free proof	normal form	value
formula	type	interface
conjunction	Cartesian product	record
disjunction	direct sum	variant
implication	functional type	function type
universal q.	dependent products	
existential q.	dependent sums	
contradiction	empty type	

Correspondences

Logic	λ-calculus	Programs
proof	strongly normalizable term	halting program
cut	redex	function call etc.
cut elimination	reduction	execution step
cut-free proof	normal form	value
formula	type	interface
conjunction	Cartesian product	record
disjunction	direct sum	variant
implication	functional type	function type
universal q.	dependent products	
existential q.	dependent sums	
contradiction	empty type	

Correspondences

Logic	**λ-calculus**	**Programs**
proof	strongly normalizable term	halting program
cut	redex	function call etc.
cut elimination	reduction	execution step
cut-free proof	normal form	value
formula	type	interface
conjunction	Cartesian product	record
disjunction	direct sum	variant
implication	functional type	function type
universal q.	dependent products	
existential q.	dependent sums	
contradiction	empty type	

Correspondences

Logic	λ-calculus	Programs
proof	strongly normalizable term	halting program
cut	redex	function call etc.
cut elimination	reduction	execution step
cut-free proof	normal form	value
formula	type	interface
conjunction	Cartesian product	record
disjunction	direct sum	variant
implication	functional type	function type
universal q.	dependent products	
existential q.	dependent sums	
contradiction	empty type	

Correspondences

Logic	**λ-calculus**	**Programs**
proof	strongly normalizable term	halting program
cut	redex	function call etc.
cut elimination	reduction	execution step
cut-free proof	normal form	value
formula	type	interface
conjunction	Cartesian product	record
disjunction	direct sum	variant
implication	functional type	function type
universal q.	dependent products	
existential q.	dependent sums	
contradiction	empty type	

Correspondences

Logic	λ-calculus	Programs
proof	strongly normalizable term	halting program
cut	redex	function call etc.
cut elimination	reduction	execution step
cut-free proof	normal form	value
formula	type	interface
conjunction	Cartesian product	record
disjunction	direct sum	variant
implication	functional type	function type
universal q.	dependent products	
existential q.	dependent sums	
contradiction	empty type	

Correspondences

Logic	λ-calculus	Programs
proof	strongly normalizable term	halting program
cut	redex	function call etc.
cut elimination	reduction	execution step
cut-free proof	normal form	value
formula	type	interface
conjunction	Cartesian product	record
disjunction	direct sum	variant
implication	functional type	function type
universal q.	dependent products	
existential q.	dependent sums	
contradiction	empty type	

Correspondences

Logic	λ-calculus	Programs
proof	strongly normalizable term	halting program
cut	redex	function call etc.
cut elimination	reduction	execution step
cut-free proof	normal form	value
formula	type	interface
conjunction	Cartesian product	record
disjunction	direct sum	variant
implication	functional type	function type
universal q.	dependent products	
existential q.	dependent sums	
contradiction	empty type	

Correspondences

Logic	**λ-calculus**	**Programs**
proof	strongly normalizable term	halting program
cut	redex	function call etc.
cut elimination	reduction	execution step
cut-free proof	normal form	value
formula	type	interface
conjunction	Cartesian product	record
disjunction	direct sum	variant
implication	functional type	function type
universal q.	dependent products	
existential q.	dependent sums	
contradiction	empty type	

Correspondences

Logic	λ-calculus	Programs
proof	strongly normalizable term	halting program
cut	redex	function call etc.
cut elimination	reduction	execution step
cut-free proof	normal form	value
formula	type	interface
conjunction	Cartesian product	record
disjunction	direct sum	variant
implication	functional type	function type
universal q.	dependent products	
existential q.	dependent sums	
contradiction	empty type	

Differences

- logic focuses on propositions (types)
- type theory focuses on programs (proofs)
- non-trivial type systems allow non-terminating programs
- such terms can be typed $\perp$
- so...
  - "propositions as types; proofs as programs" is ok
  - the converse generally does not yield a sane logical system

The system F
[Girard et al., 1989, Chap. 11],[Girard, 2004, Chap. 6]

- Discovered by Jean-Yves Girard (1972).
- A typed λ-calculus
- Known as the second-order or polymorphic λ-calculus
- Formalizes the notion of parametric polymorphism in programming languages
- Corresponds to a second-order logic via Curry-Howard
- $\vdash \Lambda\alpha \cdot \lambda x^{\alpha} \cdot x : \forall\alpha \cdot \alpha \rightarrow \alpha$
- Strongly normalizing
- Type inference is undecidable

