Identity Types and Type Setups

Uppsala, February 18th, 2009

Stockholm-Uppsala Logic Seminar

and

Gothenberg, February 19th, 2009 .

Peter Aczel

petera@cs.man.ac.uk

SCAS and Manchester University

Part I: Identity Types

Part II: Type Setups

Some References:

[1] *Homotopy Theoretic Models of Identity Types*, Steve Awodey and Michael A. Warren

[2] *The Identity Type Weak Factorisation System*, Nicola Gambino and Richard Garner

[3] *Two-dimensional Models of Type Theory*, Richard Garner

[1] Nice categories with weak factorisation systems can be used to model type theories with identity types.

[2] The category $\mathcal{C}(\mathbb{T})$ of contexts of a type theory $\mathbb{T}$ with identity types has a natural weak factorisation system.

[3] A type theory with identity types has identity contexts.

The result in [3] is exploited in [2].

Weak Factorisation Systems

A map $g : C \to D$ *has the right lifting property with respect to* $f : A \to B$, *written* $f \pitchfork g$ if, whenever given maps $A \to C$ and $B \to D$ such that

$$A \to C \to D \;=\; A \to B \to D$$

then there is a *diagonal filler* $B \to C$; i.e.

$$A \to B \to C \;=\; A \to C \text{ and } B \to C \to D \;=\; B \to D.$$

Given a set $\mathcal{M}$ of maps let

$$\begin{aligned} \mathcal{M}^{\pitchfork} &= \{g \mid \forall f \in \mathcal{M}\ f \pitchfork g\} \\ {}^{\pitchfork}\mathcal{M} &= \{f \mid \forall g \in \mathcal{M}\ f \pitchfork g\} \end{aligned}$$

$(\mathcal{A}, \mathcal{B})$ is a *weak factorisation system* if

1. every map $A \to B$ has a factorisation $A \to Y \to B$ with $A \to Y$ in $\mathcal{A}$ and $Y \to B$ in $\mathcal{B}$, and
2. $\mathcal{A}^{\pitchfork} = \mathcal{B}$ and $\mathcal{A} = {}^{\pitchfork}\mathcal{B}$.

Theorem of Gambino and Garner

Let $\mathcal{T}$ be the set of context projections $\Gamma, \Delta \to \Gamma$ in the category of contexts of a type theory $\mathbb{T}$.

Let $\mathcal{A} = {}^{\pitchfork}\mathcal{T}$ and $\mathcal{B} = \mathcal{A}^{\pitchfork}$.

Assume that $\mathbb{T}$ has identity types.

Theorem: $(\mathcal{A}, \mathcal{B})$ is a weak factorization system.

Main Lemma: Every context map $\Gamma' \to \Gamma$ has a factorization $\Gamma' \to (\Gamma, \Delta) \to \Gamma$ where $\Gamma' \to (\Gamma, \Delta)$ is in ${}^{\pitchfork}\mathcal{T}$ and $(\Gamma, \Delta) \to \Gamma$ is in $\mathcal{T}$.

Part I: Identity Types

- Identity Propositions
- Identity types with Π and Σ types
- Avoiding Π types
- Also avoiding Σ types

Identity Propositions

Liebnitz Identity: $[\mathsf{a} = \mathsf{b}] \iff \forall \mathsf{P}\ [\mathsf{P}(\mathsf{a}) \Leftrightarrow \mathsf{P}(\mathsf{b})]$

It suffices to assume: $[a = b] \iff \forall P\ [P(a) \Rightarrow P(b)]$.

$$\forall P\ [P(a) \Rightarrow P(b)]$$

$$P'(x) \equiv [P(x) \Rightarrow P(a)]$$

$$P'(a) \Rightarrow P'(b)$$

$$P'(a)$$

$$P'(b)$$

$$P(b) \Rightarrow P(a)$$

$$P(a) \ \Leftrightarrow\ P(b)$$

$$\forall P\ [P(a) \ \Leftrightarrow\ P(b)]$$

Singleton Class Definition

Impredicative: $[a = b] \iff b \in I_a$,
where

$$I_a = \bigcap \{X \mid a \in X\}.$$

Inductive:

I_a is the smallest class X such that $a \in X$.

