Simply Typed λ-calculus and the Curry-Howard Correspondence

Recap: the λ-calculus

- Recall that the λ-calculus consists of λ-terms,

$$\Lambda := V \mid \Lambda\ \Lambda \mid (\lambda V.\Lambda)$$

- Anything that can be written in this way is a valid λ-term.
- We "run" a λ-term by β-reduction:

$$(\lambda x.N)\ M \to_\beta N[x := M]$$

- The λ-calculus has the same expressive power as a Turing machine: anything you can write on a computer can be written in λ-calculus.

- One "problem" with λ-calculus is that it permits writing programs that accomplish nothing:

$$(\lambda x.x\ x)\ (\lambda x.x\ x) \to_\beta (\lambda x.x\ x)\ (\lambda x.x\ x) \to_\beta \cdots$$

  This loops forever because it β-reduces to itself.

- This is not specific to λ-calculus. On a Turing machine you can write

```
while (1) { continue; }
```

which also accomplishes nothing.

- It would be pleasing to carve out a subset of λ-calculus where every program is guaranteed to terminate.

The simply typed λ-calculus

- Our approach: introduce a *type system* that rejects non-terminating programs.
- Unlike plain λ-calculus we now have two kinds of objects:
  - **Terms** — executable fragments, as before.
  - **Types** — a meta-language on top of terms that constrains which programs can be written.

We would like to force the following:

- Each variable x gets an associated type α
- every function must declare the type of its input. We write

$$\lambda(x : \alpha).M$$

  to mean "$\lambda x.M$, but x must have type α."
- Also MN is valid only if

$$M \equiv (\lambda.(x : \alpha).U)$$

- Clearly just defining a grammar for types and terms won't be enough to accomplish this, we will also need specific *rules* regarding term formation.
- Let's first review the grammar of types of terms and then get to the rules.

Grammar of types.

- If M has type β, then we declare that $\lambda(x : \alpha).M$ has type

$$\alpha \to \beta$$

by analogy with functions $f : \alpha \to \beta$ where we specify domain and codomain.

- The grammar of types is:

$$\mathbb{T} := \mathbb{V} \mid \mathbb{T} \to \mathbb{T}$$

where $\mathbb{V}$ is some initially agreed-upon set of *base types*.

- For example if $\mathbb{V} := \{\alpha\}$ then the available types are α, $\alpha \to \alpha$, $\alpha \to \alpha \to \alpha$, $(\alpha \to \alpha) \to \alpha$, and so on.

Grammar of terms

- The grammar of terms is:

$$\Lambda_{\mathbb{T}} := V \mid \Lambda_{\mathbb{T}}\ \Lambda_{\mathbb{T}} \mid (\lambda(V : \mathbb{T}).\Lambda_{\mathbb{T}})$$

- The only change from plain λ-calculus is the type annotation on the bound variable.
- We don't annotate the output type — it can be inferred.

Derivations

- A key difference with plain λ-calculus: not every grammatically correct term is valid.
- A term must be *derivable*: constructed by a finite sequence of typing rules. This sequence is called a **derivation**.

The three typing rules

1. **Variable:** if x has been assigned type σ, then $x : \sigma$.
2. **Application:** if $M : \alpha \to \beta$ and $N : \alpha$, then $M\ N : \beta$.
3. **Abstraction:** if, assuming $x : \alpha$, we can show $N : \beta$, then

$$\lambda(x : \alpha).\ N\ :\ \alpha \to \beta.$$

Contexts and the judgement notation

- The phrase "assuming $x : \alpha$" needs to be made precise. A **context** Γ is a finite list of variable–type assignments:

$$\Gamma \;=\; x_1 : A_1,\; x_2 : A_2,\; \ldots,\; x_n : A_n$$

  where the x_i are distinct.

- It records which free variables are in scope and what their types are.

- A **typing judgement** has the form $\Gamma \vdash M : A$, read "there is a derivation of the $M : A$ from the assumptions Γ."

