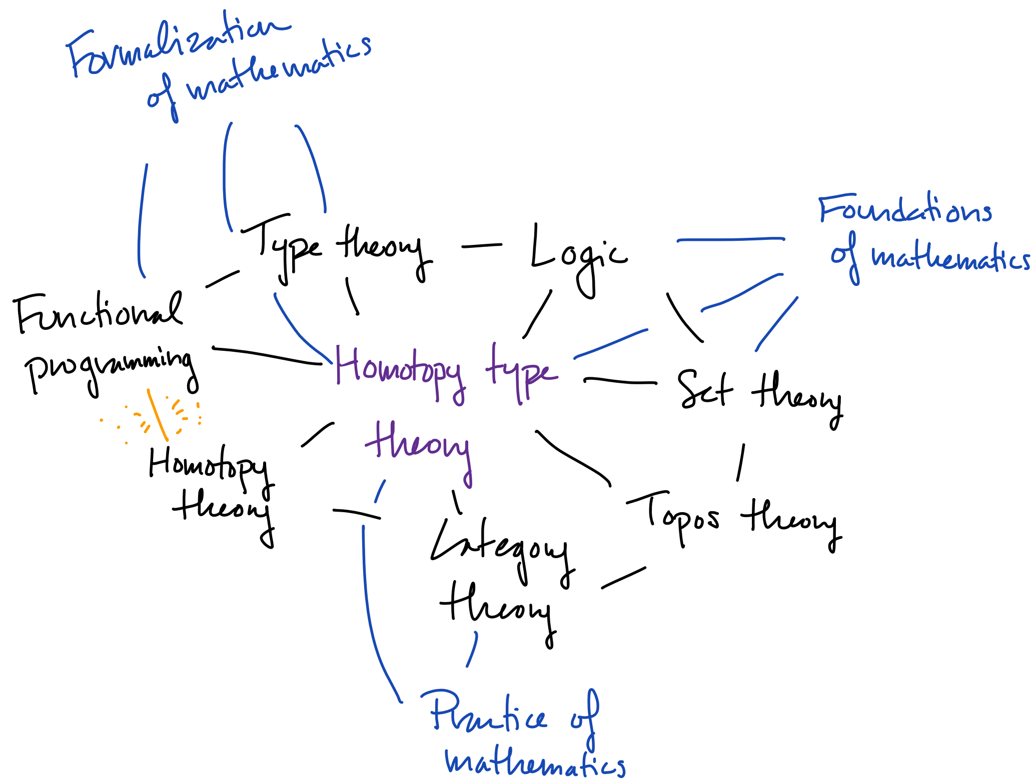

Logic (natural deduction)

We can prove statements about propositions using proof trees built out of rules.

Ex.

$$\dfrac{\dfrac{P \wedge (P \to Q)}{P}\ \wedge\text{-elim-}\ell \qquad \dfrac{P \wedge (P \to Q)}{P \to Q}\ \wedge\text{-elim-r}}{Q}\ \to\text{-elim}$$

Rules for $\wedge$:

$\wedge$-intro: $\dfrac{P \quad Q}{P \wedge Q}$ $\qquad$ $\wedge$-elim-ℓ: $\dfrac{P \wedge Q}{P}$ $\qquad$ $\wedge$-elim-r: $\dfrac{P \wedge Q}{Q}$

Rules for $\to$:

$\to$-intro:
$$\frac{\begin{array}{c}\overline{P}^{\,1}\\ \vdots \\ Q\end{array}}{P \to Q}\,1$$

$\to$-elim
$$\frac{P \quad P \to Q}{Q}$$

<u>Introduction rules</u> tell you how to prove something.
<u>Elimination</u> rules tell you how to use something.

<u>Ex</u>. $(P \wedge Q) \to P$

$$\frac{\dfrac{\overline{P \wedge Q}^{\,1}}{P}\ \wedge\text{-elim-}\ell}{(P \wedge Q) \to P}\ 1\ \to\text{-intro}$$

<u>Simply-typed λ-calculus</u>

If we make natural deduction <u>proof relevant</u>, we get the simply-typed λ-calculus.

In natural deduction, we write "P" to mean "P holds".
Now we write "$p : P$" to mean "p is a proof of P" or
"P holds (by p)" or "P is inhabited (by p)".
We can call p a proof, witness, or a <u>term</u> of P.
We can call P a proposition or a <u>type</u>.

In a derivation, the proofs are manipulated.

