Lecture 6: From STLC to Intuitionistic Propositional Logic

Recap: two systems in parallel

- We have been developing two things in parallel:
  1. The **simply typed λ-calculus** (STLC): a version of the λ-calculus where each function has a defined domain and range.
  2. A simple **logical system** with atomic propositions and implication.
- Last time we proved the Curry-Howard isomorphism for this case: these two systems are the same thing.

- Let's recall how they are set up.
- In the simply typed λ-calculus, the types are given by the grammar

$$\mathbb{T} = \alpha \mid \mathbb{T} \to \mathbb{T}$$

  where α ranges over atomic types.
- The terms are given by

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

STLC typing rules

- A term is well-typed if we can derive a typing judgement $\Gamma \vdash M : A$ using these three rules:

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

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

Example: $\lambda(x : \alpha).\, x : \alpha \to \alpha$

- To show this is a valid typing assignment, we build a derivation tree:

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

- Every well-typed assignment corresponds to such a tree built from the three rules.

The simply typed λ-calculus

- This is the simply typed λ-calculus.
- The key theorem (which we stated but have not yet proved): **every well-typed term is strongly normalizable.** That is, every sequence of β-reductions terminates.
- The price: we lose expressiveness. But we gain termination.

The other side of the picture

- Now let's look at the logical system. We consider a system even more basic than full propositional logic.
- In this system we only have: atomic propositions, implications between them, and valid rules of deduction.
- The formulas are given by the grammar

$$\mathbb{F} = p \mid \mathbb{F} \Rightarrow \mathbb{F}$$

where p ranges over propositional variables.

Natural deduction rules (implicational fragment)

- A formula is provable from assumptions Γ if we can derive $\Gamma \vdash A$ using:

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

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

Example: $A \Rightarrow A$

- Here is a proof that $A \Rightarrow A$ regardless of what A is:

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

- Compare this with the typing derivation of $\lambda(x : \alpha).\, x : \alpha \to \alpha$. They are the same tree.

The Curry-Howard isomorphism (implicational fragment)

- The Curry-Howard isomorphism is the observation that these two systems are the same thing.
- It establishes a direct correspondence between:
  - Types A in STLC $\longleftrightarrow$ formulas A in the logic.
  - Terms M of type A $\longleftrightarrow$ proofs of proposition A.

The proof strategy: two maps

- The proof of the Curry-Howard correspondence goes through the construction of two maps between derivations.
- The **erasure map** $|-|$: takes a valid typing derivation $\Gamma \vdash M : A$ and *forgets the terms*, producing a valid logical derivation $\Gamma \vdash A$.
- The **decoration map** $\mathcal{T}$: takes a valid logical derivation $\Gamma \vdash A$ and *adds terms*, producing a valid typing derivation $\Gamma \vdash M : A$.

- At each node of the derivation tree, erasure strips out the term and keeps the type/proposition. Decoration does the reverse: it reads off which rule was applied and fills in the unique term that rule demands.
- These two maps go in opposite directions between the same two sets — the set of typing derivations and the set of logical derivations.
- The theorem is that these two maps are inverses of each other. Therefore there is a **bijection** between typing derivations and logical proofs.

Precise statement

Theorem (Curry-Howard, implicational fragment)

The maps $|-|$ (erasure) and $\mathcal{T}$ (decoration) are mutually inverse bijections between:

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

In particular: a type A is inhabited if and only if the corresponding proposition A is provable.

- We proved this last time. The proof goes by induction: each typing rule maps to the corresponding logical rule under erasure, and vice versa under decoration.

Extending the system

- Now we would like to extend both systems to something richer.
- The first target: a logic that can express all simple propositions involving **and**, **or**, **true**, **false**, **negation**, and **implication**.
- This is the least we could ask for. It is full **intuitionistic propositional logic** (IPL).