The system **F**

[Girard et al., 1989, Chap. 11],[Girard, 2004, Chap. 6]

- Discovered by Jean-Yves Girard (1972).
- A typed λ-calculus
- Known as the second-order or polymorphic λ-calculus
- Formalizes the notion of parametric polymorphism in programming languages
- Corresponds to a second-order logic via Curry-Howard
- $\vdash \Lambda\alpha \cdot \lambda x^{\alpha} \cdot x : \forall\alpha \cdot \alpha \to \alpha$
- Strongly normalizing
- Type inference is undecidable

The system **F**

[Girard et al., 1989, Chap. 11],[Girard, 2004, Chap. 6]

- Discovered by Jean-Yves Girard (1972).
- A typed λ-calculus
- Known as the second-order or polymorphic λ-calculus
- Formalizes the notion of parametric polymorphism in programming languages
- Corresponds to a second-order logic via Curry-Howard
- $\vdash \Lambda\alpha \cdot \lambda x^{\alpha} \cdot x : \forall\alpha \cdot \alpha \to \alpha$
- Strongly normalizing
- Type inference is undecidable

The system F
[Girard et al., 1989, Chap. 11],[Girard, 2004, Chap. 6]

- Discovered by Jean-Yves Girard (1972).
- A typed λ-calculus
- Known as the second-order or polymorphic λ-calculus
- Formalizes the notion of parametric polymorphism in programming languages
- Corresponds to a second-order logic via Curry-Howard
- $\vdash \Lambda\alpha \cdot \lambda x^{\alpha} \cdot x : \forall\alpha \cdot \alpha \to \alpha$
- Strongly normalizing
- Type inference is undecidable

The system **F**

[Girard et al., 1989, Chap. 11],[Girard, 2004, Chap. 6]

- Discovered by Jean-Yves Girard (1972).
- A typed λ-calculus
- Known as the second-order or polymorphic λ-calculus
- Formalizes the notion of parametric polymorphism in programming languages
- Corresponds to a second-order logic via Curry-Howard
- $\vdash \Lambda\alpha \cdot \lambda x^{\alpha} \cdot x : \forall\alpha \cdot \alpha \to \alpha$
- Strongly normalizing
- Type inference is undecidable

The system **F**

[Girard et al., 1989, Chap. 11],[Girard, 2004, Chap. 6]

- Discovered by Jean-Yves Girard (1972).
- A typed λ-calculus
- Known as the second-order or polymorphic λ-calculus
- Formalizes the notion of parametric polymorphism in programming languages
- Corresponds to a second-order logic via Curry-Howard
- $\vdash \Lambda\alpha \cdot \lambda x^{\alpha} \cdot x : \forall\alpha \cdot \alpha \to \alpha$
- Strongly normalizing
- Type inference is undecidable

The system F

[Girard et al., 1989, Chap. 11],[Girard, 2004, Chap. 6]

- Discovered by Jean-Yves Girard (1972).
- A typed λ-calculus
- Known as the second-order or polymorphic λ-calculus
- Formalizes the notion of parametric polymorphism in programming languages
- Corresponds to a second-order logic via Curry-Howard
- $\vdash \Lambda\alpha \cdot \lambda x^{\alpha} \cdot x : \forall\alpha \cdot \alpha \to \alpha$
- Strongly normalizing
- Type inference is undecidable

The system **F**

[Girard et al., 1989, Chap. 11],[Girard, 2004, Chap. 6]

- Discovered by Jean-Yves Girard (1972).
- A typed λ-calculus
- Known as the second-order or polymorphic λ-calculus
- Formalizes the notion of parametric polymorphism in programming languages
- Corresponds to a second-order logic via Curry-Howard
- $\vdash \Lambda\alpha \cdot \lambda x^\alpha \cdot x : \forall\alpha \cdot \alpha \to \alpha$
- Strongly normalizing
- Type inference is undecidable

The system **F**

$$\mathsf{T} := \Lambda\alpha \cdot \lambda x^\alpha \cdot \lambda y^\alpha \cdot x : \forall\alpha \cdot \alpha \to \alpha \to \alpha$$
$$\mathsf{F} := \Lambda\alpha \cdot \lambda x^\alpha \cdot \lambda y^\alpha \cdot y : \forall\alpha \cdot \alpha \to \alpha \to \alpha$$

