MAU11602
Lecture 11
2026-02-12

Type derivations and proofs

The function `fn x => fn y => (x, y)`, the curried version of an ordered pair constructor, has polymorphic type σ `->` τ `->` σ `*` τ.
We can prove this using our typing rules and the following diagram:

$$\dfrac{\dfrac{\dfrac{\texttt{x}\colon\sigma,\ \texttt{y}\colon\tau \vdash \texttt{x}\colon\sigma \qquad \texttt{x}\colon\sigma,\ \texttt{y}\colon\tau \vdash \texttt{y}\colon\tau}{\texttt{x}\colon\sigma,\ \texttt{y}\colon\tau \vdash \texttt{(x, y)}\colon\sigma\ \texttt{*}\ \tau}}{\texttt{x}\colon\sigma \vdash \texttt{fn y => (x, y)}\colon\tau\ \texttt{->}\ \sigma\ \texttt{*}\ \tau}}{\vdash \texttt{fn x => fn y => (x, y)}\colon\sigma\ \texttt{->}\ \tau\ \texttt{->}\ \sigma\ \texttt{*}\ \tau}$$

If we remove the expressions from the diagram, leaving only the types, then we are left with something very similar to our diagrammatic proof of the tautology $p \to q \to p \wedge q$:

$$\dfrac{\dfrac{\dfrac{\sigma, \tau \vdash \sigma \qquad \sigma, \tau \vdash \tau}{\sigma, \tau \vdash \sigma\ \texttt{*}\ \tau}}{\sigma \vdash \tau\ \texttt{->}\ \sigma\ \texttt{*}\ \tau}}{\vdash \sigma\ \texttt{->}\ \tau\ \texttt{->}\ \sigma\ \texttt{*}\ \tau} \qquad \dfrac{\dfrac{\dfrac{p, q \vdash p \qquad p, q \vdash q}{p, q \vdash p \wedge q}}{p \vdash q \to p \wedge q}}{\vdash p \to q \to p \wedge q}$$

Types and proofs

To translate statements to types we replace Boolean variables with type variables and replace $\wedge$, $\vee$ and $\rightarrow$ with `*`, `+` and `->`, respectively.
In the example, taking a typing derivation of an expression with type corresponding to a given statement and erasing all expressions, i.e. everything to the left of a `:`, gave us a proof of the proposition.
Does this work in general?
Let's try another example. $\vee$ should be commutative. In other words, $p \vee q \rightarrow q \vee p$ should be a tautology. Is it?
The type corresponding to this proposition is σ `+` τ `->` τ `+` σ, i.e. a function which takes a tagged union and gives us another tagged union with the roles of left and right reversed.
Can we write down such a function?
Yes. `fn z => case z of x => right x | y => left y end` works.

Verification

I claimed `fn z => case z of x => right x | y => left y end` works, i.e. has type σ `+` τ `->` τ `+` σ, but does it?
A typing derivation is

$$\dfrac{\dfrac{\mathtt{z}:\sigma\ \mathtt{+}\ \tau \vdash \mathtt{z}:\sigma\ \mathtt{+}\ \tau \qquad \dfrac{\dfrac{\mathtt{x}:\sigma \vdash \mathtt{x}:\sigma}{\mathtt{x}:\sigma \vdash \mathtt{right\ x}:\tau\ \mathtt{+}\ \sigma}}{\mathtt{z}:\sigma\ \mathtt{+}\ \tau, \mathtt{x}:\sigma \vdash \mathtt{right\ x}:\tau\ \mathtt{+}\ \sigma} \qquad \dfrac{\dfrac{\mathtt{y}:\tau \vdash \mathtt{y}:\tau}{\mathtt{y}:\tau \vdash \mathtt{left\ y}:\tau\ \mathtt{+}}}{\mathtt{z}:\sigma\ \mathtt{+}\ \tau, \mathtt{y}:\tau \vdash \mathtt{left\ y}:}}{\mathtt{z}:\sigma\ \mathtt{+}\ \tau \vdash \mathtt{case\ z\ of\ x => right\ x\ |\ y => left\ y\ end}:\tau\ \mathtt{+}\ \sigma}}{\vdash \mathtt{fn\ z => case\ z\ of\ x => right\ x\ |\ y => left\ y\ end}:\sigma\ \mathtt{+}\ \tau\ \mathtt{->}\ \tau\ \mathtt{+}\ \sigma}$$

so yes.