The correspondence we have in mind

Logic	Type Theory
Proposition	Type
Proof of A	Term of type A
$A \Rightarrow B$ (implication)	$A \to B$ (function type)
$A \wedge B$ (conjunction)	$A \times B$ (product type)
$A \vee B$ (disjunction)	$A + B$ (sum type)
$\top$ (truth)	$\mathbf{1}$ (unit type)
$\bot$ (falsehood)	$\mathbf{0}$ (empty type)
$\neg A$ (negation)	$A \to \mathbf{0}$

The guiding principle

- Since there will be a version of Curry-Howard, we think of **terms as proofs** and **types as propositions**.
- A function from A to B is a function that takes proofs of A into proofs of B.
- The logic we develop is oriented around the notion of **exhibiting a proof**. Truth means having a construction.
- This will differ from classical logic. Let's see how.

Deriving the rules from Curry-Howard

- Let's imagine we already have the Curry-Howard correspondence and try to derive what the logical/typing rules must be.
- Each type former comes with **introduction rules** (how to build a term of that type) and **elimination rules** (how to use a term of that type).
- Under Curry-Howard, introduction = how to prove a proposition, elimination = how to use a proposition.

Why introduction rules?

- If a type has no introduction rule, we can never construct a term of that type. The type would be permanently empty — useless as a data structure, and unprovable as a proposition.
- Every type that we want to be inhabitable *must* have at least one introduction rule. This is how values come into existence.

Why elimination rules?

- The necessity of elimination rules is less obvious, but equally important.
- Without elimination rules, once we have a value of some compound type, we cannot take it apart or extract information from it. Data structures would be **write-only**: we could build them but never inspect them.

Elimination rules in practice

- In OCaml, pattern matching on a sum type is an elimination:

```
case x of
    Left a -> f a
    Right b -> g b
```

- Projecting from a tuple is an elimination:

```
let (a, b) = pair in a
```

- In Rust:

```
match result {
    Ok(val)  => process(val),
    Err(e)   => handle(e),
}
```

- Without these operations, compound types would be black boxes.

Computation as proof simplification

- Notice what happens when an introduction is immediately followed by the corresponding elimination. For example: we build a pair (M, N) and then immediately project $\pi_1(M, N)$. The result is just M — the construction was redundant.
- This is exactly what β-reduction does: it eliminates these redundant intro-elim pairs.

$$\pi_1(M, N) \longrightarrow M \qquad (\lambda(x : A).\, M)\ N \longrightarrow M[N/x]$$

- Under Curry-Howard, this has a beautiful logical reading: an introduction followed by its elimination is a **detour** in a proof. Reduction removes the detour. **Computation is proof simplification.**

Truth and falsehood are special

- Most types have both introduction and elimination rules. But two types are degenerate:
- $\mathbf{1}$ (truth) has an introduction rule but **no elimination rule**. We can always construct it, and once something is true we cannot make it untrue — so there is no rule to "undo" it.
- $\mathbf{0}$ (falsehood) has an elimination rule but **no introduction rule**. We can never construct it — but *if* we had a term of this type, we could derive anything (*ex falso*).
- These are the two extremes. Every other type has both introduction and elimination rules.

Now let's write down the rules

- For each connective/type we give the typing rule and the corresponding logical rule **side by side**.
- Notice: they are the same rule. The typing version carries a term; the logical version just records the proposition.

Implication / function type: introduction

Typing rule ($\rightarrow$I):

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

Logical rule ($\Rightarrow$I):

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

- Think of a function $f : A \to B$ as a mechanism that takes proofs of A and transforms them into proofs of B. If such a mechanism exists, then we can assert $A \Rightarrow B$ — we have a concrete way to turn any evidence for A into evidence for B.
- Note: if A is false (has no proofs), the function $f : A \to B$ still exists — it just never gets the chance to be used. There are no inputs to feed it. This is consistent with classical logic, where $A \Rightarrow B$ is valid whenever A is false.

