Type Theory

Lecture 4: Calculus of Constructions and Homotopy Type Theory

H. Geuvers

Radboud University
Nijmegen, NL

OPLSS 2026 Summer 2026

Outline

The Calculus of Constructions

Homotopy type theory

The Barendregt cube

Barendregt cube: 8 typed λ-calculi, defined in one coherent way.
Generalization: Berardi & Terlouw: Pure Type Systems

framework for defining and studying typed λ-calculi
PTS = pure type system

the PTS rules are basically the λP rules as presented before.

variations on the product rule

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A.\, B : s_2}$$

λP	$s_1 = *,\ s_2 \in \{*, \Box\}$
	$(s_1, s_2) \in \{(*,*), (*,\Box)\}$
$\lambda\!\rightarrow$	$(s_1, s_2) \in \{(*,*)\}$
$\lambda 2$	$(s_1, s_2) \in \{(*,*), (\Box,*)\}$
λC	$(s_1, s_2) \in \{(*,*), (*,\Box), (\Box,*), (\Box,\Box)\}$

(axiom) $\vdash * : \Box$

(var) $$\frac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A}$$ (weak) $$\frac{\Gamma \vdash A : s \quad \Gamma \vdash M : C}{\Gamma, x{:}A \vdash M : C}$$

(Π) $$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A.B : s_2} \text{ if } (s_1, s_2) \in \mathcal{R}$$

(λ) $$\frac{\Gamma, x{:}A \vdash M : B \quad \Gamma \vdash \Pi x{:}A.B : s}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B}$$

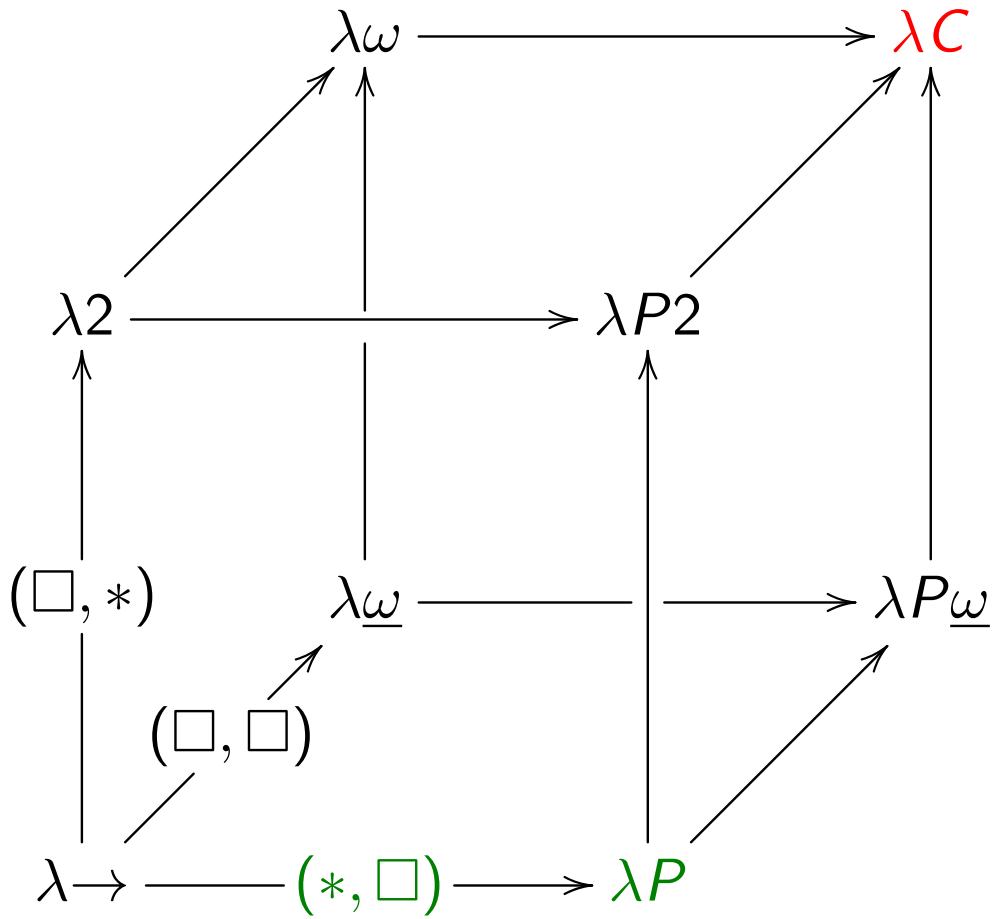
(app) $$\frac{\Gamma \vdash M : \Pi x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B[N/x]}$$

