FLOLAC 2020

Logic

III. The Curry–Howard correspondence

柯向上

中央研究院資訊科學研究所

joshko@iis.sinica.edu.tw

Annotated derivation

$$
\dfrac{\dfrac{\dfrac{\overline{\mathtt{A},\ \mathtt{A}\to\mathtt{B}\vdash\mathtt{A}\to\mathtt{B}} \qquad \overline{\mathtt{A},\ \mathtt{A}\to\mathtt{B}\vdash\mathtt{A}}}{\mathtt{A},\ \mathtt{A}\to\mathtt{B}\vdash\mathtt{B}}\,(\to\mathsf{E})}{\mathtt{A}\vdash(\mathtt{A}\to\mathtt{B})\to\mathtt{B}}\,(\to\mathsf{I})}{\vdash\mathtt{A}\to(\mathtt{A}\to\mathtt{B})\to\mathtt{B}}\,(\to\mathsf{I})
$$

Annotated derivation

$$\dfrac{\dfrac{\dfrac{\overline{\mathtt{x} : \mathtt{A},\ \mathtt{y} : \mathtt{A} \to \mathtt{B} \vdash \mathtt{A} \to \mathtt{B}} \qquad \overline{\mathtt{x} : \mathtt{A},\ \mathtt{y} : \mathtt{A} \to \mathtt{B} \vdash \mathtt{A}}}{\mathtt{x} : \mathtt{A},\ \mathtt{y} : \mathtt{A} \to \mathtt{B} \vdash \mathtt{B}}\,(\to\mathsf{E})}{\mathtt{x} : \mathtt{A} \vdash (\mathtt{A} \to \mathtt{B}) \to \mathtt{B}}\,(\to\mathsf{I})}{\vdash \mathtt{A} \to (\mathtt{A} \to \mathtt{B}) \to \mathtt{B}}\,(\to\mathsf{I})$$

- Label elements in contexts with (distinct) names.

Annotated derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathtt{x} : \mathtt{A},\ \mathtt{y} : \mathtt{A} \to \mathtt{B} \vdash \mathtt{y} : \mathtt{A} \to \mathtt{B}} \qquad \dfrac{}{\mathtt{x} : \mathtt{A},\ \mathtt{y} : \mathtt{A} \to \mathtt{B} \vdash \mathtt{x} : \mathtt{A}}}{\mathtt{x} : \mathtt{A},\ \mathtt{y} : \mathtt{A} \to \mathtt{B} \vdash \mathtt{B}}\,(\to\mathsf{E})}{\mathtt{x} : \mathtt{A} \vdash (\mathtt{A} \to \mathtt{B}) \to \mathtt{B}}\,(\to\mathsf{I})}{\vdash \mathtt{A} \to (\mathtt{A} \to \mathtt{B}) \to \mathtt{B}}\,(\to\mathsf{I})$$

- Label elements in contexts with (distinct) names.
- Represent (assum) by the name of the assumption used.

Annotated derivation

$$\dfrac{\dfrac{\dfrac{\overline{\mathtt{x : A,\ y : A \to B \vdash y : A \to B}} \qquad \overline{\mathtt{x : A,\ y : A \to B \vdash x : A}}}{\mathtt{x : A,\ y : A \to B \vdash y\ x : B}}\;(\to\mathsf{E})}{\mathtt{x : A \vdash (A \to B) \to B}}\;(\to\mathsf{I})}{\mathtt{\vdash A \to (A \to B) \to B}}\;(\to\mathsf{I})$$

- Label elements in contexts with (distinct) names.
- Represent (assum) by the name of the assumption used.
- Represent (→E) by juxtaposing the representations of its two sub-derivations.