Implication / function type: elimination

Typing rule ($\to$E):

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

Logical rule ($\Rightarrow$E, modus ponens):

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

From a proof of $A \Rightarrow B$ and a proof of A, obtain a proof of B. This is clear: if we have a proof of A, and we have a way to transform proofs of A into proofs of B, then we also have a proof of B.

Conjunction / product type: introduction

Typing rule ($\times$I):

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A \times B} \times\mathsf{I}$$

Logical rule ($\wedge$I):

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge\mathsf{I}$$

To prove $A \wedge B$, prove both A and B. To build a pair, provide both components.

Conjunction / product type: elimination

Typing rules ($\times$E):

$$\frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1(M) : A}\ \times\mathsf{E}_1 \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2(M) : B}\ \times\mathsf{E}_2$$

Logical rules ($\wedge$E):

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

From a proof of $A \wedge B$, extract a proof of either side. From a pair, project either component.

Disjunction / sum type: introduction

Typing rules ($+\mathbf{I}$):

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

Logical rules ($\vee\mathbf{I}$):

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee\mathbf{I}_1 \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee\mathbf{I}_2$$

To prove $A \vee B$, prove one side and say which. To build a sum, inject into one side.

Disjunction / sum type: elimination

Typing rule (+E):

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

Logical rule ($\vee$E):

$$\frac{\Gamma \vdash A \vee B \quad \Gamma,\, x : A \vdash C \quad \Gamma,\, y : B \vdash C}{\Gamma \vdash C}\ \vee\mathsf{E}$$

- To pattern match on a sum type, you must handle both possibilities and return something of the same type C in each branch.
- The logical reading is the same: if you want to deduce C from $A \vee B$, you need to make sure that both $A \Rightarrow C$ and $B \Rightarrow C$. Since one of A or B holds (but you don't know which), you must be prepared for either case.

Truth / unit type

Typing rule (1I):

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

Logical rule ($\top$I):

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

Truth ($\top$) can always be asserted — it requires no assumptions and no evidence. The unit type $\mathbf{1}$ is always inhabitable.

No elimination rule. Once something is true, you cannot make it not true. Truth is preserved but yields nothing new — there is no information to extract from it, and no useful way to deconstruct it.

Falsehood / empty type

Typing rule (0E):

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

Logical rule ($\bot$E, *ex falso quodlibet*):

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

From a proof of falsehood, derive anything (*ex falso quodlibet*). **No introduction rule** — there is no way to prove $\bot$, no way to construct a term of type $\mathbf{0}$. This is the whole point: since our notion of truth is tied to construction, falsehood is precisely that which cannot be constructed.

Negation

- How do we say "not A" in this system?
- The basic principle: we should not be able to assert both A and $\neg A$ simultaneously. If we could, we should be able to derive falsehood.
- So define $\neg A := A \to \mathbf{0}$.

- Why does this work? If we have both a proof of A and a proof of $\neg A$, then by the application rule ($\to$E) we can combine them:

$$\frac{\Gamma \vdash f : A \to \mathbf{0} \quad \Gamma \vdash a : A}{\Gamma \vdash f\ a : \mathbf{0}} \to \mathrm{E}$$

- That is: from $\neg A$ and A together, we derive $\mathbf{0}$ (falsehood). This is exactly what we wanted.
- This matches mathematical practice: to show $\neg A$, assume A and derive a contradiction.

Getting a feel for the logic

- This logic looks similar to classical logic but it is not. The key difference is the **meaning of truth**: a proposition is true only if we can construct a proof.
- Let's test our intuitions.

Can we assert $A \vee \neg A$?

- $A \vee \neg A$ corresponds to the type $A + (A \to \mathbf{0})$.
- To inhabit this type, we must choose a side:
  - Either produce $\mathsf{inl}(a)$ where $a : A$, or
  - produce $\mathsf{inr}(f)$ where $f : A \to \mathbf{0}$.
- But A is an arbitrary proposition. We don't know which case holds. We cannot decide.
- Consider: "the Riemann Hypothesis is either true or false." Classically this is a tautology. But to assert it constructively, you would have to say *which one* and provide a proof. We cannot do that.