(conv) $$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s}{\Gamma \vdash M : B} \text{ if } A =_\beta B$$

$$(\Pi)\quad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A.B : s_2} \quad \text{if } (s_1, s_2) \in \mathcal{R}$$

System	$\mathcal{R}$			
$\lambda\!\!\rightarrow$	$(*,*)$			
$\lambda 2$ (system F)	$(*,*)$	$(\Box,*)$		
λP (LF)	$(*,*)$		$(*,\Box)$	
$\lambda\overline{\omega}$	$(*,*)$			$(\Box,\Box)$
λP2	$(*,*)$	$(\Box,*)$	$(*,\Box)$	
$\lambda\omega$ (system Fω)	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$
$\lambda P\overline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$
λPω (CC)	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$

the Barendregt cube



Calculus of Constructions

$\lambda\!\rightarrow$ in this presentation is equivalent to $\lambda\!\rightarrow$ as presented before.
Similarly for $\lambda 2$, λP, ... This cube also gives a fine structure for the *Calculus of Constructions*, *CC (Coquand and Huet)*

- Polymorphic data types on the $*$-level,
  e.g. $\Pi\alpha{:}*.\alpha{\rightarrow}(\alpha{\rightarrow}\alpha){\rightarrow}\alpha : *$.
- Predicate domains on the $\Box$-level,
  e.g. $N{\rightarrow}N{\rightarrow}* : \Box$
- Logic on the $*$-level,
  e.g. $\varphi \wedge \psi := \Pi\alpha{:}*.(\varphi{\rightarrow}\psi{\rightarrow}\alpha){\rightarrow}\alpha : *$.
- Universal quantification (first and higher order),
  e.g. $\Pi P{:}N{\rightarrow}*.\Pi x{:}N.Px{\rightarrow}Px : *$.

Examples

- Induction

$$\forall P{:}\mathbb{N}\to *\,(\ (P\,0)\to(\forall x{:}\mathbb{N}.(P\,x\to P(S\,x)))\to\forall x{:}\mathbb{N}.P\,x\ )$$

- Higher order predicates/functions: transitive closure of a relation R

$$\lambda R : A\to A\to *\,.\,\lambda x, y : A.\\ (\forall Q : A\to A\to *\,.\,(\mathsf{Trans}\,Q\to(R\subseteq Q)\to Q\,x\,y))$$

of type

$$(A\to A\to *)\to(A\to A\to *)$$

Example trans clos higher order and inductively

- transitive closure in higher order logic:

$$\lambda R : A \to A \to * \,.\, \lambda x, y : A.$$
$$(\forall Q : A \to A \to * \,.\, (\mathsf{Trans}\, Q \to (R \subseteq Q) \to Q\, x\, y))$$

  of type

$$(A \to A \to *) \to (A \to A \to *)$$

- transitive closure inductively:

```
Inductive TrclosInd (R : A->A->Prop) : A -> A -> Prop :=
| sub : forall x y : A, R x y -> TrclosInd x y
| tra : forall x y z : A,
        TrclosInd x y -> TrclosInd y z -> TrclosInd x z.
```

Exercise trans clos higher order

Given the transitive closure of a binary relation, defined in higher order logic:

$$\begin{array}{lll} \mathsf{Trclos}\, R & := & \lambda x, y{:}A. \\ & & (\forall Q{:}A{\to}A{\to}*.(\mathsf{Trans}\, Q \to (R \subseteq Q) \to Q\, x\, y)). \end{array}$$

1. Prove that the transitive closure is transitive.
2. Prove that the transitive closure of R contains R.

Higher order logic HOL

In higher order logic (originally due to Church [1940]) we have:

- higher order domains: D, $D \to \text{Prop}$, $(D \to \text{Prop}) \to \text{Prop}$, etc (sets of predicates over predicates over ...).
- higher order function domains: $(D \to D) \to D$, $((D \to D) \to D) \to D$, etc.
- $\forall$-quantification over all domains

We can do Higher Order Logic in Rocq
In Rocq we often have the choice to define sets/predicates/relations inductively or via higher order logic. The Standard Library uses inductive representations.