Reflexive Relations Definition

$$[a =_A b] \iff \forall R\, [R \text{ reflexive } \Rightarrow (a, b) \in R].$$

Impredicative: The identity relation $I_A = \{(x, x) \mid x \in A\}$ on a class A is the intersection of all reflexive relations on A.

Inductive: I_A is the smallest reflexive relation on A; i.e. the smallest relation R on A such that

$$\forall x \in A\ (x, x) \in R.$$

Adjoint characterisations of $=_A$

Reflexive Relations:

$$\frac{[x =_A y] \vdash_{x,y} Q(x,y)}{\vdash_x Q(x,x)}$$

Singleton Class:

$$\frac{[a =_A y] \vdash_y P(y)}{\vdash P(a)} \; (a \in A)$$

Type Theoretical Logical Rules,1

Singleton Class: For $a : A$

$$[a =_A y]\ prop\ (y : A)$$

$$[a =_A a]\ true$$

$$\frac{\begin{array}{l} D(y)\ prop\ (y : A, [a =_A y]\ true) \\ D(a)\ true \end{array}}{D(y)\ true\ (y : A, [a =_A y]\ true)}$$

Type Theoretical Logical Rules,2

Reflexive Relations:

$$[x =_A y]\ prop\ (x, y : A)$$

$$[x =_A x]\ true\ (x : A)$$

$$\frac{C(x,y)\ prop\ (x, y : A, [x =_A y]\ true) \qquad C(x,x)\ true\ (x : A)}{C(x,y)\ true\ (x, y : A, [x =_A y]\ true)}$$

Identity Types with Π and Σ types

Identity Types,1: Given A *type*:

Formation:

$$I_A(x,y)\ type(x,y:A)$$

Introduction:

$$r_A(x) : I_A(x,x)\ (x:A)$$

Elimination/Computation

$$\frac{\begin{array}{ll} C(x,y,z)\ type & (x,y:A, z:I_A(x,y)) \\ d(x) : C(x,x,r_A(x)) & (x:A) \end{array}}{\begin{array}{ll} J_d(x,y,z) : C(x,y,z) & (x,y:A, z:I_A(x,y)) \\ J_d(x,x,r_A(x)) = d(x) : C(x,x,r_A(x)) & (x:A) \end{array}}$$

These are the standard rules for Identity types.

Identity Types,2: Given $a : A$:

Formation:

$$I_a(y)\ type(y : A)$$

Introduction:

$$r_a : I_a(a)$$

Elimination/Computation

$$\frac{\begin{array}{ll} D(y,z)\ type & (y : A, z : I_a(y)) \\ e : D(a, r_a) & \end{array}}{\begin{array}{ll} J'_{a,e}(y,z) : D(y,z) & (y : A, z : I_a(y)) \\ J'_{a,e}(a, r_a) = e : D(a, r_a) & \end{array}}$$

These rules are due to Christine Paulin-Mohring.

J versus J'

It is easy to define J using J'.

But it is not so easy to define J' using J.

Martin Hoffman showed that this could be done. A construction is presented as an appendum in Thomas Streicher's Habilitation Thesis. But it is almost unreadable because of the awful syntax used.

The construction uses Π-types and Σ-types. But by using a parametric strengthening of the J-rule, due to Richard Garner, Π-types can be avoided and, by using ideas also due to Garner, and more work Σ-types can also be avoided.

The following is essentially Hofmann's construction.