Annotated derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathtt{x} : \mathtt{A},\ \mathtt{y} : \mathtt{A} \to \mathtt{B} \vdash \mathtt{y} : \mathtt{A} \to \mathtt{B}} \qquad \dfrac{}{\mathtt{x} : \mathtt{A},\ \mathtt{y} : \mathtt{A} \to \mathtt{B} \vdash \mathtt{x} : \mathtt{A}}}{\mathtt{x} : \mathtt{A},\ \mathtt{y} : \mathtt{A} \to \mathtt{B} \vdash \mathtt{y}\ \mathtt{x} : \mathtt{B}}\ (\to\mathsf{E})}{\mathtt{x} : \mathtt{A} \vdash \lambda\,\mathtt{y}.\ \mathtt{y}\ \mathtt{x} : (\mathtt{A} \to \mathtt{B}) \to \mathtt{B}}\ (\to\mathsf{I})}{\vdash \lambda\,\mathtt{x}.\ \lambda\,\mathtt{y}.\ \mathtt{y}\ \mathtt{x} : \mathtt{A} \to (\mathtt{A} \to \mathtt{B}) \to \mathtt{B}}\ (\to\mathsf{I})$$

- Label elements in contexts with (distinct) names.
- Represent (assum) by the name of the assumption used.
- Represent (→E) by juxtaposing the representations of its two sub-derivations.
- Represent (→I) by prefixing $\lambda\, v$. to the representation of its sub-derivation, where v is the name of the new assumption.

Annotated derivation

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\mathtt{x : A,\ y : A \to B \vdash y : A \to B}}\,(\mathsf{var}) \qquad \dfrac{}{\mathtt{x : A,\ y : A \to B \vdash x : A}}\,(\mathsf{var})}{\mathtt{x : A,\ y : A \to B \vdash y\ x : B}}\,(\mathsf{app})}{\mathtt{x : A \vdash \lambda\, y.\ y\ x : (A \to B) \to B}}\,(\mathsf{abs})}{\mathtt{\vdash \lambda\, x.\ \lambda\, y.\ y\ x : A \to (A \to B) \to B}}\,(\mathsf{abs})
$$

- Label elements in contexts with (distinct) names.
- Represent (assum) by the name of the assumption used.
- Represent (→E) by juxtaposing the representations of its two sub-derivations.
- Represent (→I) by prefixing $\lambda\, v.$ to the representation of its sub-derivation, where v is the name of the new assumption.

This is a typing derivation for the λ-term $\mathtt{\lambda\, x.\ \lambda\, y.\ y\ x}$!

Simply typed λ-calculus (à la Curry)

Let the set of *types* be the *implicational fragment* of $\textsc{Prop}$, i.e., the subset of the propositional language generated by variables and implication only.

A λ-term t is said to *have type τ under context Γ* if, using the following rules, there is a closed typing derivation whose conclusion is $\Gamma \vdash t : \tau$. In this case we simply write $\Gamma \vdash t : \tau$.

$$\frac{}{\Gamma \vdash v : \tau}\,(\text{var}) \quad \text{if} \quad (v : \tau) \in \Gamma$$

$$\frac{\Gamma, v : \sigma \vdash t : \tau}{\Gamma \vdash \lambda v.\ t : \sigma \to \tau}\,(\text{abs}) \qquad \frac{\Gamma \vdash t : \sigma \to \tau \qquad \Gamma \vdash s : \sigma}{\Gamma \vdash t\, s : \tau}\,(\text{app})$$

The Curry–Howard correspondence

Deduction systems and programming calculi can be put in correspondence — a corresponding pair of a deduction system and a programming calculus can be regarded as logical and computational interpretations of essentially the same set of syntactic objects.

Slogan: *propositions are types; proofs are programs.*

Natural deduction for full propositional logic corresponds to simply typed λ-calculus with constants: defining the set of types to be $\textsc{Prop}$, the derivations in natural deduction (the proofs) correspond exactly to the well-typed λ-terms (the programs).