Full Church [1940] higher order logic

Church has additional things that we will not consider now:

- Negation connective with rules
- Classical logic

$$\frac{\Delta \vdash \neg\neg\varphi}{\Delta \vdash \varphi}$$

- Define other connectives in terms of $\rightarrow, \forall, \neg$ (classically).
- Choice operator $\iota_A : (A \rightarrow *) \rightarrow A$
- Rule for ι:

$$\frac{\Delta \vdash \exists! x{:}A.P\,x}{\Delta \vdash P(\iota_A P)}$$

Definability of other connectives (constructively)

We have already seen these when we treated $\lambda 2$:

$$\begin{array}{rcl} \bot & := & \forall \alpha{:}* . \alpha \\ \varphi \wedge \psi & := & \forall \alpha{:}* . (\varphi \to \psi \to \alpha) \to \alpha \\ \varphi \vee \psi & := & \forall \alpha{:}* . (\varphi \to \alpha) \to (\psi \to \alpha) \to \alpha \\ \exists x{:}A . \varphi & := & \forall \alpha{:}* . (\forall x{:}A . \varphi \to \alpha) \to \alpha \end{array}$$

Idea:
The definition of a connective is an encoding of the elimination rule.

Existential quantifier

$$\exists x{:}A.\varphi := \forall \alpha{:}* .(\forall x{:}A.\varphi \to \alpha) \to \alpha$$

Derivation of the elimination rule in HOL.

$$\frac{\exists x{:}A.\varphi \qquad \begin{array}{c}[\varphi]\\ \vdots \\ C\end{array}}{C}\; x \notin \mathsf{FV}(C, \mathsf{ass.}) \qquad \frac{\exists x{:}A.\varphi \qquad \begin{array}{c}[\varphi]\\ \vdots \\ C\end{array}}{\dfrac{(\forall x{:}A.\varphi \to C) \to C \qquad \forall x{:}A.\varphi \to C}{C}}$$

Equality

Equality is definable in higher order logic:

t and q terms are equal if they share the same properties (Leibniz equality)

Definition in HOL (for $t, q : A$):

$$t{=_A}q := \forall P{:}A{\to}*.(Pt \to Pq)$$

- This equality is reflexive and transitive (easy)
- It is also symmetric(!) Trick: find a "smart" predicate P

Exercise: Prove reflexivity, transitivity and symmetry of $=_A$.

CC versus HOL

Question: is the formulas-as-types embedding from HOLinto the type theory CC conswervative?

No: only if we disambiguate $*$ into Set and Prop.
This is the type theory of Rocq.

What is the problem?

- Axioms about "all propositions" in HOL will also apply to types.
  For example consider extensionality of propositions:

$$\mathsf{EXT} := \forall \varphi, \psi : \mathsf{Prop}.(\varphi \leftrightarrow \psi) \to \varphi = \psi.$$

  In CC this also applies to types, so $\mathbb{N} = \mathbb{N} \to \mathbb{N}$, which implies that $\mathbb{N}$ is a λ-model and all functions on $\mathbb{N}$ have a fixed point!

Limitations of CC

We have seen that algebraic data types are definable and we can define functions by iteration. We have a higher order predicate logic to reason over them. Distinguishing Set from Prop solves the non-conservativity, and enhances program extraction.

But

- We cannot prove $0 \neq 1$.
- We cannot prove induction

$$\forall P{:}\mathbb{N}\to * (\ (P\,0) \to (\forall x{:}\mathbb{N}.(P\,x \to P(S\,x))) \to \forall x{:}\mathbb{N}.P\,x\ ).$$

- No "primitive recursion" definition scheme for functions.

So Rocq added inductive types (+ structural recursion) as basic, and that's what is mainly used.
!! This significantly complicates the type checking algorithm !!

Properties of CC

- Uniqueness of types
  If $\Gamma \vdash M : A$ and $\Gamma \vdash M : B$, then $A =_\beta B$.
- Subject Reduction
  If $\Gamma \vdash M : A$ and $M \to_\beta N$, then $\Gamma \vdash N : A$.
- Strong Normalization
  If $\Gamma \vdash M : A$, then all β-reductions from M terminate.

Proof of SN is a really difficult.

