Week 3 Lecture Notes

Enoch Cheung and Clive Newstead

9/30/2013 and 10/2/2013

1 The β and η Rules

1.1 Gentzen's Inversion Principles (β rules)

Recall the β rules:

$$\begin{aligned}
&\wedge_1 : \mathsf{fst}\langle M, N\rangle \equiv M \\
&\wedge_2 : \mathsf{snd}\langle M, N\rangle \equiv N \\
&\supset_1 : (\lambda x.M)(N) \equiv [N/x]M \\
&\vee_1 : \mathsf{case}(\mathsf{inl}(M); x.P, y.Q) \equiv [M/x]P \\
&\vee_2 : \mathsf{case}(\mathsf{inr}(M); x.P, y.Q) \equiv [M/y]Q
\end{aligned}$$

The β rules can be expressed very compactly, and tells us that elimination rules should "cancel out" introduction rules. The notation used here is $\mathsf{inl}(x)$ (inject left) to mean using the proof x to prove A to prove $A \vee B$, and $\mathsf{inr}(x)$ (inject right) to mean using the proof x to prove B to prove $A \vee B$. $\mathsf{case}(x, y, z)$ means "if x, then y, else z." This also expresses the idea of dynamics of proofs, meaning that proofs can be viewed as programs.

1.2 Gentzen's Unicity Principles (η rules)

Recall the η rules we have given so far:

Truth

$$\frac{\Gamma \vdash M : \top}{\Gamma \vdash M \equiv \langle\,\rangle : \top}\ \eta\top$$

Conjunction

$$\frac{\Gamma \vdash M : A \wedge B}{\Gamma \vdash M \equiv \langle \mathsf{fst}(M), \mathsf{snd}(N) \rangle : A \wedge B}\ \eta\wedge$$

Implication

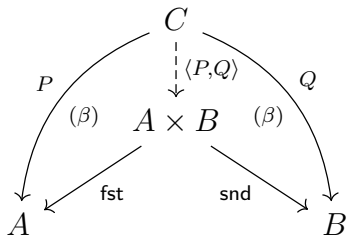
$$\frac{M : A \supset B}{\Gamma \vdash M \equiv \lambda x.Mx : A \supset B}\ \eta\supset$$

The η rules on the other hand takes a little bit more to write out, and expresses uniqueness (up to equivalence) of proofs of certain types.

The η conjunction rule can be expressed another way:

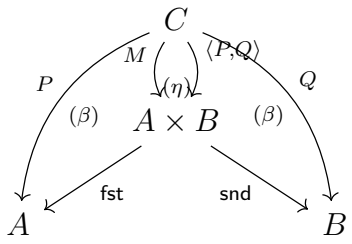
$$\frac{\Gamma \vdash M : A \wedge B \quad \Gamma \vdash \mathsf{fst}(M) \equiv P : A \quad \Gamma \vdash \mathsf{snd}(M) \equiv Q : B}{\Gamma \vdash \langle P, Q \rangle \equiv M : A \wedge B}\ \eta\wedge$$

This time we give equivalent proof terms P, Q to $\mathsf{fst}(M)$ and $\mathsf{snd}(N)$. This corresponds to the product diagram (where $A \wedge B$ corresponds to the product $A \times B$):



which says that given any object C with maps $P : C \to A$ and $Q : C \to B$, there exists a unique map $\langle P, Q \rangle : C \to A \times B$ such that the β rules make each cell commute, meaning $P \equiv \mathsf{fst}(\langle P, Q \rangle)$ and $Q \equiv \mathsf{snd}(\langle P, Q \rangle)$.

The η rule, on the other hand, gives uniqueness of the $\langle P, Q \rangle$ map, expressed as



where the η rule maks the center cell commute, meaning that given any map $M : C \to A \times B$ such that $\mathsf{fst} \circ M \equiv P$ and $\mathsf{snd} \circ M \equiv Q$, we have $M \equiv \langle P, Q \rangle$, expressing the uniqueness of the $\langle P, Q \rangle$ map.