- In this notation, the three rules become:

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}\ \text{Var} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}\ \text{App}$$

and

$$\frac{\Gamma,\, x : A \;\vdash\; M : B}{\Gamma \;\vdash\; \lambda(x : A).\, M : A \to B}\ \text{Abs}$$

Example: a typeable term

- We pick $\Gamma = \{x : \alpha, y : \beta\}$.
- By Rule 1 we have $x : \alpha, y : \beta \vdash y : \beta$.
- By Rule 3 we have $x : \alpha \vdash (\lambda(y : \beta).y) : \beta \to \beta$.
- Again by Rule 3,

$$\vdash (\lambda(x : \alpha).(\lambda(y : \beta).y) : \alpha \to (\beta \to \beta)$$

- Thus the term $\lambda(x : \alpha).(\lambda(y : \beta).y)$ is valid and has type $\alpha \to (\beta \to \beta)$.
- In fact this is the *only* term that inhabits $\alpha \to (\beta \to \beta)$.

Example: an untypeable term

- There is no way to type $x\ x$.
- We can derive this formally by rule inversion. The only rule that has something ressembling xx is rule 2 that has MN. In particular this means that if $\Gamma \vdash xx$ then

$$\Gamma \vdash x : A \to B \text{ and } \Gamma \vdash x : A$$

- The only rule that can give this type of conclusion is Var, inverting it we get $(x : A) \in \Gamma$ and $(x : A \to B) \in \Gamma$. Since we require only one type per variable in Γ this gives $A = A \to B$.
- But types are finite trees built from the grammar $\mathbb{T} := \mathbb{V} | \mathbb{T} \to \mathbb{T}$. The equation $A = A \to C$ has no solution.
- In particular $(\lambda x.x\ x)(\lambda x.x\ x)$ is untypeable because xx is untypeable, so the type system rejects this infinite loop.

Strong normalization

Theorem (Strong normalization)

Every typeable term in the simply typed λ-calculus is strongly normalizing: all reduction sequences terminate.

- We won't prove this today. It is a hard theorem.
- This is obviously beneficial: we have carved out a programming language in which every valid program is automatically terminating.

A mathematical perspective

- So far we have treated types as a tool for *programming*: they reject bad programs and guarantee termination.
- But there is a second way to look at the same system, not as a programming language, but as a *logic*.
- We will now set up this second reading carefully. We define a logical system independently, then prove it is the same system in disguise.
- This is an instance of the Curry-Howard correspondence.
- (The logical system we will see now is very basic, soon we will add more "bells and whistles" to it).

Natural deduction for IPL (implicational fragment)

- **Formulae:** $A, B ::= p \mid A \Rightarrow B$ where p ranges over propositional variables.
- **Judgement form:** $\Gamma \vdash A$, read "formulae A is provable from the formulae in assumptions Γ.". By provable we mean derivable using the rules which we specify in the next slide.
- Here $\Gamma = \{x_1 : A_1, \ldots, x_n : A_n\}$ is a finite set of **labelled assumptions**. Each A_i is a formula we are assuming to be true, and x_i is just a *name* for that assumption — a label so we can refer back to it later. (Compare: in a mathematical proof you might say "by hypothesis (H2)..." — the label x_i plays the role of "(H2)".)

- **Rules:**

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash A}\ \mathrm{Ax}$$

$$\frac{\Gamma, x : A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow \mathrm{I}^x \qquad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \Rightarrow \mathrm{E}$$

- Ax: if A is one of our assumptions, we can assert it.
- $\Rightarrow \mathrm{I}^x$: to prove $A \Rightarrow B$, temporarily assume A (calling this assumption x), prove B, then *discharge* the assumption. The superscript x records which assumption is being discharged.
- $\Rightarrow \mathrm{E}$ (modus ponens): from $A \Rightarrow B$ and A, conclude B.

Typing rules for STLC (implicational fragment)

- **Types:** $A, B ::= \alpha \mid A \to B$ where α ranges over base types.
- **Judgement form:** $\Gamma \vdash M : A$, where Γ is a typing context.
- **Rules:**

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}\ \mathsf{Var}$$

$$\frac{\Gamma,\, x : A \ \vdash\ M : B}{\Gamma \ \vdash\ \lambda(x : A).\, M \ :\ A \to B}\ \to\mathsf{I}$$

$$\frac{\Gamma \ \vdash\ M : A \to B \quad \Gamma \ \vdash\ N : A}{\Gamma \ \vdash\ M\ N \ :\ B}\ \to\mathsf{E}$$

- The two systems have the same shape. The only difference: the typing rules carry *terms*.