Cartesian products

Conjunctions correspond to cartesian products: the introduction rule gives type to the pairing operator,

$$\frac{\Gamma \vdash s : \sigma \qquad \Gamma \vdash t : \tau}{\Gamma \vdash \langle s, t \rangle : \sigma \wedge \tau}\ (\wedge\mathsf{I})$$

and the two elimination rules give types to the projections.

$$\frac{\Gamma \vdash t\ :\ \sigma \wedge \tau}{\Gamma \vdash \mathtt{outl}\ t\ :\ \sigma}\ (\wedge\mathsf{EL}) \qquad \frac{\Gamma \vdash t\ :\ \sigma \wedge \tau}{\Gamma \vdash \mathtt{outr}\ t\ :\ \tau}\ (\wedge\mathsf{ER})$$

Note that we are adding the constants $\langle \_, \_ \rangle$, `outl`, and `outr` into the language of λ-calculus.

Disjoint sums

Disjunctions correspond to disjoint sums (unions): the introduction rules give types to the injections,

$$\frac{\Gamma \vdash s : \sigma}{\Gamma \vdash \mathtt{inl}\ s : \sigma \vee \tau}\,(\vee\mathrm{IL}) \qquad \frac{\Gamma \vdash t : \tau}{\Gamma \vdash \mathtt{inr}\ t : \sigma \vee \tau}\,(\vee\mathrm{IR})$$

and the elimination rule gives type to the conditional operator.

$$\frac{\Gamma \vdash c : \sigma \vee \tau \qquad \Gamma, u : \sigma \vdash s : \vartheta \qquad \Gamma, v : \tau \vdash t : \vartheta}{\Gamma \vdash \mathtt{case}\ c \left[\begin{array}{l} u \rightsquigarrow s \\ v \rightsquigarrow t \end{array} \right. : \vartheta}\,(\vee\mathrm{E})$$

Again we add the constants $\mathtt{inl}$, $\mathtt{inr}$, and $\mathtt{case}\_ \left[\begin{array}{l} \_ \rightsquigarrow \_ \\ \_ \rightsquigarrow \_ \end{array} \right.$ to the language of λ-calculus.

Empty set

$\perp$ is interpreted as the empty set. The elimination rule gives type to a variant of Dijkstra's `abort` operator.

$$\frac{\Gamma \vdash t : \perp}{\Gamma \vdash \texttt{abort}\ t : \varphi}\ (\perp\mathsf{E})$$

Example. The type $\top$, i.e., $\perp \to \perp$, is inhabited by $\lambda\,\texttt{x}.\ \texttt{abort}\ \texttt{x}$.

Question. What is the computational meaning of `abort`?

Example: distributivity

The type

$$\mathtt{A} \wedge (\mathtt{B} \vee \mathtt{C}) \to (\mathtt{A} \wedge \mathtt{B}) \vee (\mathtt{A} \wedge \mathtt{C})$$

is inhabited by the λ-term

$$\lambda\,\mathtt{x}.\ \mathtt{case}\ (\mathtt{outr}\ \mathtt{x}) \left[\begin{array}{l} \mathtt{y} \rightsquigarrow \mathtt{inl}\ \langle \mathtt{outl}\ \mathtt{x}, \mathtt{y} \rangle \\ \mathtt{z} \rightsquigarrow \mathtt{inr}\ \langle \mathtt{outl}\ \mathtt{x}, \mathtt{z} \rangle \end{array}\right. .$$

Exercise. 'Uncompress' the term and find the corresponding derivation.

Question. When constructing the derivation,

Question. If asked to prove the classical truth of this proposition, would you prefer constructing a program or a truth table?

δ-reduction

In pure λ-calculus we have β-reduction that rewrites β-redexes.

$$(\lambda\, v.\ s)\ t \ \rightsquigarrow_{\beta}\ s\,[t/v]$$

Note that this is how an elimination form (application) cancels out an introduction form (λ-abstraction) for the same connective (Gentzen's inversion principle).