While η rules gives the uniqueness of the product map, one can ask whether the product object $A \times B$ is unique. In a Heyting algebra, we can show that if $A \wedge' B$ has the same properties (being a greatest lower bound of A and B) as $A \wedge B$, then:

$$A \wedge B \leq A \wedge' B \qquad A \wedge B \geq A \wedge' B$$

If we think of them as objects, then we have maps $F : A \times B \to A \times' B$ and $G : A \times B \to A \times' B$ such that $F \circ G = \mathsf{id}$ and $G \circ F = \mathsf{id}$, which gives $A \times B \cong A \times' B$. With the Univalence axiom, we identify equivalence things as being equal, so we can say that $A \times B = A \times' B$.

1.3 η rule for Disjunction $\vee$

We wish to give the η rule for $\vee$, but if we were to attempt to naively define it as we did before for $M : A \vee B$, then we might force M to be a proof of A or B, because a proof of A also proves $A \vee B$, but forcing proofs of A or a proof of B to be identified with a proof of $A \vee B$ would not make sense.

We will take inspiration from Shannon expansions, specifically the concept of case analysing to give two different proofs. As a toy example, we consider a proof of $\top \vee \top$

$$\begin{aligned} \mathsf{inl}(\langle\,\rangle) &= \mathsf{true} \\ \mathsf{inr}(\langle\,\rangle) &= \mathsf{false} \\ \langle\,\rangle &: \top \vee \top \\ \mathsf{case}(M; P, Q) &= \text{“if } M \text{ then } P \text{ else } Q\text{”} : \top \vee \top \end{aligned}$$

where we look at the variable M and decide to use P or Q (here P, Q does not have an input because there is no data for proof of $\top$ anyways). The is an example of a Binary Decision Diagram (BDD), because we are making a decision when examining the variable M and branching to two cases.

In general, we can imagine a much bigger BDD which has many variables examined sequentially, and look at the Shannon expansion at some variable M in the middle. The idea here is that $M = \mathsf{true}$ and $M = \mathsf{false}$ will lead to two different subtrees. We write

$$[M/z]P \equiv \text{if } M \text{ then } [\mathsf{true}/z]P \text{ else } [\mathsf{false}/z]P$$

where when we look at the variable M which P uses, there are two cases: the case where M is true, which gives $[\mathsf{true}/z]P$ and the case where M is false which gives

$[\mathsf{false}/z]P$. The point is that $[\mathsf{true}/z]P$ and $[\mathsf{false}/z]P$ do not have M as a variable, so M is fixed at a value.

Another notation for this is

$$P_z \equiv \text{if } z \text{ then } [\mathsf{true}/z]P \text{ else } [\mathsf{false}/z]P$$

To write the η rule for $\vee$, we will describe what happens when a proof $P : C$ uses a proof term $z : A \vee B$, meaning that the C follows from $A \vee B$. Now suppose we have a proof $M : A \vee B$, and we want to look at what happens when we make $P : C$ include $M : A \vee B$ by doing the substitution $[M/z]P : C$.

$$\frac{\Gamma \vdash M : A \vee B \quad \Gamma, z : A \vee B \vdash P : C}{\Gamma \vdash [M/z]P \equiv \mathsf{case}(M; x.[\mathsf{inl}(x)/z]P, y.[\mathsf{inr}(y)/z]P) : C}\ \eta\vee$$

This can be thought of as a "generalized Shannon expansion," where the Shannon expansion can be recovered as a special case

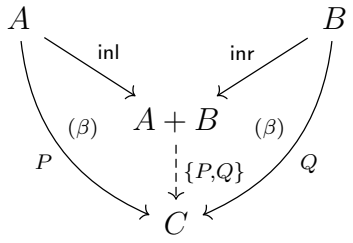
$$M \equiv \mathsf{case}(M; x.\mathsf{inl}(x); y.\mathsf{inr}(y))$$

1.4 Coproduct

In the category theoretical view, the disjunction $A \vee B$ corresponds to the coproduct $A + B$, with $\mathsf{inl} : A \to A + B$ and $\mathsf{inr} : B \to A + B$ being the canonical injections.

