

# Propositions and Types, Proofs and Programs

## Part IV: Curry-Howard

Ranald Clouston

School of Computing

Australian National University

ANU Logic Summer School 2024

# The Strange Coincidence

It seems bizarre or arbitrary that **logic** and **computation** should come together like this

- Why should Church's 1930s algorithm for universal models of computation be the same as Prawitz's 1960s algorithm for tidying up needlessly complex proofs?
- Moreover, this keeps happening: beyond Prawitz-Church we have Hindley-Milner; Girard-Reynolds; Parigot-(Sussman and Steele). Each is a logician-computer scientist pair independently inventing the same algorithms for the purposes of their field.
- This is not even counting newer work that deliberately moves concepts back and forth, e.g. modal types.

# Return to BHK

A clue lies in this part of the BHK interpretation:

*“A proof of  $A \rightarrow B$  is a construction which permits us to transform any proof of  $A$  into a proof of  $B$ ”*

A ‘construction that permits a transformation’ is an odd informal phrase, but its mathematical counterpart is clearly that of *function*.

Hence a formal treatment of the BHK interpretation of implication in intuitionistic logic seems to require some sort of ‘theory of functions’

- Like the lambda calculus!
- This is *not* to say that Curry-Howard only applies to implication and functions;
- But this is the heart of the isomorphism, where it is most obvious

# Some Aspects of Curry-Howard

Propositions

Types

Proofs

Programs

Proof Normalisation

Beta-Reduction

Assumptions

Free Variables

Function Type-Former

Implication

Other Type-Formers

Other Logical Connectives

# Lambda Calculus as Notation for Proofs

If nothing more, the lambda calculus is a formal notation for natural deduction proofs, so we don't always need to write out trees.

By writing proofs in this notation, a computer can check we have made no mistakes.

- So long as type-checking for the lambda calculus can be automated
- (which it can)

The question whether a proposition is valid can be restated as **type inhabitation**: does a program exist with this type?

# Examples of Type Inhabitation

The theorem  $A \rightarrow A$  is inhabited by, for example:

$\lambda x^A. x$  - the identity function, sometimes called 'I'

The axiom  $A \rightarrow (B \rightarrow A)$  is inhabited by:

$\lambda x^A. \lambda y^B. x$  - this function is often called 'K'

The axiom  $(A \rightarrow B) \rightarrow ((A \rightarrow (B \rightarrow C)) \rightarrow (A \rightarrow C))$  is inhabited by:

$\lambda f^{A \rightarrow B}. \lambda g^{A \rightarrow (B \rightarrow C)}. \lambda x^A. (g x) (f x)$  - a.k.a. 'S'

Nothing new here! We've seen these as natural deduction proofs.

# Other Connectives: $\wedge$ and $\times$

$\wedge$  is defined via a pair of proofs; hence, a pair of programs

- We call the collection of such a pairs a **product**, and write it with  $\times$
- Introduction is pairing; Elimination is projection

$$\frac{\Gamma \vdash t:A \quad \Gamma \vdash u:B}{\Gamma \vdash \langle t, u \rangle : A \times B}$$

$$\frac{\Gamma \vdash p:A \times B}{\Gamma \vdash \pi^1 p:A}$$

$$\frac{\Gamma \vdash p:A \times B}{\Gamma \vdash \pi^2 p:B}$$

# Other Connectives: $\vee$ and $+$

$\vee$  is introduced by proving one side of the disjunction

- Like products, this is a concept which comes up in programming;
- When we package different types together we call the resulting type a **sum**, or **disjoint union**. We will use the  $+$  symbol (sometimes  $|$  is used instead);
- Introduction is injection; Elimination is case matching

$$\frac{\Gamma \vdash t:A}{\Gamma \vdash \iota^1 t:A+B}$$

$$\frac{\Gamma \vdash t:B}{\Gamma \vdash \iota^2 t:A+B}$$

$$\frac{\Gamma \vdash s:A+B \quad \Gamma, x:A \vdash t:C \quad \Gamma, y:B \vdash u:C}{\Gamma \vdash \delta[s, x.t, y.u]:C}$$



# Other Connectives: $\perp$ and $\emptyset$

$\perp$  corresponds to a type with no elements

- Usually written  $\emptyset$
- Admittedly not very useful for basic functional programming

$$\frac{\Gamma \vdash t:\emptyset}{\Gamma \vdash \varepsilon t:A}$$

Sums and the empty type, as with products, have their own notions of reduction / normalisation, but I will focus on  $\rightarrow$  and  $\times$  for now.