Beware that these function take three arguments.

$$\text{Boolean} := \forall\alpha \cdot \alpha \to \alpha \to \alpha$$

The system F

$$\mathsf{T} := \Lambda\alpha \cdot \lambda x^{\alpha} \cdot \lambda y^{\alpha} \cdot x : \forall\alpha \cdot \alpha \to \alpha \to \alpha$$
$$\mathsf{F} := \Lambda\alpha \cdot \lambda x^{\alpha} \cdot \lambda y^{\alpha} \cdot y : \forall\alpha \cdot \alpha \to \alpha \to \alpha$$

Beware that these function take three arguments.

$$\text{Boolean} := \forall\alpha \cdot \alpha \to \alpha \to \alpha$$

The system **F**

$$\begin{aligned}
\text{and} &:= \lambda x^{\text{Boolean}} \cdot \lambda y^{\text{Boolean}} \cdot x \ \text{Boolean}\ y\ \text{F} \\
\text{or} &:= \lambda x^{\text{Boolean}} \cdot \lambda y^{\text{Boolean}} \cdot x \ \text{Boolean}\ \text{T}\ y \\
\text{and} &:= \lambda x^{\text{Boolean}} \cdot \lambda y^{\text{Boolean}} \cdot x \ \text{Boolean}\ \text{F}\ \text{T}
\end{aligned}$$

Agda

Dependently Typed Programming in Agda

> *In Hindley-Milner style languages, such as Haskell and ML, there is a clear separation between types and values.*
> *In a dependently typed language the line is more blurry — types can contain (depend on) arbitrary values and appear as arguments and results of ordinary functions.*
>
> — [Norell, 2009]

Datatypes and Pattern Matching

Booleans

```
data Bool : Set where
  true  : Bool
  false : Bool

not : Bool → Bool
not true  = false
not false = true

_or_ : Bool → Bool → Bool
true  or x = x
false or _ = false
```

Datatypes and Pattern Matching

Natural Numbers

```
data Nat : Set where
  zero : Nat
  suc  : Nat → Nat

_+_ : Nat → Nat → Nat
zero  + m = m
suc n + m = suc (n + m)

_*_ : Nat → Nat → Nat
zero  * m = zero
suc n * m = m + n * m
```

Guaranteed termination (termination checker).

Syntax

```
if_then_else_ : {A : Set} → Bool → A → A → A
if true  then x else y = x
if false then x else y = y

infixl 60 _*_
infixl 40 _+_
infixr 20 _or_
infix  5  if_then_else_
```

Evaluation is in normal order: function first.

But not call-by-need: computations are not shared.

Lists

```
infixr 40 _::_
data List (A : Set) : Set where
  []   : List A
  _::_ : A → List A → List A
```

```
data _⋆ (α : Set) : Set where
  ε   : α ⋆
  _▷_ : α → α ⋆ → α ⋆
```

Lists

```
infixr 40 _::_
data List (A : Set) : Set where
  []   : List A
  _::_ : A → List A → List A
```

```
data _⋆ (α : Set) : Set where
  ε   : α ⋆
  _▷_ : α → α ⋆ → α ⋆
```

Dependent Functions

`(x : A) → B`

- Type of functions
- taking an `x` of type `A`
- return a value of type `B`
- where `x` may appear in `B`

Special case: `x` is a type.

Dependent Functions

```
identity : (A : Set) → A → A
identity A x = x
zero' : Nat
zero' = identity Nat zero


apply : (A : Set)(B : A → Set) →
        ((x : A) → B x) → (a : A) → B a
apply A B f a = f a
```

Shorthands:

- `(x : A)(y : B) → C`
  for `(x : A) → (y : B) → C`
- `(x y : A → B)`
  for `(x : A)(x : A) → B`

Implicit Arguments

`{x : A} → B`

- Same as `(x : A) → B`
- but when used, let the type checker figure out `x`

```
id : {A : Set} → A → A
id x = x

true' : Bool
true' = id true
```

Or using `_` to request assistance from the type checker:

```
one : Nat
one = identity _ (suc zero)
```

Implicit Arguments

```
map : {A B : Set} → (A → B) → List A → List B
map f []        = []
map f (x :: xs) = f x :: map f xs

_++_ : {A : Set} → List A → List A → List A
[]        ++ ys = ys
(x :: xs) ++ ys = x :: (xs ++ ys)
```