To give some intuitions about the coproduct, if we were in the category of sets, we can think of $A + B = A \sqcup B = (\{0\} \times A) \cup (\{1\} \times B)$ as a disjoint union, then $\mathsf{inl}, \mathsf{inr}$ would be the canonical embeddings $\mathsf{inl} : a \mapsto (0, a)$ and $\mathsf{inr} : b \mapsto (1, b)$. If A, B are already disjoint, then we can let $A + B = A \cup B$ and $\mathsf{inl}, \mathsf{inr}$ would be the inclusion maps $\mathsf{inl} : a \mapsto a$ and $\mathsf{inr} : b \mapsto b$.

The β rule for $\vee$ gives the following commutative diagram:



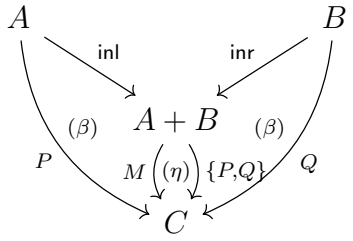
Where given any object C with maps $P : A \to C$ and $Q : B \to C$, there exists a unique map $\{P, Q\} : A + B \to C$ that is the copair of maps P, Q, which in our context corresponds to

$$\{P, Q\} \approx \mathsf{case}(-; x.P, y.Q)$$

The β rule makes the diagram commute, meaning that the composition of maps $P \equiv \{P, Q\} \circ \mathsf{inl}$ and $Q \equiv \{P, Q\} \circ \mathsf{inr}$. Written another way:

$$\mathsf{case}(\mathsf{inl}(-); x.P, y.Q) \equiv [-/x]P$$
$$\mathsf{case}(\mathsf{inr}(-); x.P, y.Q) \equiv [-/x]Q$$

The η rule expresses uniqueness, which is demonstrated by the following diagram



where given a map $M : A + B \to C$ such that $M \circ \mathsf{inl} \equiv P$ and $M \circ \mathsf{inr} \equiv Q$, the map is in fact equivalent to $M \equiv \{P, Q\}$, so the η rule makes the center cell commute. In other words,

$$\frac{M : A + B \to C \quad M \circ \mathsf{inl} = P : A \to C \quad M \circ \mathsf{inr} = Q : B \to C}{M = \{P, Q\} : A + B \to C}\ \eta{+}$$

Just as we have done for $\eta\wedge$, we can rewrite the $\eta\vee$ rule by explicitly naming $P : Q \to C$ and $Q : B \to C$ as follows

$$\frac{\begin{array}{c}\Gamma, z : A + B \vdash M : C \\ \Gamma, x : A \vdash [\mathsf{inl}(x)/z]M \equiv P : C \\ \Gamma, y : B \vdash [\mathsf{inr}(y)/z]M \equiv Q : C\end{array}}{\Gamma, z : A + B \vdash M \equiv \mathsf{case}(z; x.P, y.Q) : C}\ \eta{\vee}$$

1.5 Definitional equality vs. Propositional equality

Our different treatments of β rules and η rules above suggests that there is something fundamentally different between equivalence given by β rules and equivalence given by η rules. Indeed, there is a distinction which we will make more clear later. For now, note that

β rules	Analytical ("self-evident")	Definitional equality
η rules	Synthetic ("require proof")	Propositional equality

The β rules can be thought of as self-evident, or analytical, because it just says that our notation such as $\mathsf{fst}, \mathsf{snd}, \langle -, - \rangle, \mathsf{inl}, \mathsf{inr}, \mathsf{case}$ should behave the way we expect them to. On the other hand, the η rules are not so obvious, and expresses the equivalence of two things that behaves the same way, so they are synthetic, or requires proof instead of being self-evident.

The notion of equality produced by β rules is called definitional equality, or judgemental equality, which is more basic. The notion of equality produced by η rules is called propositional equality, which has to be expressed by a type (so it is typical).