Double negation elimination

- Another important point of divergence: in classical logic, $\neg\neg A \Rightarrow A$.
- The type $((A \to \mathbf{0}) \to \mathbf{0}) \to A$ asks: given a proof that A is irrefutable, produce a proof of A.
- But knowing "there is no way to deny A" does not hand us an actual construction of A. We are stuck.
- The inability to refute something does not mean it is true.

But we can prove $\neg\neg(A \vee \neg A)$

- Although we cannot assert $A \vee \neg A$, we *can* prove that there is no way to deny it.
- We need to inhabit the type $((A + (A \to \mathbf{0})) \to \mathbf{0}) \to \mathbf{0}$.

- **Proof.** Suppose $h : (A + (A \to \mathbf{0})) \to \mathbf{0}$.
- Define $f := \lambda(a : A).\ h(\mathsf{inl}(a))$.
- Then $f : A \to \mathbf{0}$, i.e. f is a proof of $\neg A$.
- So $\mathsf{inr}(f) : A + (A \to \mathbf{0})$.
- Apply h: we get $h(\mathsf{inr}(f)) : \mathbf{0}$. Contradiction.

- The full term witnessing this is:

$$\lambda(h : (A + (A \to \mathbf{0})) \to \mathbf{0}).\ h(\mathsf{inr}(\lambda(a : A).\ h(\mathsf{inl}(a))))$$

- So we can prove: "there is no way to deny excluded middle." But we cannot assert excluded middle itself.
- This is a way of embedding classical reasoning into constructive logic: classical tautologies are provable under a double negation translation.

The extended type grammar

- Now let's write down the full type system precisely.
- Types:

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

The extended term grammar

- Terms:

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

- Each construct is justified by its typing rule.

All typing rules collected (I)

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

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

All typing rules collected (II)

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

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

$$\frac{}{\Gamma \vdash () : \mathbf{1}}\ \mathbf{1}\mathsf{I} \qquad \frac{\Gamma \vdash M : \mathbf{0}}{\Gamma \vdash \mathsf{abort}(M) : C}\ \mathbf{0}\mathsf{E}$$

Computation rules

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

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

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

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

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

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

The extended Curry-Howard correspondence

Theorem (Curry-Howard, full IPL)

There is a bijection (via erasure and decoration) between typing derivations $\Gamma \vdash M : A$ *in the extended STLC and natural deduction proofs* $\Gamma \vdash A$ *in intuitionistic propositional logic.*

- The proof is the same as before: define erasure (drop terms) and decoration (add terms), show they are inverses by induction on derivations.
- But now we have more rules to check. Let's go through the argument.

Erasure is well-defined (I)

- We must show that erasing terms from any typing derivation gives a valid logical derivation. By induction on the typing derivation.
- **Case Var.** The derivation ends with

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

Erase the term x:

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

Valid. ✓

Erasure is well-defined (II)

- **Case $\times\mathbf{I}$.** The derivation ends with

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A \times B}$$

  By induction, erasing the premises gives valid proofs of $\Gamma \vdash A$ and $\Gamma \vdash B$. Applying $\wedge\mathbf{I}$:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge\mathbf{I}$$

  Valid. ✓

Erasure is well-defined (III)

- **Case $+$E.** The derivation ends with

$$\frac{\Gamma \vdash M : A + B \quad \Gamma, x : A \vdash N_1 : C \quad \Gamma, y : B \vdash N_2 : C}{\Gamma \vdash \mathsf{case}\ M\ \ldots : C}$$

  By induction, erasing gives valid proofs of $\Gamma \vdash A \vee B$, $\Gamma, x : A \vdash C$, and $\Gamma, y : B \vdash C$. Applying $\vee$E:

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, x : A \vdash C \quad \Gamma, y : B \vdash C}{\Gamma \vdash C}\ \vee\mathsf{E}$$

  Valid. ✓

- The cases $\to$I, $\to$E, $\times\mathsf{E}_1$, $\times\mathsf{E}_2$, $+\mathsf{I}_1$, $+\mathsf{I}_2$, $\mathbf{1}$I, $\mathbf{0}$E are all the same pattern.