Verification, continued

Erasing the expressions from the typing derivation gives

$$\dfrac{\dfrac{\sigma \vdash \sigma}{\sigma \vdash \tau \ \texttt{+}\ \sigma}}{\dfrac{\sigma \ \texttt{+}\ \tau \vdash \sigma \ \texttt{+}\ \tau}{\sigma \ \texttt{+}\ \tau, \sigma \vdash \tau \ \texttt{+}\ \sigma}} \qquad \dfrac{\dfrac{\tau \vdash \tau}{\tau \vdash \tau \ \texttt{+}\ \sigma}}{\sigma \ \texttt{+}\ \tau, \tau \vdash \tau \ \texttt{+}\ \sigma}$$

$$\dfrac{\sigma \ \texttt{+}\ \tau \vdash \tau \ \texttt{+}\ \sigma}{\vdash \sigma \ \texttt{+}\ \tau \ \texttt{->}\ \tau \ \texttt{+}\ \sigma}$$

The logical counterpart would be

$$\dfrac{\dfrac{\dfrac{p \vdash p}{p \vdash q \vee p}}{p \vee q, p \vdash q \vee p} \qquad \dfrac{\dfrac{q \vdash q}{q \vdash q \vee p}}{p \vee q, q \vdash q \vee p}}{\dfrac{p \vee q \vdash q \vee p}{\vdash p \vee q \to q \vee p}}$$

Every step conforms to our rules of inference, so this is indeed a valid proof.

Why this works

The typing rules for the minimalist version of our language, with ordered pairs, tagged unions and functions, correspond to our rules of inference for zeroeth order logic.
The typing rules for ordered pairs are

$$\frac{\Gamma \vdash e_1 : \sigma \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash (e_1, e_2) : \sigma \ \texttt{*} \ \tau} \qquad \frac{\Gamma \vdash e : \sigma \ \texttt{*} \ \tau}{\Gamma \vdash \texttt{\#1} \ e : \sigma} \qquad \frac{\Gamma \vdash e : \sigma \ \texttt{*} \ \tau}{\Gamma \vdash \texttt{\#2} \ e : \tau}$$

The rules of inference for $\wedge$ were

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q} \qquad \frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash P} \qquad \frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash Q}$$

The typing rules for functions and the rules of inference for $\rightarrow$ were

$$\frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \texttt{fn} \ x \ \texttt{=>} \ e : \sigma \ \texttt{->} \ \tau} \qquad \frac{\Gamma \vdash e_2 : \sigma \qquad \Gamma \vdash e_1 : \sigma \ \texttt{->} \ \tau}{\Gamma \vdash e_1 \, e_2 : \tau}$$

$$\frac{\Gamma, P \vdash Q}{\Gamma \vdash P \rightarrow Q} \qquad \frac{\Gamma \vdash P \qquad \Gamma \vdash P \rightarrow Q}{\Gamma \vdash Q}$$

Why this works, continued

The typing rules for tagged unions were

$$\frac{\Gamma \vdash e : \sigma}{\Gamma \vdash \texttt{left}\ e : \sigma\ \texttt{+}\ \tau} \qquad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \texttt{right}\ e : \sigma\ \texttt{+}\ \tau}$$

$$\frac{\Gamma \vdash z : \sigma\ \texttt{+}\ \tau \qquad \Gamma, x : \sigma \vdash e_1 : \rho \qquad \Gamma, y : \tau \vdash e_2 : \rho}{\Gamma \vdash \texttt{case}\ z\ \texttt{of}\ x\ \texttt{=>}\ e_1\ \texttt{|}\ y\ \texttt{=>}\ e_2\ \texttt{end} : \rho}$$