Decidability Questions

$\Gamma \vdash M : A?$	TCP
$\Gamma \vdash M : ?$	TSP
$\Gamma \vdash ? : A$	TIP

For CC:

- TIP is undecidable
- TCP/TSP: simultaneously.
  The type checking algorithm is close to the one for λP. (In λP we had a judgement of correct context; this form of judgement could also be introduced for CC)

The Extended Calculus of Constructions has in addition

- Cumulativity: $* \subseteq \Box_0 \subseteq \Box_1 \subseteq \ldots$, so

$$\frac{\Gamma \vdash A : *}{\Gamma \vdash A : \Box_0} \qquad \frac{\Gamma \vdash A : \Box_i}{\Gamma \vdash A : \Box_{i+1}}$$

- Σ-types:

$$\frac{\Gamma \vdash A : * \;\; \Gamma, x{:}A \vdash B : *}{\Gamma \vdash \Sigma x{:}A.B : *} \qquad \frac{\Gamma \vdash A : \Box_i \;\; \Gamma, x{:}A \vdash B : \Box_j}{\Gamma \vdash \Sigma x{:}A.B : \Box_{\max(i,j)}}$$

For $\varphi : *$

- We have $\Pi A{:}\Box_i.\varphi : *$, but
- We do not have $\Sigma A{:}\Box_i.\varphi : *$.

Note: The type theory of Rocq has in addition $\mathrm{Set} : \Box$ and rules $(\mathrm{Set}, \mathrm{Set}), (\Box_i, \mathrm{Set}), (\mathrm{Set}, *)$.

Homotopy type theory (HoTT)

Fields medal 2002

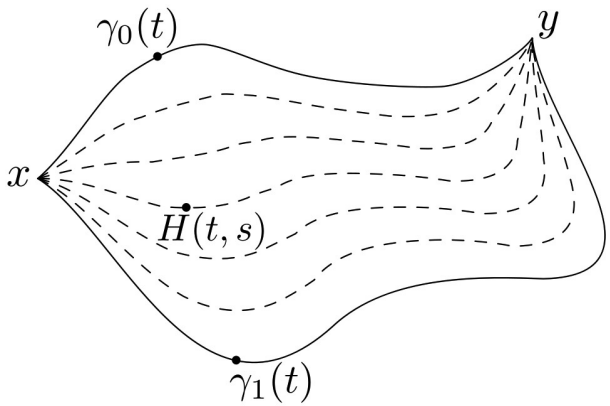
- homotopy theory algebraic varieties
- formulation of motivistic cohomology

Vladimir Voevodsky
2006

mathematics independent of specific definitions

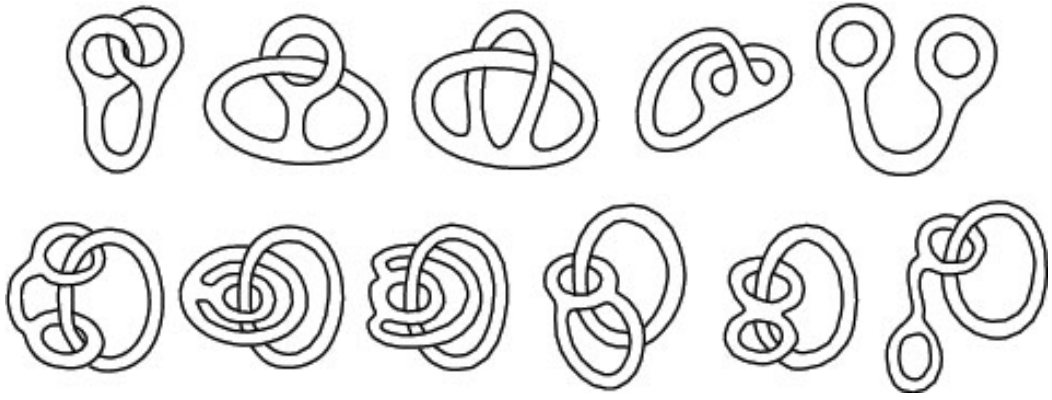
homotopy type theory

- homotopy is the 'proper' notion of equality
- homotopy = continuous transformation