The correspondence, stated precisely

- Define a map $|-|$ ("erasure") from typing derivations to natural deduction proofs by *forgetting the terms*.
- Define a map $\mathcal{T}$ ("decoration") from natural deduction proofs to typing derivations by *adding terms* according to the rules.
- These maps will be uniquely defined given a derivation tree.
- The erasure map is obvious, the decoration one a little bit less.

Theorem (Curry-Howard, implicational fragment)

The maps $|-|$ *and* $\mathcal{T}$ *are mutually inverse bijections between:*

- *typing derivations* $\Gamma \vdash M : A$ *in STLC, and*
- *natural deduction derivations* $\Gamma \vdash A$ *in IPL,*

where Γ *is identified on both sides (same labels, same formulae/types).*

- In particular: A is provable in IPL if and only if the type A is inhabited in STLC.
- Moreover, there is a bijection between the set of proofs of $\Gamma \vdash A$ and the set of terms M such that $\Gamma \vdash M$.
- To a classically trained mathematician the second sentence sounds strange, we are so used to identifying all proofs and only thinking in terms of true and false!

Example: erasure in action

- Start with the typing derivation of $\lambda(x : A).\, x \ : \ A \to A$:

$$\dfrac{\dfrac{(x : A) \in \{x : A\}}{x : A \ \vdash \ x : A}\,\mathsf{Var}}{\vdash \ \lambda(x : A).\, x \ : \ A \to A}\ {\to}\mathsf{I}$$

- Apply erasure $|-|$. At each node, drop the term and keep the type:

$$\dfrac{\dfrac{(x : A) \in \{x : A\}}{x : A \ \vdash \ A}\,\mathsf{Ax}}{\vdash \ A \Rightarrow A}\ {\Rightarrow}\mathsf{I}^{x}$$

- We obtain a valid natural deduction proof of $A \Rightarrow A$. The tree has the same shape; only the term annotations have been removed.

Example: decoration in action

- Now go the other way. Start from the natural deduction proof of $A \Rightarrow A$:

$$\dfrac{\dfrac{(x : A) \in \{x : A\}}{x : A \vdash A}\,\mathsf{Ax}}{\vdash A \Rightarrow A}\,\Rightarrow\mathsf{I}^{x}$$

- Apply decoration $\mathcal{T}$. Bottom-up, the rules force the terms uniquely:
  - The Ax node uses assumption labelled x, so the term must be x.
  - The $\Rightarrow\mathsf{I}^{x}$ node discharges $x : A$ and produces $A \Rightarrow A$, so the term must be $\lambda(x : A).$ (whatever was above).
- Result:

$$\dfrac{\dfrac{(x : A) \in \{x : A\}}{x : A \vdash x : A}\,\mathsf{Var}}{\vdash \lambda(x : A).\, x \;:\; A \to A}\,\to\mathsf{I}$$

What the example shows

- Erasure and decoration are *mechanical* .
- The proof of the correspondence (next slides) simply verifies that this always works, for every derivation tree, by induction on the tree.

Proof (erasure is well-defined)

- We show that $|-|$ sends every typing derivation to a valid natural deduction proof. This is by induction on the typing derivation.
- **Case Var.** The derivation ends with

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}$$

Erasing the term x gives $\dfrac{(x : A) \in \Gamma}{\Gamma \vdash A}$ which is a valid application of Ax. ✓

- **Case $\to$I.** The derivation ends with

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda(x : A).\, M \;:\; A \to B}$$

  By induction, erasing the premise gives a valid proof of $\Gamma, x : A \vdash B$. Applying $\Rightarrow \mathsf{I}^x$ to this gives a valid proof of $\Gamma \vdash A \Rightarrow B$. ✓

- **Case $\to$E.** The derivation ends with

$$\frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}$$

  By induction, erasing the premises gives valid proofs of $\Gamma \vdash A \Rightarrow B$ and $\Gamma \vdash A$. Applying $\Rightarrow$E gives a valid proof of $\Gamma \vdash B$. ✓

Proof (decoration is well-defined)

- Conversely, we show that $\mathcal{T}$ sends every natural deduction proof to a valid typing derivation, again by induction.
- **Case Ax.** The proof ends with $\dfrac{(x : A) \in \Gamma}{\Gamma \vdash A}$. We decorate this as $\dfrac{(x : A) \in \Gamma}{\Gamma \vdash x : A}$, using the variable x as the term. Valid by Var. ✓