# Roundabout Proofs with Conjunction

Recall the 'roundabout' (part of a) proof:

$$\frac{\frac{A \quad B}{A \wedge B}}{A}$$

We thought this should be reducible to the top left proof of A

# Roundabout Programs with Products

Translated into lambda calculus:

$$\frac{\frac{\Gamma \vdash t:A \quad \Gamma \vdash u:B}{\Gamma \vdash \langle t, u \rangle : A \times B}}{\Gamma \vdash \pi^1 \langle t, u \rangle : A}$$

Following the previous slide, can we reduce the bottom program to the top left program?

- Yes! This is the reduction one might expect,  $\pi^1 \langle t, u \rangle \mapsto t$
- $\pi^2 \langle t, u \rangle \mapsto u$  motivated similarly

# Another Roundabout Proof with Conjunction

How about this roundabout?

$$\frac{\frac{A \wedge B}{A} \quad \frac{A \wedge B}{B}}{A \wedge B}$$

(where the two top proofs of  $A \wedge B$  are the same)

This suggests another reduction

- $\langle \pi^1 t, \pi^2 t \rangle \mapsto t$

This is called an **eta** (instead of beta) rule

- Important for powerful type systems, but not needed for basic functional languages, so I will ignore in this course (also for other connectives).

# Important Properties

What would we like to be true about the typed computational system we are building?

## **Subject Reduction / Preservation / Soundness:**

- If  $\Gamma \vdash t:A$  and  $t \mapsto u$  then  $\Gamma \vdash u:A$

## **Canonicity / Progress / Completeness:**

- Roughly, reduction should not end until we've reached an acceptable value

## **Strong Normalisation:**

- No infinite loops by applying reduction repeatedly
- ('weak' normalisation asks whether there is *any* finite path to termination)

# Roundabout Proofs with Implication

Follow the introduction-then-elimination pattern as for products:

$$\frac{\frac{[A] \quad B}{A \rightarrow B}}{A \quad B} B$$

We cannot just replace the final proof of B with the earlier proof of B

- The earlier proof of B has A as an assumption
- Not killed if we get rid of the  $\rightarrow$ I rule
- But we can replace the assumption A with the right hand proof of A!

# Assumption Replacement as Substitution

Given a proof of B *with* an assumption A, and a proof of A, we should be able to give proof of B *without* the assumption A.

- What does this mean in the language of the lambda calculus?
- Replace all occurrences of a **free variable** of type A with a **term** of type A

This is substitution!

**Substitution lemma:** If  $\Gamma, x:A \vdash t:B$  and  $\Gamma \vdash u:A$  then  
 $\Gamma \vdash t[u/x]:B$

The roundabout elimination from the previous slides is then  
 $(\lambda x.t)u \mapsto t[u/x]$  as expected!

# Strong Normalisation

Strong normalisation follows trivially for products, but not implication

- $(\lambda x.t)u \mapsto t[u/x]$  will not necessarily reduce the length of the term

It also does *not* hold in any obvious way by induction on proof rules

$$\frac{\Gamma \vdash f:A \rightarrow B \quad \Gamma \vdash t:A}{\Gamma \vdash ft:B}$$

Suppose for induction that  $f$  and  $t$  are strongly normalising

- Say  $f$  reduces to normal form  $f'$  and  $t$  to normal form  $t'$
- Then  $ft$  reduces to  $f't'$  but this need not be normal:  $f'$  could start with  $\lambda$



# Tait's Method

We prove by induction a property that is *stronger* than strong normalisation.