Definition of J' using J, 1:

Given I_A and $a : A$:

Step 1: Define, for $x, y : A, z : I_A(x, y)$,

$$
\begin{array}{ll}
I_a(y) & \equiv I_A(a, y) \\
r_a & \equiv r_A(a) \\
A_0(x) & \equiv (\Sigma x' : A) I_A(x, x') \\
C(x, y, z) & \equiv I_{A_0(x)}(< x, r_A(x) >, < y, z >) \\
d(x) & \equiv r_{A_0(x)}(< x, r_A(x) >) : C(x, x, r_A(x))
\end{array}
$$

Use the J rule with C, d to define

$$f(x, y, z) \equiv J_d(x, y, z) : C(x, y, z)$$

such that $f(x, x, r_A(x)) = d(x) : C(x, x, r_A(x))$.

Definition of J' using J, 2:

Given also $D(y, z)\ type\ (y : A, z : I_a(y))$:

Step 2: Define $A_1 \equiv A_0(a)$ and, for $x_1, y_1 : A_1, z_1 : I_{A_1}(x_1, y_1)$,

$$\begin{aligned} B_1(x_1) &\equiv D(\pi_1(x_1), \pi_2(x_1)) \\ C_1(x_1, y_1, z_1) &\equiv B_1(x_1) \rightarrow B_1(y_1) \\ d_1(x_1) &\equiv (\lambda u : B_1(x_1))u : C_1(x_1, x_1, r_{A_1}(x_1)) \end{aligned}$$

Use the J rule with C_1, d_1 to define

$$g(x_1, y_1, z_1) \equiv J_d(x_1, y_1, z_1) : C_1(x, y, z)$$

such that

$$g(x_1, x_1, r_{A_1}(x_1)) = d_1(x_1) : C_1(x_1, x_1, r_{A_1}).$$

Definition of J' using J, 3:

Given a, D as before and $e : D(a, r_a)$:

Step 3: Define, for $y : A, z : I_a(y)$,

$$a_1 \equiv < a, r_a >: A_1$$

$$J'_{a,e}(y,z) \equiv app(g(a_1, < y, z >, f(a, y, z)), e) : D(y, z)$$

Then

$$\begin{aligned} J'_{a,e}(a, r_a) &= app(g(a_1, a_1, f(a, a, r_a)), e) \\ &= app(g(a_1, a_1, r_{A_1}(a_1)), e) \\ &= app((\lambda u : B_1(a))u, e) \\ &= e : D(a, r_a). \end{aligned}$$

Avoiding Π types

The parametric J-rule:

For $x, y : A, z : I_A(x, y)$,

$$\frac{\begin{array}{ll} C(x, y, z, \vec{u})\ type & (\vec{u} : \vec{E}(x, y, z))) \\ d(x, \vec{u}) : C(x, x, r_A(x), \vec{u}) & (\vec{u} : \vec{E}(x, x, r_A(x))) \end{array}}{\begin{array}{ll} J_d(x, y, z, \vec{u}) : C(x, y, z, \vec{u}) & (\vec{u} : \vec{E}(x, y, z)) \\ J_d(x, x, r_A(x), \vec{u}) = d(x, \vec{u}) : C(x, x, r_A(x), \vec{u})) & (\vec{u} : \vec{E}(x, x, r_A(x))) \end{array}}$$

$\vec{u} : \vec{E}(x, y, z)$ is the context of parameters relative to the declarations of x, y, z.

$\vec{u} : \vec{E}(x, x, r_A(x))$ is the resulting context of parameters relative to the declaration of x after substituting x for y and $r_A(x)$ for z.

The parametric substitution rule:

For $x, y : A, z : I_A(x, y), \vec{u} : \vec{E}(x)$,

$$\frac{B(x, \vec{u})\ type}{\begin{array}{ll} sub(x, y, z, \vec{u}, v) : B(y, \overrightarrow{sub}(x, y, z, \vec{u})) & (v : B(x, \vec{u})) \\ sub(x, x, r_A(x), \vec{u}, v) = v : B(x, \vec{u}) & (v : B(x, \vec{u})) \end{array}}$$

where, if $\vec{u} \equiv u_1, \ldots, u_n$ then $\overrightarrow{sub}(x, y, z, \vec{u}) \equiv u'_1, \ldots, u'_n$ with $u'_i \equiv sub(x, y, z, u'_1, \ldots, u'_{i-1}, u_i)\ \ (i = 1, \ldots, n)$.