2 Natural numbers

We'd like to capture the idea of definition by recursion. We will do so in two ways. First we will implement the natural numbers syntactically as a type, denoted Nat—it is a ubiquitous example of an inductively defined type. Then we will implement the natural numbers in a category theoretic context, as a so-called natural numbers object (NNO), denoted $\mathbb{N}$.

2.1 Syntactic definition: Nat

The type Nat has two introduction rules:

$$\frac{}{\Gamma \vdash 0 : \mathsf{Nat}}\ \mathsf{Nat}\text{-}I_0, \qquad \frac{\Gamma \vdash M : \mathsf{Nat}}{\Gamma \vdash s(M) : \mathsf{Nat}}\ \mathsf{Nat}\text{-}I_s$$

and one elimination rule, which can be thought of as a `for` loop or a recursion:

$$\frac{\Gamma \vdash M : \mathsf{Nat} \quad \Gamma \vdash P : A \quad \Gamma, x : A \vdash Q : A}{\Gamma \vdash \mathsf{rec}(P, x.Q)(M) : A}\ \mathsf{Nat}\text{-}E$$

We call rec the *recursor*.

We can think of 0 as being zero and s as being the successor operation, which takes a natural number n to its successor $n + 1$.

The β-rules for Nat are what they 'should be':

$$\frac{\Gamma \vdash P : A \quad \Gamma, x : A \vdash Q : A}{\Gamma \vdash \mathsf{rec}(P, Q)(0) \equiv P : A}\ \beta\text{-}\mathsf{Nat}_0$$

$$\frac{\Gamma \vdash M : \mathsf{Nat} \quad \Gamma \vdash P : A \quad \Gamma, x : A \vdash Q : A}{\Gamma \vdash \mathsf{rec}(P, Q)(s(M)) \equiv [\mathsf{rec}(P, Q)(M)/x]Q : A}\ \beta\text{-}\mathsf{Nat}_s$$

The η-rule for the NNO is somewhat ugly:

$$\frac{\Gamma, z : \mathsf{Nat} \vdash M : A \quad \Gamma, z : \mathsf{Nat} \vdash [s(z)/z]M \equiv [M/x]Q \quad \Gamma \vdash [0/z]M \equiv P : A}{\Gamma, z : \mathsf{Nat} \vdash M \equiv \mathsf{rec}(P, Q)(z) : A}\ \eta\text{-}\mathsf{Nat}$$

It says that 'if something behaves like the recursor, then it is the recursor'.

Given $n \in \mathbb{N}$, define the numeral $\bar{n} = \underbrace{s(s(\cdots s}_{n \text{ times}}(0)\cdots))$. With a slight abuse of notation, the β then tells us that

$$\mathsf{rec}(P, Q)(\bar{n}) \equiv \underbrace{Q(Q(\cdots Q}_{n \text{ times}}(P)\cdots))$$

That is, $\mathsf{rec}(P, Q)(0) \equiv P$ and $\mathsf{rec}(P, Q)(\overline{n+1}) \equiv Q(\mathsf{rec}(P, Q)(\bar{n}))$. This is precisely a definition by recursion.

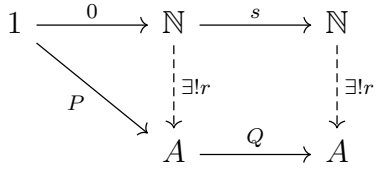
A special case of this is when $P = 0$ and Q is the successor operation. Then

$$z : \mathsf{Nat} \vdash \mathsf{rec}(0, s.s(y))(z) \equiv z : \mathsf{Nat}$$

This is what we'd expect: if you apply the successor operation to 0 n times then what you obtain is n.

