Logic and Proofs

CSE203

Benjamin Werner — Pierre-Yves Strub

École polytechnique

Constructive proofs and the Curry-Howard isomorphism

December 7th 2022

Bachelor S3 2022-23

What are proofs ?

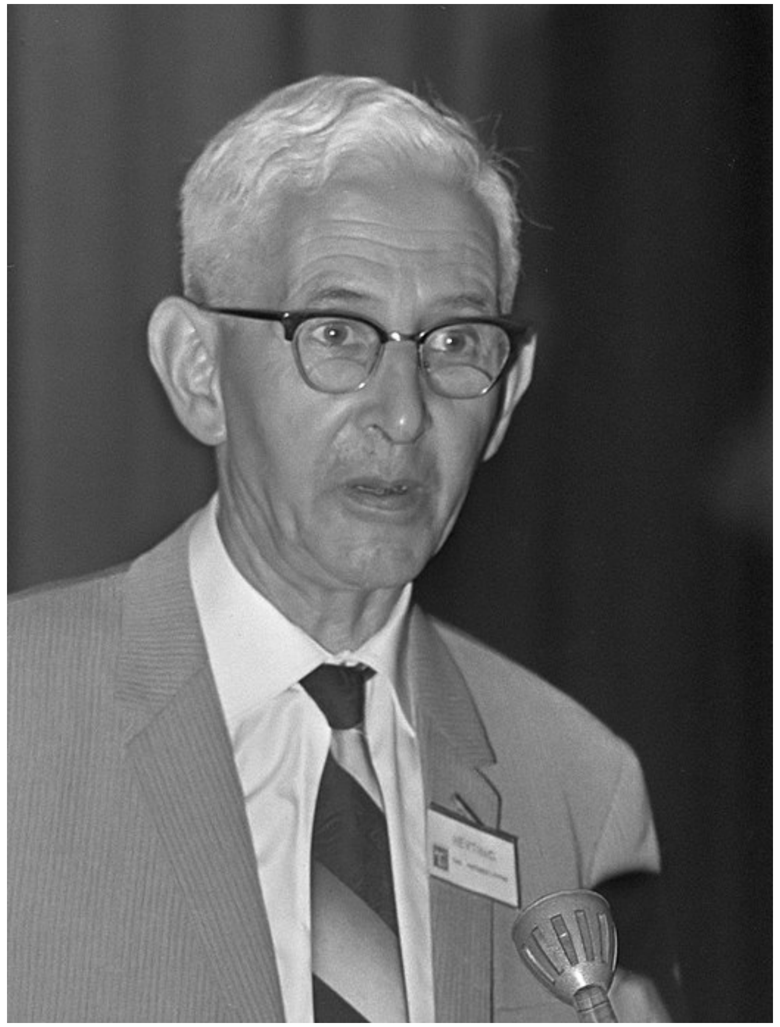
one answer, deduction trees :

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{\text{ax}}{(A\vee B)\wedge C\,,\,A \vdash A}
\qquad
\dfrac{\dfrac{\text{ax}}{(A\vee B)\wedge C\,,\,A \vdash (A\vee B)\wedge C}}{(A\vee B)\wedge C\,,\,A \vdash C}
}{(A\vee B)\wedge C\,,\,A \vdash (A\wedge C)}\;\wedge\text{-i}
}{(A\vee B)\wedge C\,,\,A \vdash (A\wedge C)\vee(B\wedge C)}\;\vee\text{-i1}
\qquad
\dfrac{\cdots}{(A\vee B)\wedge C\,,\,B \vdash (A\wedge C)\vee(B\wedge C)}
\qquad
\wedge\text{-e1}\,\dfrac{\dfrac{\text{ax}}{(A\vee B)\wedge C \vdash (A\vee B)\wedge C}}{(A\vee B)\wedge C \vdash A\vee B}
}{
\dfrac{(A\vee B)\wedge C \vdash (A\wedge C)\vee(B\wedge C)}{(A\vee B)\wedge C \Rightarrow (A\wedge C)\vee(B\wedge C)}\;\Rightarrow\text{-i}
}\;\vee\text{-E}
$$

This is a purely syntactical answer…

Heyting's proposal

In the 1920s, the school of *intuitionistic* mathematics or *constructive* mathematics



Arend Heyting

L.E.J. Brouwer

Heyting's semantics

A proof of… is…

$A \wedge B$ — a way to compute… — a pair (a, b) where a is a proof of A, b is a proof of B

$A \vee B$ — a pair (ε, c) where $\varepsilon = 0$ and $c : A$ or $\varepsilon = 1$ and $c : B$

$A \Rightarrow B$ — a function f s.t. if $a : A$ then $f(a) : B$

Constructivism

In architecture

In art…

What is constructivism in mathematics ?