- **Case $\Rightarrow\mathsf{I}^x$.** The proof ends with

$$\frac{\Gamma, x : A \vdash B}{\Gamma \vdash A \Rightarrow B}$$

  By induction, decorating the premise gives $\Gamma, x : A \vdash M : B$ for some term M. We decorate the conclusion as $\Gamma \vdash \lambda(x : A).\, M : A \to B$. Valid by $\to\mathsf{I}$. ✓

- **Case $\Rightarrow\mathbf{E}$.** The proof ends with

$$\frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

  By induction, decorating the premises gives $\Gamma \vdash M : A \to B$ and $\Gamma \vdash N : A$. We decorate the conclusion as $\Gamma \vdash M\ N : B$. Valid by $\to\mathsf{E}$. ✓

Proof (the maps are inverse)

- $\mathcal{T} \circ |-|$ **is the identity on typing derivations.** Given a typing derivation, erase the terms, then re-decorate. At each step the decoration procedure recovers the same term: the variable rule recovers the variable name x, $\Rightarrow\mathsf{I}^x$ recovers $\lambda(x : A).\, M$, and $\Rightarrow\mathsf{E}$ recovers $M\ N$. (This uses the fact that the assumption labels in the proof match the variable names in the term — this is precisely why we label assumptions.)
- $|-| \circ \mathcal{T}$ **is the identity on proofs.** Given a proof, decorate it, then erase. The erasure simply removes what was added, returning the original proof.
- Hence the two maps are bijections. $\square$

Example: an unprovable proposition

- Is $(A \Rightarrow B) \Rightarrow A$ a theorem of IPL? It suffices to show it fails for one instance. Let $A = p$ and $B = q$ where p, q are distinct propositional variables (base types).
- By the correspondence, we ask: is $(p \to q) \to p$ inhabited? Suppose $\vdash M : (p \to q) \to p$. Inverting Abs:

$$\frac{f : p \to q \ \vdash \ N : p}{\vdash \ \lambda(f : p \to q).\, N \ : \ (p \to q) \to p}$$

  We need $f : p \to q \vdash N : p$. We show no such derivation exists. We check that none of the rules apply:

  - Var: the only variable is $f : p \to q \neq p$. $\times$
  - Abs: gives N a function type, but p is a base type. $\times$
  - App: then $f : p \to q \vdash N_1 : C \to p$ and $f : p \to q \vdash N_2 : C$. By Var, $N_1 = f$ forces $C = p$ and $q = p$, contradicting $p \neq q$. $\times$

- The type is uninhabited, so the formula is not a theorem. $\square$

The role of Howard's paper

- Curry observed the correspondance between provability and type inhabitation.
- Howard (1969, published 1980) extended this in two directions:
  1. he made the correspondence between *proofs* (not just provability) and *terms* (not just types) precise.
  2. he generalized from the implicational fragment of propositional logic to full propositional logic (including $\wedge$, $\vee$, $\neg$) and to predicate logic ($\forall$, $\exists$). The quantifiers correspond to dependent types.

Extending the type system

- We extend the system to propositional logic by adding products, sums, a unit type, and an empty type. New grammar for types:

$$A, B ::= \alpha \mid A \to B \mid A \times B \mid A + B \mid \mathbf{1} \mid \mathbf{0}$$

- $A \times B$: a pair (a, b) with $a : A$ and $b : B$
- $A + B$: a union type, inhabited by either an $a : A$ or a $b : B$.
- $\mathbf{1}$: the unit type with exactly one element $()$.
- $\mathbf{0}$: the empty type with no elements. No type inhabits it. Anything is typable if it can be inhabited.
- All types (except $\mathbf{1}$ and $\mathbf{0}$) follow the same pattern as $\to$: they have an introduction rule (how to build a value of that type) and an elimination rule (how to use one).

The extended term grammar

- The grammar of terms extends accordingly:

$$M, N ::= x \mid \lambda(x : A).\, M \mid M\ N$$
$$\mid\ (M, N) \mid \pi_1(M) \mid \pi_2(M)$$
$$\mid\ \mathsf{inl}(M) \mid \mathsf{inr}(M) \mid \mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}(x) \Rightarrow N_1 \mid \mathsf{inr}(y) \Rightarrow N_2$$
$$\mid\ () \mid \mathsf{abort}(M)$$

- (M, N), π_1, π_2: build and decompose pairs.
- $\mathsf{inl}(M)$, $\mathsf{inr}(M)$, case: build and decompose tagged unions.
- $()$: the unique inhabitant of $\mathbf{1}$.
- $\mathsf{abort}(M)$: from $M : \mathbf{0}$, produce anything.