2.2 Category theoretic definition: NNO

Fix a category $\mathcal{C}$ and suppose that $\mathcal{C}$ has a terminal object 1. A natural numbers object in $\mathcal{C}$ is an object $\mathbb{N}$ equipped with arrows $0 : 1 \to \mathbb{N}$ and $s : \mathbb{N} \to \mathbb{N}$ satisfying the following univeral property:



That is, given any morphism $P : 1 \to A$ and $Q : A \to A$ there exists a unique morphism $r = \mathsf{rec}(P, Q) : \mathbb{N} \to A$ such that

$$\mathsf{rec}(P, Q) \circ 0 = P \quad \text{and} \quad \mathsf{rec}(P, Q) \circ s = Q \circ \mathsf{rec}(P, Q)$$

These two equations correspond precisely with the β rules for Nat.

The η rule corresponds with the uniqueness: if $M : \mathbb{N} \to A$ satisfies $M \circ s = Q \circ M$ and $M \circ 0 = P$ then $M = \mathsf{rec}(P, Q)$.

Concrete example

In the category of sets, take $\mathbb{N}$ to be the set of natural numbers. The terminal object is any singleton $\{*\}$, and we can define $0 : \{*\} \to \mathbb{N}$ by $0(*) = 0 \in \mathbb{N}$ and $s : \mathbb{N} \to \mathbb{N}$ by $s(n) = n + 1$. Then the triple $(\mathbb{N}, 0, s)$ defines a natural numbers object: if $P \in A$ and $Q : A \to A$ then we can define $\mathsf{rec}(P, Q) : \mathbb{N} \to A$ by

$$\mathsf{rec}(P, Q)(0) = P \quad \text{and} \quad \mathsf{rec}(P, Q)(n + 1) = Q(\mathsf{rec}(P, Q)(n))$$

It is then clear that the above diagram commutes, and we can prove that $\mathsf{rec}(P, Q)$ is the unique such function by induction on its argument.

NNO as an initial algebra

There is an equivalent definition of a natural numbers object as an *initial algebra*.

Given an endofunctor (i.e. a functor F from a category $\mathcal{C}$ to itself), an *F-algebra* is a pair (A, α), where A is an object in the category and $\alpha : F(A) \to A$ is a morphism.

A *homomorphism of F-algebras* $f : (A, \alpha) \to (B, \beta)$ is a map $f : A \to B$ making the following square commute:

$$\begin{array}{ccc} F(A) & \xrightarrow{F(f)} & F(B) \\ \downarrow{\scriptstyle \alpha} & & \downarrow{\scriptstyle \beta} \\ A & \xrightarrow{f} & B \end{array}$$

That is, f respects α and β in the only way it can.

An *initial F-algebra* is an F-algebra (I, ι) such that given any other F-algebra (A, α) there exists a unique F-algebra homomorphism $(I, \iota) \to (A, \alpha)$.

With these definitions in mind, a natural numbers object is precisely an initial F-algebra, where F is the functor $1 + (-)$.

To see how this functor acts on morphisms, consider the more general scenario of having morphisms $f : A \to A'$ and $g : B \to B'$. Then we have morphisms

$$\mathsf{inl} \circ f : A \to A' + B' \quad \text{and} \quad \mathsf{inr} \circ g : B \to A' + B'$$

Then the universal property of the coproduct gives rise to a map

$$f + g = \{\mathsf{inl} \circ f, \mathsf{inr} \circ g\} : A + B \to A' + B'$$

What this means more concretely is as follows. A natural numbers object is an object $\mathbb{N}$ equipped with a morphism $\{0, s\} : 1 + \mathbb{N} \to \mathbb{N}$ such that if $\{P, Q\} : 1 + A \to A$ is another morphism then there is a unique morphism $\mathsf{rec}(P, Q) : \mathbb{N} \to A$ such that $\{P, Q\} \circ (1 + \mathsf{rec}(P, Q)) = \mathsf{rec}(P, Q) \circ \{0, s\}$.

3 Intensional and extensional equality

We can implement addition by

$$p = \lambda x \lambda y \, \mathsf{rec}(x, z.s(z))(y)$$

Given numerals $\bar{m}$ and $\bar{n}$ it is clear that $p\,\bar{m}\,\bar{n} = \overline{m+n}$, so this definition does implement $+$.

We could have recursed on x instead of y. Indeed, we can define $q = \lambda x \lambda y \, pyx$.