Decoration is well-defined

- Conversely, given a logical derivation, we add terms to produce a typing derivation. By induction on the proof.
- **Case Ax.** The proof uses assumption labelled $x : A$. Decorate with the variable x: the typing derivation is $\Gamma \vdash x : A$ by Var. ✓
- **Case $\wedge$I.** The proof ends with $\Gamma \vdash A \wedge B$ from proofs of A and B. By induction, decorating gives $\Gamma \vdash M : A$ and $\Gamma \vdash N : B$. Decorate the conclusion as $\Gamma \vdash (M, N) : A \times B$. Valid by $\times$I. ✓

- **Case $\vee$E.** The proof ends with $\Gamma \vdash C$ from $\Gamma \vdash A \vee B$, $\Gamma, x : A \vdash C$, $\Gamma, y : B \vdash C$. By induction, decorating gives $\Gamma \vdash M : A + B$, $\Gamma, x : A \vdash N_1 : C$, $\Gamma, y : B \vdash N_2 : C$. Decorate the conclusion as

  $$\Gamma \vdash \mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}(x) \Rightarrow N_1 \mid \mathsf{inr}(y) \Rightarrow N_2 : C$$

  Valid by $+$E. ✓

- Every other case follows the same pattern: the logical rule determines a unique term constructor.

The maps are inverse

- $\mathcal{T} \circ |-|$ **is the identity on typing derivations.** Erase the terms, then re-decorate. At each step the decoration recovers the original term: Ax recovers the variable name, $\wedge$I recovers the pairing, $\vee$E recovers the case expression, and so on. This works because the assumption labels in the proof match the variable names in the term.
- $|-| \circ \mathcal{T}$ **is the identity on proofs.** Decorate a proof, then erase. The erasure removes exactly what was added.
- Hence the two maps are mutually inverse bijections. $\square$

Strong normalization and decidability

- Recall the theorem (which we have not yet proved): every well-typed term in the extended STLC is strongly normalizable.
- What does this mean for the logic?

- **Claim:** Strong normalization implies that intuitionistic propositional logic is decidable. Given any proposition A, we can determine whether it is provable or not.

- **Step 1.** By strong normalization, if A has any proof at all, it has one in **normal form**. (Take any proof and normalize it — strong normalization guarantees this terminates.)
- **Step 2.** A normal proof has the **subformula property**: every formula appearing in it is a subformula of A or of the assumptions in Γ.

- **Step 3.** So in particular there exists a derivation where every formula appearing above a given node is strictly shorter than the formula at that node — even when the proof branches like a tree.
- **Step 4.** Since formulas have finite size, every branch of the proof tree has bounded depth. Since finitely many rules can be applied at each node, the search tree is finite. So we can exhaustively search it: either we find a proof or we confirm none exists.

Summary

- We extended STLC with products ($\times$), sums ($+$), unit ($\mathbf{1}$), and empty ($\mathbf{0}$).
- Under Curry-Howard, this corresponds to full intuitionistic propositional logic.
- The logic is **constructive**: truth means having a witness. Excluded middle and double negation elimination fail. But $\neg\neg(A \vee \neg A)$ is provable.
- Strong normalization of the type system implies decidability of the logic.

Next time

- We will prove the strong normalization theorem (Tait's method / logical relations).
- We will also look at what happens when we add polymorphism (System F) — and see how Curry-Howard extends to second-order logic.