A Tricky One

```
_○_ : {A : Set}
      {B : A → Set}
      {C : (x : A) → B x → Set}
      (f : {x : A}(y : B x) → C x y)
      (g : (x : A) → B x)
      (x : A)
       → C x (g x)
(f ○ g) x = f (g x)
```

A Tricky One

```
_∘_ :
      (f : B → C)
      (g : A → B)
      (x : A)
       → C
(f ∘ g) x = f (g x)
```

A Tricky One

```
_∘_ : {A : Set}
      {B : Set}
      {C : Set}
      (f : B → C)
      (g : A → B)
      (x : A)
       → C
(f ∘ g) x = f (g x)
```

A Tricky One

```
_∘_ : {A : Set}
      {B : A → Set}
      {C : Set}
      (f : B → C)
      (g : (x : A) → B x)
      (x : A)
       → C
(f ∘ g) x = f (g x)
```

A Tricky One

```
_∘_ : {A : Set}
      {B : A → Set}
      {C : Set}
      (f : (B x) → C)
      (g : (x : A) → B x)
      (x : A)
       → C
(f ∘ g) x = f (g x)
```

A Tricky One

```
_○_ : {A : Set}
      {B : A → Set}
      {C : Set}
      (f : {x : A}(B x) → C)
      (g : (x : A) → B x)
      (x : A)
       → C
(f ○ g) x = f (g x)
```

A Tricky One

```
_○_ : {A : Set}
      {B : A → Set}
      {C : (x : A) → B x → Set}
      (f : {x : A}(y : B x) → C x y)
      (g : (x : A) → B x)
      (x : A)
       → C x (g x)
(f ○ g) x = f (g x)
```

Datatype families

```
data Vec (A : Set) : Nat → Set where
  []    : Vec A zero
  _::_  : {n : Nat} → A → Vec A n → Vec A (suc n)
```

- A is a parameter
- Nat provides the indexes

Datatype families

```
head : {A : Set}{n : Nat} → Vec A (suc n) → A
head (x :: xs) = x
```

- this is correct, but. . .
- what should be surprising?
- where is the case for the empty list?
- the type checker knows it's impossible here!
- why?
- `Vec A (suc n)`

Datatype families

```
head : {A : Set}{n : Nat} → Vec A (suc n) → A
head (x :: xs) = x
```

- this is correct, but. . .
- what should be surprising?
- where is the case for the empty list?
- the type checker knows it's impossible here!
- why?
- `Vec A (suc n)`

Datatype families

```
head : {A : Set}{n : Nat} → Vec A (suc n) → A
head (x :: xs) = x
```

- this is correct, but. . .
- what should be surprising?
- where is the case for the empty list?
- the type checker knows it's impossible here!
- why?
- `Vec A (suc n)`

Datatype families

```
head : {A : Set}{n : Nat} → Vec A (suc n) → A
head (x :: xs) = x
```

- this is correct, but. . .
- what should be surprising?
- where is the case for the empty list?
- the type checker knows it's impossible here!
- why?
- `Vec A (suc n)`

Datatype families

```
head : {A : Set}{n : Nat} → Vec A (suc n) → A
head (x :: xs) = x
```

- this is correct, but. . .
- what should be surprising?
- where is the case for the empty list?
- the type checker knows it's impossible here!
- why?
- `Vec A (suc n)`

Datatype families

```
head : {A : Set}{n : Nat} → Vec A (suc n) → A
head (x :: xs) = x
```

- this is correct, but. . .
- what should be surprising?
- where is the case for the empty list?
- the type checker knows it's impossible here!
- why?
- `Vec A (suc n)`

Recommended Readings

[Norell, 2009]
A thorough introduction to Agda.

https://youtu.be/Da9WjINqY9c
Introductory examples on proof-terms in Coq.

Bibliography I

Girard, J.-Y. (2004).
Cours de Logique, Rome, Automne 2004.
http://logica.uniroma3.it/uif/corso/.

Girard, J.-Y., Lafont, Y., and Taylor, P. (1989).
Proofs and Types.
Cambridge University Press.
http://www.cs.man.ac.uk/~pt/stable/Proofs+Types.html.

Norell, U. (2009).
Dependently typed programming in Agda.
In *Proceedings of the 4th International Workshop on Types in Language Design and Implementation*, TLDI '09, pages 1–2, New York, NY, USA. ACM.