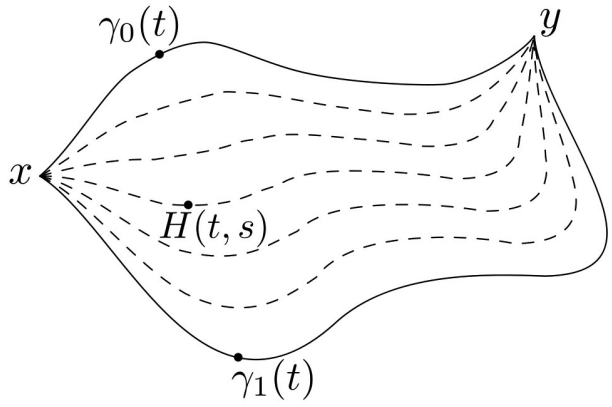


Homotopy Theory

Part of Algebraic Topology dealing with homotopy groups: associating groups to topological spaces to classify them.



- an equality is a path from one object to another (continuous transformation)
- higher equality = transformation between paths = a path between paths.



Identity type in Rocq

Identity is an inductive type

```
Inductive identity (A : Type) : A -> A -> Type :=
    refl : forall a : A, a = a.
```

The smallest binary relation on A containing $\{(x, x) \mid x : A\}$.

Giving

```
refl : forall (A : Type)(a : A), a = a.
```

and

```
ident_ind
  : forall(A : Type)(P : forall a b : A, a=b ->
Prop),
    (forall a : A, P a a refl)
    -> forall (x y : A) (i : x = y), P x y i
```

+ a reduction rule

Identity type in more mathematical presentation

The identity elimination is usually called the J-rule. We have

$$\mathsf{refl} : \forall A : \mathsf{Type}, \forall a : A, a = a$$

$$\frac{P \ : \ \forall a, b : A, a = b \to \mathsf{Prop} \qquad r \ : \ \forall a : A, P\ a\ a\ \mathsf{refl}}{J\ r \ : \ \forall x, y : A, \forall i : x = y, P\ x\ y\ i}\ \text{J-rule}$$

with computation rule

$$J\ r\ a\ a\ (\mathsf{refl}\ a) \to r.$$

Properties of the Identity type I

The J-rule gives

$$\left\{ \frac{P \;:\; \forall a, b : A, a = b \to \mathsf{Prop} \qquad r \;:\; \forall a : A, P\, a\, a\, \mathsf{refl}}{J\, r \;:\; \forall x, y : A, \forall i : x = y, P\, x\, y\, i} \text{ J-rule} \right\}$$

- Substitutivity (Leibniz property)

$$\frac{t \;:\; Q\, a \qquad p \;:\; a = b}{t' \;:\; Q\, b}$$

  But: t' is not just t.
  In fact $t' \equiv J\, a\, b\, r\, t$ (take $P\, x\, y\, i := Q\, x \to Q\, y$). This is called transport.

Note: Type theory is intensional: terms contain a lot of explicit information to guarantee that type checking is decidable.

The following "simpler" Leibniz rule would render type checking undecidable

$$\frac{t : Q(a) \qquad r : a = b}{t : Q(b)}$$

Properties of the Identity type II

The J-rule gives

$$\left\{\frac{P \;:\; \forall a, b : A, a = b \to \mathsf{Prop} \qquad r \;:\; \forall a : A, P\, a\, a\, \mathsf{refl}}{J\, r \;:\; \forall x, y : A, \forall i : x = y, P\, x\, y\, i}\ \text{J-rule}\right\}$$

- Identity is an equivalence relation:
  - $\mathsf{sym} : a = b \to b = a$ (symmetry)
    for $p : a = b$, $\mathsf{sym}\, p := J\, a\, b\, p$ (taking $P\, x\, y\, i := y = x$).
  - $\mathsf{trans} : a = b \to b = c \to a = c$ (transitivity).

Notation:
p^{-1} for $\mathsf{sym}\, p$, and $p \cdot q$ for $\mathsf{trans}\, p\, q$.

Various equations hold:
$(p^{-1})^{-1} = p$, $p \cdot p^{-1} = \mathsf{refl}$, $(p \cdot q)^{-1} = q^{-1} \cdot p^{-1}$.

Properties of the Identity type III

The J-rule does not give:

- Function extensionality

$$\frac{f, g : A \to B \qquad r : \forall a : A, f\ a = g\ a}{t : f = g}\ \text{FunExt}$$

  for some term t.

- Proof Irrelevance (all proofs are equal).

$$\frac{A : \text{Prop} \qquad a : A \qquad b : A}{t : a = b}\ \text{PI}$$

  for some term t.