This can be derived using the parametric J-rule with $C(x, y, z, \vec{u}, v) \equiv B(y, \overrightarrow{sub}(x, y, z, \vec{u}))$ and $d(x, \vec{u}, v) \equiv v$.

Definition of J' using the parametric J-rule

The aim here is to avoid Π-types by using the parametric J-rule. As in the earlier Step 1, we can use the J-rule to define, for $x, y : A, z : I_A(x, y)$,

$$f(x, y, z) : I_{A_0(x)}(x_1, < y, z >),$$

where $A_0(x) \equiv (\Sigma x' : A) I_A(x, x')$ and $x_1 \equiv < x, r_A(x) >$, such that $f(x, x, r_A(x)) = r_{A_0(x)}(x_1)$.

Given a, D, e we can now use substitution (without parameters) to define, for $y : A, x : I_A(a, y)$,

$$J'_{a,e}(y, z) \equiv \;\; sub(< a, r_A(a) >, < y, z >, f(a, y, z), e) : D(y, z).$$

We have still used Σ-types, which we want to avoid.

Also avoiding Σ types

Definition of J' avoiding Σ-types,1

Given A, D, e we first use parametric substitution with one parameter $v_1 : I_A(a,x)$ and $B(y, v_1) \equiv D(y, v_1)$. So we get, with $x, y : A$, $z : I_A(x,y)$ and $v_1 : I_A(a,x)$,

$$sub(x,y,z,v_1,u) : B(y, sub(x,y,z,v_1)) \quad (u : B(x,v_1))$$

such that $sub(x,x,r_A(x),v_1,u) = u : B(x,v_1)$ $(u : B(x,v_1))$
Here $sub(x,y,z,v_1) : I_A(a,y)$ such that

$$sub(x,x,r_A(x),v_1) = v_1 : I_A(a,x).$$

Now put $x = a, v_1 = r_A(a), u = e$ and define, for $y : A, z : I_A(a,y)$,

$$\begin{aligned} h_{a,e}(y,z) &\equiv sub(a,y,z,r_A(a),e) \\ f_a^1(y,z) &\equiv sub(a,y,z,r_A(a)). \end{aligned}$$

Definition of J' avoiding Σ-types,2

For $y : A, z : I_A(a, y)$, we have $h_{a,e}(y, z) : D(y, f_a^1(y, z))$ and $f_a^1(y, z) : I_A(a, y)$ such that

$$\begin{aligned} h_{a,e}(a, r_A(a)) &= e : D(a, r_A(a)) \\ f_a^1(a, r_A(a)) &= r_A(a) : I_A(a, a). \end{aligned}$$

We use the J-rule with $C(x, y, z) \equiv I_{I_A(x,y)}(sub(x, y, z, v_1), z)$ and $d(x) \equiv r_{I_A(x,x)}(r_A(x))$ to get

$$f_a^2(y, z) = J_d(a, y, z) : I_{I_A(a,y)}(f_a^1(y, z), z)$$

such that $f_a^2(a, r_A(a)) = r_{I_A(a,a)}(r_A(a))$.