For λ-calculus with constants, we should also specify how to reduce the *δ-redexes*, which involve the introduction and elimination forms of the other connectives.

$$\texttt{outl}\ \langle s, t\rangle \ \rightsquigarrow_{\delta}\ s \qquad \texttt{outr}\ \langle s, t\rangle \ \rightsquigarrow_{\delta}\ t$$

$$\texttt{case}\ (\texttt{inl}\ p) \left[\begin{array}{l} u \rightsquigarrow s \\ v \rightsquigarrow t \end{array}\right. \ \rightsquigarrow_{\delta}\ s\,[p/u]$$

$$\texttt{case}\ (\texttt{inr}\ q) \left[\begin{array}{l} u \rightsquigarrow s \\ v \rightsquigarrow t \end{array}\right. \ \rightsquigarrow_{\delta}\ t\,[q/v]$$

Proof normalisation

β-/δ-redexes in λ-terms correspond to *detours* in derivations, and evaluation of λ-terms corresponds to *proof normalisation*.

$$\dfrac{\dfrac{\dfrac{\overline{\mathtt{B \to C \to B, A} \vdash \mathtt{B \to C \to B}}}{\mathtt{B \to C \to B} \vdash \mathtt{A \to B \to C \to B}}\,(\to\mathsf{I})}{\vdash (\mathtt{B \to C \to B}) \to \mathtt{A \to B \to C \to B}}\,(\to\mathsf{I}) \qquad \dfrac{\dfrac{\overline{\mathtt{B, C} \vdash \mathtt{B}}}{\mathtt{B} \vdash \mathtt{C \to B}}\,(\to\mathsf{I})}{\vdash \mathtt{B \to C \to B}}\,(\to\mathsf{I})}{\vdash \mathtt{A \to B \to C \to B}}\,(\to\mathsf{E})$$

normalises to

$$\dfrac{\dfrac{\dfrac{\overline{\mathtt{A, B, C} \vdash \mathtt{B}}}{\mathtt{A, B} \vdash \mathtt{C \to B}}\,(\to\mathsf{I})}{\mathtt{A} \vdash \mathtt{B \to C \to B}}\,(\to\mathsf{I})}{\cancel{\mathtt{B \to C \to B}} \vdash \mathtt{A \to B \to C \to B}}\,(\to\mathsf{I})$$

The corresponding reduction is

$$(\lambda\,\mathtt{x}.\ \lambda\,\mathtt{y}.\ \mathtt{x})\ (\lambda\,\mathtt{z}.\ \lambda\,\mathtt{w}.\ \mathtt{z}) \ \rightsquigarrow_{\beta}\ \lambda\,\mathtt{y}.\ \lambda\,\mathtt{z}.\ \lambda\,\mathtt{w}.\ \mathtt{z}.$$

Detours

Corresponding to the β-/δ-redexes, the possible forms of detours are

$$\dfrac{\dfrac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\,(\to\mathsf{I}) \qquad \Gamma \vdash \varphi}{\Gamma \vdash \psi}\,(\to\mathsf{E})$$

$$\dfrac{\dfrac{\Gamma \vdash \varphi \qquad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\,(\wedge\mathsf{I})}{\Gamma \vdash \varphi}\,(\wedge\mathsf{EL}) \qquad \dfrac{\dfrac{\Gamma \vdash \varphi \qquad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\,(\wedge\mathsf{I})}{\Gamma \vdash \psi}\,(\wedge\mathsf{ER})$$

$$\dfrac{\dfrac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}\,(\vee\mathsf{IL}) \qquad \Gamma, \varphi \vdash \vartheta \qquad \Gamma, \psi \vdash \vartheta}{\Gamma \vdash \vartheta}\,(\vee\mathsf{E})$$

$$\dfrac{\dfrac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}\,(\vee\mathsf{IR}) \qquad \Gamma, \varphi \vdash \vartheta \qquad \Gamma, \psi \vdash \vartheta}{\Gamma \vdash \vartheta}\,(\vee\mathsf{E})$$

Exercise. What are the results of eliminating the above detours?

Gentzen's inversion principle

Quoting (the English translation of) Gentzen's own words:

> *The introductions represent, as it were, the 'definitions' of the symbols concerned, and the eliminations are no more, in the final analysis, than the consequences of these definitions. This fact may be expressed as follows: In eliminating a symbol, we may use the formula with whose terminal symbol we are dealing only 'in the sense afforded it by the introduction of that symbol'.*

Question. What is the relationship between Gentzen's inversion principle and the general notion of computation (whatever that is)?