Again we can prove that $q\,\bar{m}\,\bar{n} = \overline{m+n}$, so q is another implementation of addition.

Despite this fact, we will not in general be able to prove

$$x : \mathsf{Nat}, y : \mathsf{Nat} \vdash pxy \equiv qxy$$

This seems odd: for every $m, n \in \mathbb{N}$ (in the 'real world') we can prove that $p\,\bar{m}\,\bar{n} = q\,\bar{m}\,\bar{n}$. If we had a principle of induction then we'd be able to deduce that $pxy = qxy$ generically. However, we have no such principle!

Morally this should not be the case: that is, p and q are not *definitionally equal*. This illustrates the distinction between *intensional equality* (a.k.a. *definitional equality*) and *extensional equality*. This distinction is very important in computer science and philosophy: it captures the idea of two programmes having the same input–output behaviour but different algorithms.

Extensional equality. We can think of the *extension* of a function as being its graph, i.e. a set of ordered pairs of the form $(\mathsf{input}, \mathsf{output})$. Two programmes may have the same input/output behaviour without being the same programme. In Frege's terminology, two types are extensionally equal if they have the same *reference*.

We cannot expect extensional equality to be computable; for instance, extensional equality of elements of type $(\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ already has high quantifier complexity.

Intensional equality. We can think of the *intension* of a function as being its description, or an algorithm that computes the function. Thus two functions that are intensionally equal must be extensionally equal, but the converse is not true. Intensional equality is synthetic. In Frege's terminology, two types are intensionally equal if they have the same *sense*.

3.1 Equality in type theory

Recall Martin–Löf's distinction between judgements and propositions. With this in mind:

- Intensional equality is an inductively defined judgement;
- Extensional equality is a proposition: it may be subject to judgement.

For example, the following is a proposition:

$$pxy =_{\mathsf{Nat}} qxy$$

It requires proof. We will attempt to develop a way of saying that, to prove this, it is sufficient to prove for each $m, n \in \mathbb{N}$ that $p\,\bar{m}\,\bar{n} = q\,\bar{m}\,\bar{n}$.

Under our propositions-as-types correspondence, we conclude that extensional equality 'is' a family of types. For instance,

$$x : \mathsf{Nat}, y : \mathsf{Nat} \vdash x =_{\mathsf{Nat}} y\ \mathsf{type} \tag{1}$$

We'll write the type $x =_{\mathsf{Nat}} y$ as $\mathsf{Id}_{\mathsf{Nat}}(x, y)$ to emphasise that we really want to think of it as a type and not a proposition.

Instantiating by substitution from (1) gives

$$\frac{\Gamma \vdash M : \mathsf{Nat} \quad \Gamma \vdash N : \mathsf{Nat}}{\Gamma \vdash \mathsf{Id}_{\mathsf{Nat}}(M, N)\ \mathsf{type}}$$

But we needn't stop at Nat; we may replace it by an arbitrary type A (which may itself—usefully—be an identity type!). For instance, given $x : \mathsf{Nat}$ we may obtain a new type $\mathsf{Seq}(x)$, which can be thought of as the sequences of Nats of length x:

$$\frac{\Gamma \vdash x : \mathsf{Nat}}{\Gamma \vdash \mathsf{Seq}(x) :\ \mathsf{type}}$$

Observe the following fact: given $m, n \in \mathbb{N}$, it is true that

$$\mathsf{Seq}(p\,\bar{m}\,\bar{n}) \equiv \mathsf{Seq}(q\,\bar{m}\,\bar{n})$$

because $p\,\bar{m}\,\bar{n} \equiv q\,\bar{m}\,\bar{n}$. However we cannot generalise to

$$\mathsf{Seq}(pxy) \equiv \mathsf{Seq}(qyx)$$

because $\mathsf{Seq}(pxy)$ and $\mathsf{Seq}(qyx)$ are not definitionally equivalent. But they *are* related in some way. Later, we will come to define what we mean by 'related' here. A good guess might be along the lines of 'isomorphism', but this will turn out to be far too strong. What we need is some kind of 'equivalence'. This equivalence will tie itself to both the notion of a homotopy and that of a categorical equivalence.