Additional Typing rules for the extended system (I)

Products:

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A \times B}\ \times\mathsf{I} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1(M) : A}\ \times\mathsf{E}_1 \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2(M) : B}\ \times\mathsf{E}$$

Sums:

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathsf{inl}(M) : A + B}\ +\mathsf{I}_1 \qquad \frac{\Gamma \vdash M : B}{\Gamma \vdash \mathsf{inr}(M) : A + B}\ +\mathsf{I}_2$$

$$\frac{\Gamma \vdash M : A + B \quad \Gamma, x : A \vdash N_1 : C \quad \Gamma, y : B \vdash N_2 : C}{\Gamma \vdash \mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}(x) \Rightarrow N_1 \mid \mathsf{inr}(y) \Rightarrow N_2\ :\ C}\ +\mathsf{E}$$

Additional Typing rules for the extended system (II)

Unit:

$$\frac{}{\Gamma \vdash () : \mathbf{1}}\ \mathbf{1}\mathsf{I}$$

No elimination rule — there is nothing useful to extract from ().

Empty:

$$\frac{\Gamma \vdash M : \mathbf{0}}{\Gamma \vdash \mathsf{abort}(M) : C}\ \mathbf{0}\mathsf{E}$$

No introduction rule — there is no way to produce an element of $\mathbf{0}$.

- In each case, the pattern is the same: introductions build the type, eliminations take it apart. Compare with $\to$I (λ-abstraction) and $\to$E (application).

Computation rules

- When an elimination meets its corresponding introduction, they cancel:

$$(\lambda(x : A).\, M)\ N \longrightarrow M[x := N]$$

$$\pi_1(M, N) \longrightarrow M \qquad \pi_2(M, N) \longrightarrow N$$

$$\text{case inl}(M) \text{ of inl}(x) \Rightarrow N_1 \mid \text{inr}(y) \Rightarrow N_2 \longrightarrow N_1[x := M]$$

$$\text{case inr}(M) \text{ of inl}(x) \Rightarrow N_1 \mid \text{inr}(y) \Rightarrow N_2 \longrightarrow N_2[y := M]$$

- There is no rule for abort: since $\mathbf{0}$ has no introduction, abort is never applied to a value.
- Under Curry-Howard, each of these is a proof simplification: an introduction immediately followed by its elimination is a detour, and the reduction removes it.

Intuitionistic propositional logic

- **Formulae:**

$$P, Q ::= p \mid P \Rightarrow Q \mid P \wedge Q \mid P \vee Q \mid \top \mid \bot$$

  where p ranges over propositional variables.

- This is the same grammar as our type system, with different notation:

Type	Connective
$A \to B$	$A \Rightarrow B$ (implication)
$A \times B$	$A \wedge B$ (conjunction)
$A + B$	$A \vee B$ (disjunction)
$\mathbf{1}$	$\top$ (truth)
$\mathbf{0}$	$\bot$ (falsity)

Natural deduction rules for IPL

Implication:

$$\frac{\Gamma, x : A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow \mathsf{I}^x \qquad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \Rightarrow \mathsf{E}$$

Conjunction:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge \mathsf{I} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge \mathsf{E}_1 \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge \mathsf{E}_2$$

Disjunction:

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee \mathsf{I}_1 \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee \mathsf{I}_2 \qquad \frac{\Gamma \vdash A \vee B \quad \Gamma, x : A \vdash C \quad \Gamma,}{\Gamma \vdash C}$$

Truth:

$$\frac{}{\Gamma \vdash \top}\ \top\mathsf{I}$$

Falsity:

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash C}\ \bot\mathsf{E}$$

- These are exactly the typing rules from the previous slides, with the terms erased. Each typing rule carries a term; each logical rule does not.

How IPL differs from classical logic

- Classical propositional logic has one axiom that IPL lacks: the **law of excluded middle**,

$$A \vee \neg A$$

  which asserts that every proposition is either true or false.