Definition of J' avoiding Σ-types,3

Given $y : A$, let $A' = I_A(a, y)$. For $z : A'$, we use substitution, with $B(z) \equiv D(y, z)$ to get

$$sub(z', z, w, u) : B(z) \ (z', w : I_{A'}(z', z), u : B(z'))$$

such that $sub(z', z', r_{A'}(z'), u) = u : B(z') \ (z' : A', u : B(z'))$. We can now define, for $y : A, z : A'$,

$$J'_{a,e}(y, z) \equiv sub(f_a^1(y, z), z, f_a^2(y, z), h_{a,e}(y, z)) : D(y, z),$$

and get, as $h_{a,e}(a, r_A(a)) = e$,

$$\begin{aligned} J'_{a,e}(a, r_A(a)) &= sub(f_a^1(a, r_A(a)), r_A(a), f_a^2(a, r_A(a)), e) \\ &= sub(r_A(a), r_A(a), r_{I_A(a,a)}(r_A(a)), e) \\ &= e : D(a, r_A(a)) \end{aligned}$$

Part II: Type Setups

A Motivation for Type Setups,1

If $\Gamma \equiv x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n$ is a context it is natural to write $\Gamma \equiv \vec{x} : \vec{A}$, where

$$\vec{x} \equiv x_1, x_2, \ldots, x_n \text{ and } \vec{A} \equiv A_1, [x_1]A_2, \ldots, [x_1, \ldots, x_{n-1}]A_n.$$

We then write $\vec{a} : \vec{A}$ for the sequence of judgments

$$a_1 : A_1, a_2 : A_2[a_1/x_1], \ldots, a_n : A_n[a_1, \ldots, a_{n-1}/x_1, \ldots, x_{n-1}]$$

where $\vec{a} \equiv a_1, \ldots, a_n$. So

- $\vec{A}$ is like a single type,
- $\vec{x} : \vec{A}$ is like a single variable declaration
- $\vec{a} : \vec{A}$ is like a single judgement

A Motivation for Type Setups,2

Let $\Delta \equiv y_1 : B_1, \ldots, y_m : B_m$ such that Γ, Δ is a context. It is natural to write $\Delta \equiv \vec{y} : \vec{B}(\vec{x})$, where

$$\vec{B}(\vec{x}) \equiv B_1, [y_1]B_2, \ldots, [y_1, \ldots, y_m]B_m.$$

Then, if $\vec{a} : \vec{A}$,

$$\begin{aligned} \Delta[\vec{a}/\vec{x}] &\equiv y_1 : B_1[\vec{a}/\vec{x}], \ldots, y_m : B_m[\vec{a}/\vec{x}] \\ &\equiv \vec{y} : \vec{B}(\vec{a}) \end{aligned}$$

So $\vec{B}(\vec{x})$ is like a family of types over the type $\vec{A}$. We have a new type theory. To make this precise we need an abstract notion of type theory. This is the notion of a

TYPE SETUP.

A Motivation for Type Setups,3

If $\mathbb{T}$ is a type setup let $\mathbb{T}^*$ be the new type setup, constructed along the lines we have described.

Some conjectured results:

- $\mathbb{T}^*$ is indeed a type setup and has Σ-types. It is the ‘free’ type setup with Σ-types generated from $\mathbb{T}$.
- (Garner) If $\mathbb{T}$ has identity types then so does $\mathbb{T}^*$.
- $\mathbb{T}$ and $\mathbb{T}^*$ have equivalent categories of contexts.

Conclusion:

We may as well assume that a type theory/setup has Σ-types.

Category notions for the semantics of type dependency

- Category with attributes Cartmell 1978, Moggi 1991, Type category Pitts 1997
- Contextual category Cartmell 1978, Streicher 1991
- Category with families Dybjer 1996, Hoffman 1997
- Category with display maps (less general) Taylor 1986, Lamarche 1987, Hyland and Pitts 1989
- Comprehension category (more general) Jacobs 1991
- other relevant notions: locally cartesian closed categories, fibrations, indexed categories
- Type setups (for syntax) new notion

Category with families (CwF)