Constructive semantics

$A \vee B$ ⟶ a pair (ε, c) where $\varepsilon = 0$ and $c : A$
or $\varepsilon = 1$ and $c : B$

Computational content

we can check which is the true proposition (A or B)

$\exists\, x. A(x)$ ⟶ a pair (t, a) where $a : A(t)$

the "witness"

the proof

$\forall x. A(x)$ ⟶ a function f s.t. $f(t) : A(t)$

A non constructive proof

$$\boxed{\exists\,(a,b) \in \mathbb{R},\quad a \notin \mathbb{Q} \;\wedge\; b \notin \mathbb{Q} \;\wedge\; a^{\mathbf{b}} \in \mathbb{Q}}$$

We know that $\sqrt{2} \notin \mathbb{Q}$

If $\sqrt{2}^{\sqrt{2}} \in \mathbb{Q}$ ok: a = b = $\sqrt{2}$

If $\sqrt{2}^{\sqrt{2}} \notin \mathbb{Q}$ take : a = $\sqrt{2}^{\sqrt{2}}$ b= $\sqrt{2}$

we have $a^b = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{(\sqrt{2}\sqrt{2})} = \sqrt{2}^{2} = 2 \in \mathbb{Q}$

Heyting's semantics

A proof of…	is…
$A \wedge B$	a pair (a, b) where a is a proof of A b is a proof of B
$A \vee B$	a pair (ε, c) where $\varepsilon = 0$ and $c : A$ or $\varepsilon = 0$ and $c : B$
$A \Rightarrow B$	a function f s.t. if $a : A$ then $f(a) : B$

Seems familiar ?

Heyting's semantics

A proof of… is…

$A \wedge B$ — a pair $(a,b) : A \wedge B$ if $a : A$, $b : B$

$(a,b) : A \times B$

$A \vee B$ — a pair (ε, c) where $\varepsilon = 0$ and $c : A$

or $\varepsilon = 0$ and $c : B$

$(\varepsilon, c) : A + B$

$A \Rightarrow B$ — a function f s.t. if $a : A$ then $f(a) : B$

$f : A \to B$

Seems familiar ?

Propositions are types !

The Curry-Howard isomorphism

$$
\begin{tikzcd}
\text{Propositions} \arrow[rr, leftrightarrow] \arrow[d] & & \text{Types} \arrow[d] \\
\text{Proofs} \arrow[rr, leftrightarrow] & & \text{programs}
\end{tikzcd}
$$

Propositions-as-types
proofs-as-programs

The proofs are programs

⇒-fragment

```
fun a => a  :  A -> A
```

```
fun ab bc a => bc (ab a) :
                 (A -> B) -> (B -> C) -> A -> C
```

Conjunction = cartesian product

```
Inductive and (A B : Prop) : Prop :=
 | conj : A -> B -> (and A B).
```

```
fun ab : A /\ B => match ab with
   | conj a b => conj b a
   end
                                   :   A/\B -> B/\A
```

Disjunction = sum-type

```
Inductive or (A B : Prop) : Prop :=
| or_introl : A -> or A B
| or_intror : B -> or A B
```

```
fun ab =>
   match ab with
   | or_introl a => or_intror a
   | or_intror b => or_introl b
   end                             : A\/B -> B\/A
```

Dependent Types

`forall x : A, B` is a function type

`A -> B` is just a notation for `forall x : A, B` when `x` does not occur in `B`

`even : nat -> Prop` is a function from `nat` to types
`Prop` is a type of types (like `Type`)

The details of the difference between `Prop` and `Type` is out of the scope of the course

Interesting example : existential quantifier

A proof of `exists x : A, P x` is a pair of:

- an object `t` of type `A`
- a proof of `(P t)`

This can be defined inductively:

```
Inductive ex (A:Type)(P: A -> Prop) :=
  | ex_intro : forall x : A, (P x) -> (ex A P).
```

A proof involving ∃

```
Inductive ex (A:Type)(P: A -> Prop) :=
  | ex_intro : forall x : A, (P x) -> (ex A P).
```

```
pq : forall x, P x -> Q x
```

Show : `(exists x, P x) -> (exists y, Q y)`

```
fun ep : exists x, P x =>
   match ep with
  | ex_intro z pz =>
        (ex_intro z (pq z pz))
  end.
```

The architecture of Coq

- The logical rules are typing rules : t : A
- The lemmas are constants : l1 := p : P
- The axioms are variables : a : P
- The type checker is a proof-checker
- The type checker is the critical part : the (only) one we need to trust
- Computations are used in the type-checker

Program extraction

We prove

```
Lemma div2 : forall n, exists p,
                        n = p+p \/ n = (S (p+p)).
```

Then we "execute" it:

div2 4   yields 2 and left (even)
div 7   yields 3 and right (odd)

Termination and coherence

Suppose we can define non-terminating functions, like:

```
Fixpoint F (a : A) : A := F a.
```

What is the problem ?

Two examples of proofs of `False`…

A first problem

```
Definition negb (x : bool) := match x with
    | true => false
    | false => true
end.

Fixpoint foo (x : bool) := negb (foo x).
```

```
(foo true) = negb (foo true)
```

$$\left.\begin{array}{l}\texttt{true} \;\;= \texttt{negb (true)}\\ \texttt{false} = \texttt{negb (false)}\end{array}\right\} \Rightarrow \boxed{\texttt{true = false}}$$

A second version

```
Fixpoint goo (x:True) : False := goo x.
```

```
goo : True -> False
```