- Uniqueness of Identity Proofs (UIP).

$$\frac{a, b : A \qquad q_0, q_1 : a = b}{t : q_0 = q_1}\ \text{UIP}$$

  for some term t.

Proof Irrelevance

In type theory, terms may depend on proofs, for example the reciprocal in $\mathbb{R}$ takes a proof argument:

$$\frac{1}{\_} : \forall x : \mathbb{R}.x \neq 0 \to \mathbb{R}$$

Now, if $x : \mathbb{R}$ and $q : x \neq 0$ and $r :\neq 0$, we want

$$\frac{1}{x, q} = \frac{1}{x, r}$$

which we get from PI (Proof Irrelevance), because then $q = r$. Often we can program/prove around this. In this case we have as equation for $\frac{1}{\_}$ (for all x, $p : x \neq 0$):

$$x \cdot \frac{1}{x, p} = 1 = \frac{1}{x, p} \cdot x.$$

And so:

$$\frac{1}{x, p} = \frac{1}{x, p} \cdot (x \cdot \frac{1}{x, q}) = (\frac{1}{x, p} \cdot x) \cdot \frac{1}{x, q} = \frac{1}{x, q}.$$

Uniqueness of Identity Proofs (UIP)

Isn't UIP derivable??

$$\frac{a, b : A \qquad q_0, q_1 : a = b}{t : q_0 = q_1}\ \mathrm{UIP}$$

for some term t.

The intuition of the type $a = b$ is that the only term of this type is refl (and then a and b should be the same).

UIP is equivalent to the K-rule:

$$\frac{a : A \qquad q : a = a}{t : q = \mathrm{refl}\, a\, a}\ \mathrm{K}$$

for some term t.

This rule may look even more natural ....

There is a countermodel to K (and UIP): M. Hofmann and Th. Streicher, *The groupoid interpretation of type theory*, 1998.

Types are groupoids

A type can be interpreted as a groupoid, which is defined either as

- A group where the binary operation is a partial function,
- A category in which every arrow is invertible.

A groupoid (seen as a group) should satisfy the following

- Associativity: If $p * q \downarrow$ and $q * r \downarrow$, then $(p * q) * r \downarrow$ and $p * (q * r) \downarrow$ and $(p * q) * r = p * (q * r)$.
- Inverse: $p^{-1} * p \downarrow$ and $p^{-1} * p = p * p^{-1} = 1$
- Identity: If $p * q \downarrow$, then $(p * q)^{-1} = q^{-1} * p^{-1}$.

- These are exactly the laws for our proofs of identities if we read $p * q$ as composition of proofs $p \cdot q$ (via trans) and p^{-1} as the inverse of a proof (via sym)!
- In a groupoid the K rule ($\forall p, p = 1$) obviously does not hold!

Types are topological spaces, equality proofs are paths

Voevodsky: A type A is a topological space and if $a, b : A$ with $p : a = b$, then

$$p \text{ is a continuous path from } a \text{ to } b \text{ in } A.$$

If $p, q : a = b$ and $h : p = q$, then

$$h \text{ is a continuous transformation from } p \text{ to } q \text{ in } A$$

also called a homotopy.

Equality proofs are paths, path-equalities are higher paths

Which of the following hold under the topological interpretation?

Note: A property $P : \forall ab : A, a = b \to \text{Prop}$ should be closed under continuous transformations of points and paths.

$$\frac{P : \forall ab : A, a = b \to \text{Prop} \qquad r : \forall a : A, P\ a\ a\ \text{refl}}{J\ r : \forall xy : A, \forall i : x = y, P\ x\ y\ i}\ \text{J}$$

$$\frac{a, b : A \qquad q_0, q_1 : a = b}{t : q_0 = q_1}\ \text{UIP (for some term } t\text{).}$$

$$\frac{a : A \qquad q : a = a}{t : q = \text{refl}\ a\ a}\ \text{K (for some term } t\text{).}$$

hProps, hSets

For some types we *do* have PI or UIP.
For example, for a data type like $\mathbb{N}$, we definitely want UIP.

1. A type A is an h-prop in case all terms of type A are equal (A is contractible A satisfies PI), that is, we have

$$\Pi x, y : A.\ x = y.$$

2. A type A is an h-set in case all equality proofs between terms of type A are equal. (It satisfies UIP.) That is, we have

$$\Pi x, y : A.\ \Pi p, q : x = y.\ p = q$$

And there are higher levels, called h-levels.