Question. Gentzen's inversion principle is asymmetric in its treatment of introduction and elimination. Does the dual of the principle make any sense?

Subject reduction and strong normalisation

For simply typed λ-calculus we have the following results.

Theorem (subject reduction)**.** If $\Gamma \vdash t : \tau$ and $t \rightsquigarrow_{\beta\delta} t'$, then $\Gamma \vdash t' : \tau$.

Theorem (strong normalisation)**.** Every reduction sequence of a well-typed λ-term terminates at a normal form.

They are readily translated into theorems about derivations.

Theorem. Elimination of a detour produces a derivation with the same conclusion.

Theorem. Every derivation can be normalised (to a derivation that does not contain detours).

Canonicity

Definition. A λ-term is in *canonical form* if its outermost constructor is an introduction form, i.e., one of the following:

- λ-abstraction,
- pairing $\langle \_, \_ \rangle$, and
- injections `inl` and `inr`.

Theorem (canonicity)**.** If $\vdash t : \tau$ and t is in normal form, then t is in canonical form.

PROOF Induction on the typing derivation of t. The elimination forms give rise to redexes, in contradiction to the assumption that t is in normal form.

Exercise. Expand the above proof sketch.

Question. Is it important for a deduction system to have strong normalisation and canonicity?

Question. Why is the context empty in the canonicity statement?

Classical axioms

We obtained the classical deduction system NK by adding to NJ an inference rule, which corresponds to introducing a constant, say

$$\frac{}{\Gamma \vdash \mathtt{LEM}\,\varphi : \varphi \vee \neg\varphi}\,\textsf{(LEM)}$$

We do not know how to reduce `LEM`, however. This breaks canonicity, and the deduction system ceases to be computationally meaningful.

Question. Why is there a connection between constructivity and computation?

Question. Without canonicity, is NK still meaningful?

Underivability

Corollary. NJ is consistent, i.e., $\nvdash_{\mathrm{NJ}} \bot$.

PROOF If $\vdash_{\mathrm{NJ}} \bot$, then there is a λ-term of type $\bot$ in canonical form. But none of the canonical forms can have type $\bot$.

Corollary (disjunction property). If $\vdash_{\mathrm{NJ}} \varphi \vee \psi$, then either $\vdash_{\mathrm{NJ}} \varphi$ or $\vdash_{\mathrm{NJ}} \psi$.

PROOF A λ-term of type $\varphi \vee \psi$ under the empty context can be reduced to either `inl` *p* where $\vdash p : \varphi$ or `inr` *q* where $\vdash q : \psi$.

Remark. The disjunction property does not hold for NK.

Corollary. $\mathtt{A} \vee \neg\mathtt{A}$ is underivable in NJ.

PROOF If $\vdash_{\mathrm{NJ}} \mathtt{A} \vee \neg\mathtt{A}$, then either $\vdash_{\mathrm{NJ}} \mathtt{A}$ or $\vdash_{\mathrm{NJ}} \neg\mathtt{A}$ by the disjunction property, and thus either $\models \mathtt{A}$ or $\models \neg\mathtt{A}$ by soundness. But neither $\mathtt{A}$ nor $\neg\mathtt{A}$ is a tautology.

Unifying programming and reasoning

The Curry–Howard correspondence suggests that programs and proofs be identified. Both of them are *mental constructions*, which are exactly what intuitionistic mathematics cares about.

Per Martin-Löf: ‘If programming is understood

- not as the writing of instructions for this or that computing machine
- but as the design of methods of computation that it is the computer’s duty to execute
  - (a difference that Dijkstra has referred to as the difference between comput**er** science and comput**ing** science),

then it no longer seems possible to distinguish the discipline of programming from constructive mathematics.’

Martin-Löf Type Theory

Martin-Löf Type Theory is an influential framework in which programs and proofs are treated uniformly. It is simultaneously

- a computationally meaningful higher-order logic system and
- a very expressively typed functional programming language.

There are numerous variations, extensions, and applications of MLTT. The *dependently typed* programming language Agda that we will see next is one of its descendants.