Propositions and Types, Proofs and Programs

Part IV: Curry-Howard

Ranald Clouston
School of Computing
Australian National University

ANU Logic Summer School 2023

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

I think the clue lies in this part of the BHK interpretation:

> *"A proof of* A→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 $A \to A$ is inhabited by, for example:

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

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

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

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

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

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

Other Connectives: ∧ and ×

∧ 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 ×
- Introduction is pairing; Elimination is projection

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

$$\frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \pi^1 p : A} \qquad \frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \pi^2 p : B}$$

Products and the Lambda Calculus

Unlike λ and application, pairing and projection are not part of the untyped lambda calculus.

- They are not needed, as pairs can be encoded
- But moving to STLC breaks our encodings
- As it should! We want products to be their own type so e.g. you can only take the first projection of a term of product type.

Similar comments apply to other connectives

Other Connectives: ∨ and **+**

∨ is defined via one proof, which is 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

$$\frac{\Gamma \vdash t{:}A}{\Gamma \vdash \iota^1 t{:}A+B} \qquad \frac{\Gamma \vdash t{:}B}{\Gamma \vdash \iota^2 t{:}A+B}$$

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

Other Connectives: $\bot$ and `0`

$\bot$ corresponds to a type with no elements

- Usually written `0`
- Admittedly not very useful for basic functional programming

$$\frac{\Gamma \vdash t : 0}{\Gamma \vdash \varepsilon t : A}$$

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

Roundabout Proofs with Conjunction

Recall the ‘roundabout’ (part of a) proof:

$$\dfrac{\dfrac{A \qquad\qquad B}{A \wedge B}}{A}$$

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

Roundabout Programs with Products

Translated into lambda calculus:

$$\dfrac{\dfrac{\Gamma \vdash \mathtt{t:A} \qquad\qquad \Gamma \vdash \mathtt{u:B}}{\Gamma \vdash \mathtt{<t,u>:A\times B}}}{\Gamma \vdash \pi^1\mathtt{<t,u>:A}}$$

Following the previous slide, can we reduce to the top-level program?

- Yes! This is the reduction one might expect, $\pi^1\mathtt{<t,u>} \mapsto \mathtt{t}$
- $\pi^2\mathtt{<t,u>} \mapsto \mathtt{u}$ motivated similarly

Another Roundabout Proof with Conjunction

How about this roundabout?

$$\dfrac{\dfrac{A \wedge B}{A} \qquad \dfrac{A \wedge B}{B}}{A \wedge B}$$

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

This suggests another reduction

- `<π¹t,π²t> ↦ 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**:

- We're not ready to define this yet, but we do not want reduction to end until we've reached our tidiest possible proof / sensible result of computation

Strong Normalisation:

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

Important Properties for Products

Subject Reduction:

- General definition: if $\Gamma \vdash t{:}A$ and $t \mapsto u$ then $\Gamma \vdash u{:}A$
- If $\Gamma \vdash \pi^1\langle t,u\rangle{:}A$ then $\Gamma \vdash t{:}A$
- If $\Gamma \vdash \pi^2\langle t,u\rangle{:}B$ then $\Gamma \vdash u{:}B$
- Trivial to prove these facts from the proof rules

Strong Normalisation:

- $\pi^1\langle t,u\rangle \mapsto t$ and $\pi^2\langle t,u\rangle \mapsto t$ reduce the size of terms
- If the original term is finitely long, we can only reduce finitely many times

Roundabout Proofs with Implication

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

$$\dfrac{\dfrac{\begin{matrix}[A] \\ B\end{matrix}}{A \to B} \qquad A}{B}$$

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 →I rule
- But we can replace the assumption A with the right hand proof of A!

Assumption Replacement as Substitution

A proof of B with an assumption A, plus a proof of A, should always give us a 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!

Subject Reduction

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

Follows by induction on proof rules.

Hence

Subject Reduction holds for the system with functions and products

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

Induction on Proof Rules

I don’t want to sweat too many proof details in these lectures

- But it would be good to understand what ‘induction on proof rules’ involves
- Used a lot to prove properties of type systems

Assume that a certain property holds for all premises of each proof rule, and show that it holds for the conclusion

- Let’s look at some cases for the substitution lemma

Substitution Lemma: Base Cases

"If $\Gamma, x{:}A \vdash t{:}B$ and $\Gamma \vdash u{:}A$ then $\Gamma \vdash t[u/x]{:}B$"

Two cases to consider for the axiom rule:

- If $\Gamma, x{:}A \vdash x{:}A$ and $\Gamma \vdash u{:}A$ then $\Gamma \vdash x[u/x]=u{:}A$
- If $\Gamma, y{:}B, x{:}A \vdash y{:}B$ and $\Gamma, y{:}B \vdash u{:}A$ then $\Gamma, y{:}B \vdash y[u/x]=y{:}B$

The first case, where the variable substituted is the same as the variable introduced, follows immediately.