<u>Ex</u>. Q follows from $P \wedge (P \to Q)$.

$$\dfrac{\wedge\text{-elim-l}\ \dfrac{a : P \wedge (P \to Q)}{\mathrm{pr}_1\, a : P} \qquad \dfrac{a : P \wedge (P \to Q)}{\mathrm{pr}_2\, a : P \to Q}\ \wedge\text{-elim-r}}{(\mathrm{pr}_1\, a)(\mathrm{pr}_2\, a) : Q}\ \to\text{-elim}$$

Notice that the resulting term $(\mathrm{pr}_1\, a)(\mathrm{pr}_2\, a) : Q$ actually records the proof tree, in the sense that the proof tree can be reconstructed if you only know the resulting term.

<u>Rules for $\wedge$</u>

$\wedge$-form: $\dfrac{P \text{ TYPE} \quad Q \text{ TYPE}}{P \wedge Q \text{ TYPE}}$

$\wedge$-intro: $\dfrac{\Gamma \vdash p : P \quad \Gamma \vdash q : Q}{\Gamma \vdash (p, q) : P \wedge Q}$

$$\wedge\text{-elim-}l: \frac{\Gamma \vdash a : P \wedge Q}{\Gamma \vdash pr_1\, a : P} \qquad \wedge\text{-elim-}r: \frac{\Gamma \vdash a : P \wedge Q}{\Gamma \vdash pr_2\, a : Q}$$

$$\wedge\text{-comp-}\beta\text{-}l: \frac{\Gamma \vdash p : P \quad q : Q}{\Gamma \vdash pr_1(p,q) \doteq p : P} \qquad \wedge\text{-comp-}\beta\text{-}r: \frac{\Gamma \vdash p : P \quad q : Q}{\Gamma \vdash pr_2(p,q) \doteq q : Q}$$

$$\wedge\text{-comp-}\eta: \frac{\Gamma \vdash a : P \wedge Q}{\Gamma \vdash (pr_1 a, pr_2 a) \doteq a : P \wedge Q}$$

Notice that if we think of types as sets and terms as elements, then $P \wedge Q$ behaves like the product.

<u>Thm.</u> (Lambek 1985)
There is an interpretation of the STLC with $\wedge$ and $\to$ into Set, the category of sets. (Actually there is a equivalence with CCC.)

<u>Rules for $\to$:</u>

$$\to\text{-form}: \frac{P \text{ TYPE} \quad Q \text{ TYPE}}{P \wedge Q \text{ TYPE}}$$

$$\to\text{-intro} \quad \frac{\text{a proof of } Q \text{ from } P}{P \to Q} \rightsquigarrow \frac{\Gamma, x : P \vdash q : Q}{\Gamma \vdash \lambda x. q : P \to Q}$$

Now we need contexts, and we want every rule to hold in every context.

Ex. How do we prove $P \wedge Q \to P$?

projection: $\Gamma, x:T, \Delta \vdash x:T$

$$\dfrac{\dfrac{\dfrac{}{a : P\wedge Q \vdash a : P \wedge Q}}{a : P \wedge Q \vdash pr_1\, a : P}\ \wedge\text{-elim-}\ell}{\vdash \lambda a.\, pr_1\, a : (P\wedge Q) \to P}\ \to\text{-intro}$$

$\rightsquigarrow P\wedge Q$

$\rightsquigarrow \dfrac{P \wedge Q}{P}$

$\rightsquigarrow \dfrac{\dfrac{\dfrac{}{P\wedge Q}}{P}}{(P\wedge Q)\to P}$

So an expression like $a : P\wedge Q \vdash pr_1\, a : P$ records the hypotheses of the proof tree in the context and the structure in the term.

<u>Theorem</u> (Howard 1969) (Often called the Curry-Howard correspondence)
The proof trees of natural deduction are in 1-to-1 correspondence with terms of the STLC.

<u>Rules for $\to$ continued.</u>

$\to$-form: $\dfrac{P \text{ TYPE} \quad Q \text{ TYPE}}{P \to Q \text{ TYPE}}$

$\to$-intro $\dfrac{\Gamma, x:P \vdash q : Q}{\Gamma \vdash \lambda x.\, q : P \to Q}$

$\to$-elim
$$\frac{\Gamma \vdash f : P \to Q \qquad \Gamma \vdash p : P}{\Gamma \vdash fp : Q}$$

$\to$-comp-β
$$\frac{\Gamma, x : P \vdash q : Q \qquad \Gamma \vdash p : P}{\Gamma \vdash (\lambda x . q)(p) \doteq q[p/x]}$$
substitution

$\to$-comp-η
$$\frac{\Gamma \vdash f : P \to Q}{\Gamma \vdash \lambda x . (fx) \doteq f : P \to Q}$$

Under the Howard correspondence, $\to$ corresponds to implication
and under the Lambek interpretation, $\to$ corresponds to functions
(or internal homs).