It can be shown that well-known data types like $\mathbb{N}$ and $\mathbb{B}$ (booleans) are h-sets.

Homotopy Type Theory

Voevodsky's Homotopy Type Theory (HoTT):

- We need to add: Univalence Axiom: for all types A and B:

$$(A = B) \simeq (A \simeq B)$$

  where $A \simeq B$ denotes that A and B are isomorphic: there are $f : A \to B$ and $g : B \to A$ such that $\forall x : A, g(f\,x) = x$ etc.

- HoTT is the internal language for homotopy theory. All proofs in homotopy theory should be formalised in type theory. (Agda and Rocq give support for that.)

Challenge: Is there a computational interpretation of the axiom UA?

Why Univalence is useful

- From UA one can prove function extensionality:

$$\frac{f, g : A \to B \qquad r : \forall a : A, f\ a = g\ a}{t : f = g}\ \text{FunExt}$$

- UA allows to treat isomorphic data types as equal data types, also called the Structure Identity Principle
  If you have an algorithm (proof) about binary $\mathbb{N}$, you have an algoritm (proof) about unary $\mathbb{N}$.

Higher Inductive Types (HITs)

Another addition of HoTT to type theory are higher inductive types: inductive types + path constructors.

```
Inductive circle : Type :=
  | base : circle
  | loop : base == base.
```

with elimination and computation rules.

```
Inductive torus : Type :=
  | base : torus
  | merid : base == base
  | equat : base == base
  | surf : merid ∘ equat = equat ∘ merid
```

Challenge: What are the proper general rules for higher inductive types?

Higher Inductive Types for data types (I)

Given a type A, we define the type $\mathcal{K}(A)$ of Kuratowski finite sets.

Higher Inductive Type $\mathcal{K}(A) :=$

$$
\begin{array}{l}
\mid \varnothing \quad : \mathcal{K}(A) \\
\mid \{\cdot\} : A \to \mathcal{K}(A) \\
\mid \cup \quad : \mathcal{K}(A) \to \mathcal{K}(A) \to \mathcal{K}(A) \\
\mid \textbf{nl} : \prod(x : \mathcal{K}(A)),\ \varnothing \cup x = x \\
\mid \textbf{nr} : \prod(x : \mathcal{K}(A)),\ x \cup \varnothing = x \\
\mid \textbf{idem} : \prod(x : A),\ \{x\} \cup \{x\} = \{x\} \\
\mid \textbf{assoc} : \prod(x, y, z : \mathcal{K}(A)),\ x \cup (y \cup z) = (x \cup y) \cup z \\
\mid \textbf{com} \quad : \prod(x, y : \mathcal{K}(A)),\ x \cup y = y \cup x \\
\mid \textbf{trunc} : \prod(x, y : \mathcal{K}(A)),\ \prod(p, q : x = y),\ p = q
\end{array}
$$

Note that the constructor **trunc**, which is a path between paths, forces the type $\mathcal{F}A$ to be an h-set.

The induction and recursion principles for $\mathcal{K}(A)$ should also transfer the paths and higher paths.

Higher Inductive Types for data types (II)

Given a type A, we define the type $\mathcal{L}(A)$ of listed finite sets.

Higher Inductive Type $\mathcal{L}(A) :=$

$\mid$ **nil** $: \mathcal{L}(A)$

$\mid$ $\cdot :: \cdot : A \to \mathcal{L}(A) \to \mathcal{L}(A)$

$\mid$ **dupl** $: \prod(a : A), \prod(x : \mathcal{L}(A)),\ a :: a :: x = a :: x$

$\mid$ **coml** $: \prod(a, b : A), \prod(x : \mathcal{L}(A)),\ a :: b :: x = b :: a :: x$

$\mid$ **truncl** $: \prod(x, y : \mathcal{L}(A)), \prod(p, q : x = y),\ p = q$

Theorem $\mathcal{K}(A) \simeq \mathcal{L}(A)$.

- Assuming UA, this equivalence becomes an equality.
- This implies that we can replace everywhere $\mathcal{K}(A)$ by $\mathcal{L}(A)$ (and vice versa) and we can use the induction (and recursion) principle of $\mathcal{K}(A)$ for $\mathcal{L}(A)$.

Questions?