The second case, where the two variables are different, follows because we can introduce y:B regardless of any other variables.

Substitution Lemma: →E

"If $\Gamma, x{:}A \vdash t{:}B$ and $\Gamma \vdash u{:}A$ then $\Gamma \vdash t[u/x]{:}B$"

$$\frac{\Gamma, x{:}A \vdash f{:}C\to B \qquad \Gamma, x{:}A \vdash t{:}C}{\Gamma, x{:}A \vdash ft{:}B}$$

By induction $\Gamma \vdash f[u/x]{:}C\to B$ and $\Gamma \vdash t[u/x]{:}C$

Hence by →E we have $\Gamma \vdash (f[u/x])(t[u/x]){:}B$

But $(f[u/x])(t[u/x])$ is $(ft)[u/x]$

Strong Normalisation

Strong normalisation does *not* hold in the easy way it did for products

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

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

$$\frac{\Gamma \vdash f{:}A \to B \qquad \Gamma \vdash t{:}A}{\Gamma \vdash ft{:}B}$$

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 λ

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** of type A
- For any base type b, $t \in \mathrm{RED}_b$ if t is strongly normalising
- $t \in \mathrm{RED}_{A\times B}$ if $\pi^1 t \in \mathrm{RED}_A$ and $\pi^2 t \in \mathrm{RED}_B$
- $t \in \mathrm{RED}_{A\to B}$ if for all $u \in \mathrm{RED}_A$ we have $t\,u \in \mathrm{RED}_B$

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:Bool ⊢ x:Bool`

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

- `⊢ t: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 λ; 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 of an implication theorem is always a λ-abstraction, i.e. a function
- A normal proof of a conjunction theorem is always a pair of proofs of theorems

Roundabout Proofs with Disjunction

The introduction-then-elimination pattern:

$$\dfrac{\dfrac{A}{A \vee B} \qquad \begin{matrix}[A] \\ C\end{matrix} \qquad \begin{matrix}[B] \\ C\end{matrix}}{C}$$

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:

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

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:

$$\frac{A \to B \qquad A}{B}$$

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 Γ.

- 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:

$$\dfrac{\dfrac{A\vee A \qquad \dfrac{[A] \quad [A]}{A\wedge A} \qquad \dfrac{[A] \quad [A]}{A\wedge A}}{A\wedge A}}{A} \qquad\qquad \dfrac{\dfrac{\bot}{A\wedge B}}{A}$$

- 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 $\bot$ or A

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

New Conversions for ⊥

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

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

In terms of functions:

- There is always exactly one (trivial) function from `0` 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 ∨ - **Example**

Where ∨E is followed by an elimination, push the elimination up:

$$\dfrac{\dfrac{A \vee B \quad \begin{matrix}[A] \\ C \wedge D\end{matrix} \quad \begin{matrix}[B] \\ C \wedge D\end{matrix}}{C \wedge D}}{C} \qquad \text{to} \qquad \dfrac{A \vee B \quad \dfrac{\begin{matrix}[A] \\ C \wedge D\end{matrix}}{C} \quad \dfrac{\begin{matrix}[B] \\ C \wedge D\end{matrix}}{C}}{C}$$

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

Commuting Conversions for ∨

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

  $\mapsto \delta[s,x.\delta[t,x'.t',y'.u'],\delta[u,x'.t',y'.u']]$

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 ⊥ resolves easily:

$$\dfrac{\dfrac{\perp}{A \wedge B}}{A} \qquad \text{to} \qquad \dfrac{\perp}{A}$$

i.e. $\pi^1 \varepsilon\, z \;\mapsto\; \varepsilon\, z$

Checking our Counter-Examples

The counter-example for ∨:

$$\dfrac{\dfrac{A\vee A \qquad \dfrac{[A] \qquad [A]}{A\wedge A} \qquad \dfrac{[A] \qquad [A]}{A\wedge A}}{A\wedge A}}{A}$$

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

Checking our Counter-Examples

The counter-example for $\vee$:

$$\dfrac{A \vee A \qquad \dfrac{\dfrac{[A] \qquad [A]}{A \wedge A}}{A} \qquad \dfrac{\dfrac{[A] \qquad [A]}{A \wedge A}}{A}}{A}$$

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 $\vee$:

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

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

$\mapsto \delta[s, x.x, y.y]$

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, or even eta conversions

- The remainder of our time will be spent extending these key ideas