We can also regard types a program specification
- ex: A type $P \to P$ specifies a program that takes input of type P and produces an output of type P.

and terms as programs meeting that specification.
- ex: We can construct the identity $id_P : P \to P$.

This also falls under the name "Curry-Howard correspondence"
and makes formalization in Agda possible.

Dependent type theory

- In natural deduction, we have no terms.

- In the simply typed lambda calculus, terms can depend on other terms.

  e.g. $a : P \wedge Q \vdash pr(a) : P$

- In dependent type theory, types can depend on terms.

  e.g. $n : \mathbb{N} \vdash \mathrm{Vect}_n \ \text{TYPE}$

  $n : \mathbb{N} \vdash \mathrm{isEven}(n) \ \text{TYPE}$

If we interpret types as:

- propositions, dependent types are predicates.
- sets, dependent types are families of sets.
- programs, dependent types are program specifications with a parameter.

We have the same rules as before, except the formation rules can also have a context.

$\wedge$-form: $\dfrac{\Gamma \vdash P \ \text{TYPE} \qquad \Gamma \vdash Q \ \text{TYPE}}{\Gamma \vdash P \wedge Q \ \text{TYPE}}$ $\qquad$ $\to$-form: $\dfrac{\Gamma \vdash P \ \text{TYPE} \qquad \Gamma \vdash Q \ \text{TYPE}}{\Gamma \vdash P \to Q \ \text{TYPE}}$

Ex. $\dfrac{n : \mathbb{N} \vdash \mathrm{isEven}(n) \qquad n : \mathbb{N} \vdash \mathrm{isDivThree}(n)}{n : \mathbb{N} \vdash \mathrm{isEven}(n) \wedge \mathrm{isDivThree}(n)}$

$n : \mathbb{N} \vdash \mathrm{isEven}(n) \to \mathrm{isDivThree}(n)$

Dependent function types

Ex (informal):

Consider the set Vect of all vectors (in $\mathbb{N}^n$) of any length (n)

(i.e., finite lists of natural numbers).

We could define a function $0 : \mathbb{N} \to \text{Vect}$ where $0(n)$ is the vector of length n whose components are all 0.

But $0(n)$ actually lives in Vect_n.

We can encode this by considering 0 as a dependent function

$$0 : \prod_{n:\mathbb{N}} \text{Vect}_n \quad (\text{sometimes write } 0 : (n:\mathbb{N}) \to \text{Vect}_n)$$

The elimination rule (function application) gives us $0(n) : \text{Vect}_n$ for any $n:\mathbb{N}$.

Ex. Suppose we have predicate

$$n:\mathbb{N} \vdash \text{isEven}(n) \vee \text{isOdd}(n)$$

and we want to show this is true for all n.

We need a term

$$n:\mathbb{N} \vdash t(n) : \text{isEven}(n) \vee \text{isOdd}(n).$$

The introduction rule (λ-abstraction) gives us a dependent function

$$\vdash \lambda n. t(n) : \prod_{n:\mathbb{N}} \text{isEven}(n) \vee \text{isOdd}(n).$$

In the logical interpretation, we interpret $\to$ as implication and $\prod$ as $\forall$.

Rem: $\to$ is a special case of Π.

ex. $\prod_{n:\mathbb{N}} \mathrm{Vect}$ is the same as $\mathbb{N} \to \mathrm{Vect}$ (the rules become the same.)

We usually only postulate Π-types.

Rules for Π-types.

Π-form:
$$\frac{\Gamma, x:A \vdash B \quad \mathrm{TYPE}}{\Gamma \vdash \prod_{x:A} B}$$

Π-intro:
$$\frac{\Gamma, x:A \vdash b : B}{\Gamma \vdash \lambda x.b : \prod_{x:A} B}$$

Π-elim:
$$\frac{\Gamma \vdash f : \prod_{x:A} B(x)}{\Gamma \vdash fa : B[a/x]}$$

Π-comp-β:
$$\frac{\Gamma, x:A \vdash b : B \qquad \Gamma \vdash a : A}{\Gamma \vdash (\lambda x.b)(a) \doteq b[a/x]}$$

Π-comp-η:
$$\frac{\Gamma \vdash f : \prod_{x:A} B}{\Gamma \vdash \lambda x.(fx) \doteq f : \prod_{x:A} B}$$