- Equivalently, IPL lacks **double negation elimination** $\neg\neg A \Rightarrow A$, or **Peirce's law** $((A \Rightarrow B) \Rightarrow A) \Rightarrow A$. Adding any one of these recovers classical logic.

- A consequence: **truth tables do not work for IPL.**

The Curry-Howard correspondence (full statement)

Theorem (Curry-Howard)

The erasure and decoration maps extend to mutually inverse bijections between:

- *typing derivations* $\Gamma \vdash M : A$ *in the simply typed* λ*-calculus with products, sums,* $\mathbf{1}$*, and* $\mathbf{0}$*, and*
- *natural deduction derivations* $\Gamma \vdash A$ *in intuitionistic propositional logic.*

- There is a bijection between the set of proofs of $\Gamma \vdash A$ and the set of terms M such that $\Gamma \vdash M : A$.
- In particular: a proposition A is provable in IPL if and only if the type A is inhabited.
- The proof is identical in structure to the one we gave for the implicational fragment: one induction case per rule, with no new ideas. Each new connective adds one more case to the erasure direction and one more case to the decoration direction.

Truth in intuitionistic logic

- In classical logic, every proposition is either true or false, period. The truth value is fixed, even if we don't know it. A proof is something that *reveals* a pre-existing fact.
- In intuitionistic logic, truth means something different: a proposition is true only when we can **construct a witness** for it. Truth is thus tied to *construction* or *inhabitation*. There is a beautiful quote of Brouwer to that effect:

  "*There are no non-experienced truths.*"

  — L.E.J. Brouwer

- According to Brouwer, for something to be true you have to "experience it'' in your own mind, you have to feel your mind construct and establish the truth of the fact.
- So when a newspaper tells you that so and so (that you never met) did so and so on so and so continent (where you've never been) and you have no mental model for even ascertaining the claim... why do we accept it as truth?

Brouwer and Schopenhauer

- Brouwer was largely inspired by Schopenhauer:

> *"The world is my representation": this is a truth valid with reference to every living and knowing being, although man alone can bring it into reflective, abstract consciousness. If he really does so, philosophical discernment has dawned on him. It then becomes clear and certain to him that he does not know a sun and an earth, but only an eye that sees a sun, a hand that feels an earth; that the world around him is there only as representation, i.e., only in reference to another thing, namely that which represents, and this is himself.*
>
> — Schopenhauer, *The World as Will and Representation*

Truth and representation

- Schopenhauer is not promoting here some sort of post-modernism in the sense of "everything goes", "there is no truth", neither is he claiming that "Platonism is false", he is rather doing something more subtle. He makes the obvious undebatable claim that our only access to the world is through representations, our senses, our mind, in any case through intermediaries.
- So truth, if it is to mean anything to us, must be grounded in what we can actually construct and verify using those intermediaries, not in what might lie behind the curtain. This is why Brouwer insists: "there are no non-experienced truths.''

An analogy

- Consider the proposition: "there is a small rock at coordinates (x, y) near Alpha Centauri right now.'' If you studied a bit of special relativity you know that this is *unknowable in principle*. The event is spacelike-separated from us: no causal chain could ever connect us to the answer.
- The classical logician insists the proposition is still either true or false, but this requires asserting a truth value that no possible observer could ever access. At that point, why even make the claim that this is either true or false?
- Brouwer's argument is analogous: if no construction can reach a proposition, asserting it has a truth value is metaphysics, not mathematics.

The Kantian roots

- Schopenhauer (and then Brouwer) is developing a Kantian point of view. Kant's central insight is that the "thing in itself' ' is inaccessible — we only have knowledge of things as constructed through intermediaries, those being the senses and the mind.
- For example, in intuitionistic logic $A \vee \neg A$ cannot be asserted for any proposition A. To assert a disjunction, we must know *which* side holds and provide evidence for that side. For an arbitrary proposition A there is no uniform way to know (e.g in the same way that the halting problem is undecidable).
- However $\neg\neg(A \vee \neg A)$ is true in intuitionistic logic, i.e "we have no grounds to assert it, but we know it cannot be refuted".
- Just as Kant doesn't deny the thing-in-itself exists, he rather says that we can't access it, the intuitionist doesn't deny excluded middle, he says we can't construct it.