The rules of inference for $\vee$ were

$$\frac{\Gamma \vdash P}{\Gamma \vdash P \vee Q} \qquad \frac{\Gamma \vdash Q}{\Gamma \vdash P \vee Q} \qquad \frac{\Gamma \vdash P \vee Q \qquad \Gamma, P \vdash R \qquad \Gamma, Q \vdash R}{\Gamma \vdash R}$$

One more example

Consider the expression `fn x => fn k => k x`. What is its type?
The following diagram shows that it is polymorphic of type σ `->` $(\sigma$ `->` $\tau)$ `->` τ

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathtt{k}\colon \sigma \texttt{ -> } \tau \vdash \mathtt{k}\colon \sigma \texttt{ -> } \tau}}{\mathtt{x}\colon \sigma, \mathtt{k}\colon \sigma \texttt{ -> } \tau \vdash \mathtt{k}\colon \sigma \texttt{ -> } \tau} \quad \dfrac{}{\mathtt{x}\colon \sigma \vdash \mathtt{x}\colon \sigma}}{\mathtt{x}\colon \sigma, \mathtt{k}\colon \sigma \texttt{ -> } \tau \vdash \mathtt{k}\ \mathtt{x}\colon \tau}}{\mathtt{x}\colon \sigma \vdash \mathtt{fn\ k => k\ x}\colon (\sigma \texttt{ -> } \tau) \texttt{ -> } \tau}}{\vdash \mathtt{fn\ x => fn\ k => k\ x}\colon \sigma \texttt{ -> } (\sigma \texttt{ -> } \tau) \texttt{ -> } \tau}$$

The corresponding tautology is $p \to (p \to q) \to q$ and a proof is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p \to q \vdash p \to q}}{p, p \to q \vdash p \to q} \quad \dfrac{}{p \vdash p}}{p, p \to q \vdash p}}{p, p \to q \vdash q}}{p \vdash (p \to q) \to q}}{\vdash p \to (p \to q) \to q}$$

Curry-Howard

The connection between programs and proofs is called the Curry-Howard correspondence or, less accurately, the Curry-Howard isomorphism.
It extends well beyond what we've seen so far.
So far we don't have inference rules for $\top$ and $\bot$. Let's remedy this, starting with $\top$.
$\top$ should be an axiom, since it represents the value true, which is of course true no matter what Boolean values are assigned any variables.
Also, $\top \to P$ should be true if and only if P is, so we should have rules of inference

$$\frac{P}{\top \to P} \qquad \frac{\top \to P}{P}$$

The counterpart of $\top$ in types is the unit type and the fact that $\top$ is an axiom corresponds to value `()` having type `unit`.
What about the rules of inference? These should correspond to some language constructs, which we can call `force` and `delay`, with typing rules

$$\frac{e : \tau}{\texttt{delay}\ e : \texttt{unit -> } \tau} \qquad \frac{e : \texttt{unit -> } \tau}{\texttt{force}\ e : \tau}$$

Laziness

The Curry-Howard correspondence suggests that we consider language features `force` and `delay` with typing rules

$$\frac{e : \tau}{\texttt{delay}\ e : \texttt{unit -> } \tau} \qquad \frac{e : \texttt{unit -> } \tau}{\texttt{force}\ e : \tau}$$

This takes care of the type, and I've implicitly assumed they behave grammatically like functions, but what should their semantics be? In other words, what rules should govern their evaluation?

`force` takes a τ-valued function of a unit argument, which could only be `()`, and gives you a value of type τ. The obvious way to do this is to evaluate the function at `()`.

`delay` is in some sense the inverse. It takes an expression of type τ and gives you a function of `unit` argument which gives a value of type τ. The obvious way to do this is to use the expression to define an anonymous function.

If implemented as above these aren't new language features. We can consider `delay` e as a shorthand for `fn _ =>` e and to consider `force` e as a shorthand for e `()`.