- a category $Ctxt$ of contexts Γ and substitutions $\sigma : \Delta \to \Gamma$, with a distinguished terminal object $(\ )$,
- a functor $T : Ctxt^{op} \to Fam$ mapping

$$\Gamma \mapsto \{Term(\Gamma, A)\}_{A \in Type(\Gamma)}$$

  and, if $\sigma : \Delta \to \Gamma$ then

$$\begin{array}{ll} A \in Type(\Gamma) & \mapsto A\sigma \in Type(\Delta) \\ a \in Term(\Gamma, A) & \mapsto a\sigma \in Term(\Delta, A\sigma) \end{array}$$

- an assignment, to each context Γ and each $A \in Type(\Gamma)$, of a comprehension $(\Gamma.A, \mathsf{p}_A, \mathsf{v}_A)$ such that

$$\mathsf{p}_A : \Gamma.A \to \Gamma \text{ and } \mathsf{v}_A \in Term(\Gamma.A, A\mathsf{p}_A);$$

  i.e. a terminal object in the category of (Γ', θ, a) such that $\theta : \Gamma' \to \Gamma$ and $a \in Term(\Gamma', A\theta)$.

The large CwF of sets

- $Ctxt = Set$ and, for each set I,
- $Type(I)$ is the class of families of sets $A = \{A_i\}_{i \in I} \in Set^I$,
- $Term(I, A) = \prod_{i \in I} A_i$ and, if $\sigma : J \to I$ in Set,
- $A\sigma = \{A_{\sigma j}\}_{j \in J}$,
- $a\sigma = \{a_{\sigma j}\}_{j \in J}$, for $a = \{a_i\}_{i \in I}$.
- $I.A = \sum_{i \in I} A_i$,
- $\mathsf{p}_A(i, x) = i$ for $(i, x) \in I.A$,
- $\mathsf{v}_A = \{x\}_{(i,x) \in I.A}$.

Type Setups, 1

The notion of a type setup abstracts away from the details of how terms and types are formed, but keeps the following notions.

- contexts Γ,
- substitutions $\sigma : \Delta \rightarrow \Gamma$, between contexts, the contexts and substitutions forming a category $Ctxt$,
- $\iota_\Gamma : \Gamma \rightarrow \Gamma$ is the identity on Γ and $\sigma \circ \tau : \Lambda \rightarrow \Gamma$ is the composition of $\sigma : \Delta \rightarrow \Gamma$ and $\tau : \Lambda \rightarrow \Delta$.
- For each context Γ, there is the set $Type(\Gamma)$ of Γ-types A and the set $Term(\Gamma, A)$ of Γ-terms a of type A, for each Γ-type A.
- Substitutions must 'act' on types and terms to give a functor $T : Ctxt^{op} \rightarrow Fam$, where Fam is the category of set-indexed families of sets.

Type Setups, 2

- For each context Γ

$$T(\Gamma) = \{Term(\Gamma, A)\}_{A \in Type(\Gamma)}$$

- For each substitution $\sigma : \Delta \to \Gamma$, $T(\sigma) : T(\Gamma) \to T(\Delta)$ maps

$$\begin{array}{ll} A \in Type(\Gamma) & \mapsto A\sigma \in Type(\Delta) \\ a \in Term(\Gamma, A) & \mapsto a\sigma \in Term(\Delta, A\sigma) \end{array}$$

- such that

$$A\iota_\Gamma = A \text{ and } a\iota_\Gamma = a$$

and if also $\tau : \Lambda \to \Delta$ then

$$A(\sigma \circ \tau) = (A\sigma)\tau \text{ and } a(\sigma \circ \tau) = (a\sigma)\tau.$$

Type Setups, 3

- Each context Γ is a finite sequence

$$x_1 : A_1, \ldots, x_n : A_n$$

of typed variable declarations.

- The empty sequence $(\ )$ is a context.

- If $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$ then

$\Gamma' \equiv \ \Gamma, x : A \ \equiv \ x_1 : A_1, \ldots, x_n : A_n, x : A$ is a context iff
  - Γ is a context,
  - x is a variable, not in $\{x_1, \ldots, x_n\}$ and
  - $A \in Type(\Gamma)$.