- This method is outlined in detail in Chapter 6 of [Proofs and Types](#)

Define, for each type  $A$ , a set  $RED_A$  of untyped lambda terms

- We call these the **reducible terms** for type  $A$
- For any base type  $b$ ,  $t \in RED_b$  if  $t$  is strongly normalising
- $t \in RED_{A \times B}$  if  $\pi^1 t \in RED_A$  and  $\pi^2 t \in RED_B$
- $t \in RED_{A \rightarrow B}$  if for all  $u \in RED_A$  we have  $tu \in RED_B$

We then prove by induction on the RED sets that all reducible terms are strongly normalizing, and by induction on typing rules that all terms of type  $A$  are in  $RED_A$ .

# Canonicity

Remember that we want a property called canonicity that captures the idea that the rules of reduction are 'strong enough'.

This is easiest to see if we have some inhabited base types

- Suppose we have a base type `Bool` with two elements, `True` and `False`
- We would like terms of type `Bool` to compute until they return one of these two elements, not 'get stuck' prematurely
- This fails for arbitrary terms with free variables, e.g.  $x:\text{Bool} \vdash x:\text{Bool}$

So this is a property of *closed* terms with no free variables

- $\vdash t:\text{Bool}$  and  $t$  is normal (cannot reduce), then  $t$  is either `True` or `False`

# Canonicity, Generally

Canonicity for Bool is proved by a general statement:

- If  $\vdash t:A$  and  $t$  is normal then its outermost connective is an introduction rule.
- So closed terms of function types start with  $\lambda$ ; of product type are pairs; and of type `Bool` are `True` or `False`.
- Simple proof by induction on the length of terms.

For the intuitionist, this is a perfect match for the BHK interpretation!

- A normal proof without assumptions of an implication theorem is always a  $\lambda$ -abstraction, i.e. a function
- A normal proof without assumptions of a conjunction theorem is always a pair of proofs of theorems

# Roundabout Proofs with Disjunction

The introduction-then-elimination pattern:

$$\frac{\frac{A}{A \vee B} \quad [A] \quad C \quad [B] \quad C}{C}$$

We return the first proof of  $C$  in the line above (discarding the second) but with assumption  $A$  replaced by the proof of  $A$

- Similarly if  $A \vee B$  came via  $\vee I_2$  from  $B$
- $\perp$  does not need a new beta rule at all – no introduction, no roundabouts!

# Beta Reduction Summarised

Five beta rules:

- $(\lambda x.t)u \mapsto t[u/x]$
- $\pi^1\langle t, u \rangle \mapsto t$
- $\pi^2\langle t, u \rangle \mapsto u$
- $\delta[\iota^1 r, x.t, y.u] \mapsto t[r/x]$
- $\delta[\iota^2 r, x.t, y.u] \mapsto u[r/y]$

The properties of subject reduction, strong normalisation, and canonicity all continue to hold.

But there is one further desirable property that holds for the function-product fragment, but fails when we add sums and/or the empty type....

# Subformula Property

It can sometimes seem difficult to write a natural deduction proof:

$$\frac{A \rightarrow B \quad A}{B}$$

Where does  $A$  come from? Could it be anything?

In fact there is a remarkable theorem called the subformula property:

If  $\Gamma \vdash t : B$  and  $t$  is normal, then all subterms of  $t$  have a type that is a subtype of  $B$ , or a subtype of some type in  $\Gamma$ .

- So there are only finitely many (normal) choices of  $A$  in the example above

# Failure of the Subformula Property

The subformula property *fails* for disjunction and falsum:

$$\begin{array}{c}
 \frac{A \vee A \quad \frac{[A] \quad [A]}{A \wedge A} \quad \frac{[A] \quad [A]}{A \wedge A}}{A \wedge A} \\
 \frac{A \wedge A}{A}
 \end{array}
 \qquad
 \frac{\perp}{A \wedge B}
 \qquad
 \frac{A \wedge B}{A}$$

- These derivations are normal:
- $\pi^i \delta[s, x.\langle x, x \rangle, y.\langle y, y \rangle]$  and  $\pi^1 \varepsilon z$
- But  $A \wedge A$  not a subformula of  $A \vee A$  or  $A$ , and  $A \wedge B$  not a subformula of  $\perp$  or  $A$

The problem is the arbitrary ‘parasitic’ conclusion of  $\vee E$  and  $\perp E$

# New Conversions for $\perp$

$\perp$ E followed by an elimination reduces to  $\perp$ E:

- $\pi^1 \varepsilon^{A \wedge B} t \mapsto \varepsilon^A t$
- $\pi^2 \varepsilon^{A \wedge B} t \mapsto \varepsilon^B t$
- $(\varepsilon^{A \rightarrow B} t) u \mapsto \varepsilon^B t$
- $\delta^C [\varepsilon^{A \vee B} t, x.u, y.v] \mapsto \varepsilon^C t$
- $\varepsilon^A \varepsilon^\perp t \mapsto \varepsilon^A t$

In terms of functions:

- There is always exactly one (trivial) function from  $\emptyset$  to anything else

With these new conversions the subformula property holds

- The old properties need to be rechecked, but happily also still hold



# New Conversions for $\vee$ - Example

Where  $\vee E$  is followed by an elimination, push the elimination up:

$$\frac{\frac{A \vee B \quad \frac{\frac{[A] \quad C \wedge D}{C \wedge D} \quad [B] \quad C \wedge D}{C \wedge D}}{C \wedge D}}{C}}{\text{to} \quad \frac{A \vee B \quad \frac{\frac{[A] \quad C \wedge D}{C} \quad [B] \quad C \wedge D}{C}}{C}}$$

These are collectively known as the **commuting conversions**.

# Commuting Conversions for $V$

- $\pi^1 \delta[s, x.t, y.u] \mapsto \delta[s, x.\pi^1 t, y.\pi^1 u]$
- $\pi^2 \delta[s, x.t, y.u] \mapsto \delta[s, x.\pi^2 t, y.\pi^2 u]$
- $(\delta[s, x.t, y.u])v \mapsto \delta[s, x.tv, y.uv]$
- $\varepsilon \delta[s, x.t, y.u] \mapsto \delta[s, x.\varepsilon t, y.\varepsilon u]$
- $\delta[\delta[s, x.t, y.u], x'.t', y'.u']$   
 $\mapsto \delta[s, x.\delta[t, x'.t', y'.u'], \delta[u, x'.t', y'.u']]$

These are not all easy to read!

- But do give us the subformula property without sacrificing other properties

# Checking our Counter-Examples

The counter-example to the subformula property for  $\perp$  resolves easily:

$$\frac{\frac{\perp}{A \wedge B}}{A} \quad \text{to} \quad \frac{\perp}{A}$$

i.e.  $\pi^1 \varepsilon Z \mapsto \varepsilon Z$

# Checking our Counter-Examples

The counter-example for  $v$ :

$$\frac{\begin{array}{ccc} & [A] & [A] \\ & \hline AVA & A \wedge A & A \wedge A \end{array}}{\hline A \wedge A}$$
$$\frac{A \wedge A}{A}$$

i.e.  $\pi^i \delta[s, x.\langle x, x \rangle, y.\langle y, y \rangle]$

# Checking our Counter-Examples

The counter-example for  $\forall$ :

$$\frac{\begin{array}{cc} \frac{[A] \quad [A]}{A \wedge A} & \frac{[A] \quad [A]}{A \wedge A} \\ A \vee A & A \end{array}}{A}$$

i.e.  $\pi^i \delta[s, x.\langle x, x \rangle, y.\langle y, y \rangle] \mapsto \delta[s, x.\pi^i \langle x, x \rangle, y.\pi^i \langle y, y \rangle]$

# Checking our Counter-Examples

The counter-example for  $\forall$ :

$$\frac{A \vee A \quad [A] \quad [A]}{A}$$

$$\begin{aligned} \text{i.e. } \pi^i \delta[s, x.\langle x, x \rangle, y.\langle y, y \rangle] &\mapsto \delta[s, x.\pi^i \langle x, x \rangle, y.\pi^i \langle y, y \rangle] \\ &\mapsto \delta[s, x.x, y.y] \end{aligned}$$

# Conclusion

- There is an isomorphism between certain sorts of logic and certain sorts of programs
- The connectives of propositional logic correspond to type-formers
- Hence the formulae of logic correspond to types
- Natural deduction proofs correspond to functional programs
- Proof Normalisation corresponds to Beta-Reduction
  - Perhaps with commuting conversions, and/or eta conversions
- The remainder of our time will be spent extending these key ideas