4 Dependent types: setup

Dependent types are *families* of types. Atomic judgements are of the form

contexts / closed types:	$\Gamma\ \mathsf{ctx}$
	$\Gamma \equiv \Gamma'$
open types / families of types:	$\Gamma \vdash A\ \mathsf{type}$
	$\Gamma \vdash A \equiv A'$
elements of types:	$\Gamma \vdash M : A$
	$\Gamma \vdash M \equiv M' : A$

The symbol $\equiv$ denotes what we will interpret as *definitional equality*. We denote the *empty context* by $\cdot$ when we need to. The introduction rules for contexts are:

$$\frac{}{\cdot\ \mathsf{ctx}} \qquad \frac{\Gamma\ \mathsf{ctx} \quad \Gamma \vdash A\ \mathsf{type}}{\Gamma, x : A\ \mathsf{ctx}}$$

Thus we have some notion of *dependence*; it allows us to make sense of expressions like $x : \mathsf{Nat}, y : \mathsf{Seq}(x) \vdash \cdots$, which was impossible before.

$$\frac{}{\cdot \equiv \cdot\ \mathsf{ctx}} \qquad \frac{\Gamma \equiv \Gamma' \quad \Gamma \vdash A \equiv A'}{\Gamma, x : A \equiv \Gamma', x : A'}$$

The following rule corresponds with reflexivity:

$$\frac{}{\Gamma, x : A, \Delta \vdash x : A}$$

The following rules (one for each judgement J) correspond with weakening:

$$\frac{\Gamma, \Delta \vdash J \quad \Gamma \vdash A\ \mathsf{type}}{\Gamma, x : A, \Delta \vdash J}$$

Exercise. What are the corresponding rules for exchange and contraction?

The following rule, called *substitution* or *instantiation*, corresponds with transitivity:

$$\frac{\Gamma, x : A, \Delta \vdash J \quad \Gamma \vdash M : A}{\Gamma[M/x]\Delta \vdash [M/x]J}$$

The following rules together are called *functionality*

$$\frac{\Gamma, x : A, \Delta \vdash N : B \quad \Gamma \vdash M \equiv M' : A}{\Gamma[M/x]\Delta \vdash [M/x]N \equiv [M'/x]N : [M/x]B}$$

$$\frac{\Gamma, x : A, \Delta \vdash B\ \mathsf{type} \quad \Gamma \vdash M \equiv M' : A}{\Gamma[M/x]\Delta \vdash [M/x]B \equiv [M'/x]B}$$

Finally, the following rules are *type equality*, which tell us that definitionally equal types classify the same things:

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A \equiv A'}{\Gamma \vdash M : A'} \qquad \frac{\Gamma \vdash M \equiv M' : A \quad \Gamma \vdash A \equiv A'}{\Gamma \vdash M \equiv M' : A'}$$

Identity types

Given a type A and elements $M : A$ and $N : A$ we can form an *identity type* $\mathsf{Id}_A(M, N)$. The formation rule for Id is thus:

$$\frac{\Gamma \vdash A\ \mathsf{type} \quad \Gamma \vdash M : A \quad \Gamma \vdash N : A}{\Gamma \vdash \mathsf{Id}_A(M, N)}\ \mathsf{Id}\text{-}F$$

It will be useful in HoTT to consider the case when A is itself an identity type, i.e. we have the type

$$\mathsf{Id}_{\mathsf{Id}_A(A,B)}(\alpha, \beta)$$

This extends to any (finite) dimension.

We also have an Id-introduction rule, which tells us that any element M of a type A is in some way 'related' to itself. Formally:

$$\frac{\Gamma \vdash A : M}{\Gamma \vdash \mathsf{refl}_A(M) : \mathsf{Id}_A(M, M)}\ \mathsf{Id}\text{-}I$$

Id-elimination will follow next week.