Type Setups, 4

- If Γ, Δ are contexts, with

$$\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$$

the each substitution $\Delta \to \Gamma$ has the form

$$[x_1 := a_1, \ldots, x_n := a_n]_{\Delta \to \Gamma}.$$

- If $\Gamma' \equiv x_1 : A_1, \ldots, x_n : A_n, x : A$ is a context then

$$\sigma' \equiv [\sigma, x := a]_{\Delta \to \Gamma'} \equiv [x_1 := a_1, \ldots, x_n := a_n, x := a]_{\Delta \to \Gamma'}$$

is a substitution $\Delta \to \Gamma'$ iff

$\sigma \equiv [x_1 := a_1, \ldots, x_n := a_n]_{\Delta \to \Gamma}$ is a substitution, and
$a \in Term(\Delta, A\sigma)$.

Type Setups, 5

- If $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$ is a context then, for $i = 1, \ldots, n$,

$$A_i \in Type(\Gamma) \text{ and } x_i \in Term(\Gamma, A_i).$$

- If $\sigma \equiv [x_1 := a_1, \ldots, x_n := a_n]_{\Delta \to \Gamma}$ is a substitution then it is

the unique substitution $\Delta \to \Gamma$ such that, for $i = 1, \ldots, n$,

$$x_i \sigma = a_i.$$

Type Setups, 6

- If Γ, Δ are contexts such that $\Gamma \subseteq \Delta$ (i.e. every declaration in Γ is also a declaration in Δ) then

$$Type(\Gamma) \subseteq Type(\Delta) \text{ and } Term(\Gamma, A) \subseteq Term(\Delta, A)$$

for each $A \in Type(\Gamma)$.

- Also, if $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$ then

$$\iota_{\Delta\to\Gamma} \equiv [x_1 := x_1, \ldots, x_n := x_n]_{\Delta\to\Gamma}$$

is an inclusion substitution; i.e. for $A \in Type(\Gamma)$ and $a \in Term(\Gamma, A)$,

$$A\iota_{\Delta\to\Gamma} = A \text{ and } a\iota_{\Delta\to\Gamma} = a.$$

- If $\Gamma' \equiv \Gamma, x : A$ then $(\Gamma', \iota_{\Gamma'\to\Gamma}, x)$ is a comprehension.

Π-types, 1

- We say that a type setup has Π-types if the standard formation, introduction, elimination and computation rules for Π-types are correct for the type setup; i.e. if $\Gamma' \equiv \Gamma, x : A$ is a context then there are the following assignments:

$$B \in Type(\Gamma') \mapsto (\Pi x : A)B \in Type(\Gamma),$$

$$b \in Term(\Gamma', B) \mapsto (\lambda x)b \in Term(\Gamma, (\Pi x : A)B),$$

$$\left.\begin{array}{l} f \in Term(\Gamma, (\Pi x : A)B) \\ a \in Term(\Gamma, A) \end{array}\right\} \mapsto app(f, a) \in Term(\Gamma, B[a/x])$$

such that if $f = (\lambda x)b$ then $app(f, a) = b[a/x]$.

Π-types, 2

- These must commute with substitution; i.e. for each $\sigma : \Delta \to \Gamma$,

$$\begin{aligned} ((\Pi x : A)B)\sigma &= (\Pi x : A\sigma)B\sigma', \\ ((\lambda x)b)\sigma &= (\lambda x)b\sigma', \\ app(f, a)\sigma &= app(f\sigma, a\sigma), \end{aligned}$$

where $\sigma' \equiv [\sigma, x := x]_{\Delta \to \Gamma'} : \Delta \to \Gamma'$.

- Also, if $y \notin var(\Gamma)$ then

$$(\Pi x : A)B = (\Pi y : A)B[y/x] \text{ and } (\lambda x)b = (\lambda y)b[y/x].$$

- The requirement that the type setup has other forms of type can be explained in a similar way.