Towards a Proof-Irrelevant Calculus of Inductive Constructions

Irrelevant Propositions for Type Theory

Philipp Haselwarter[1]
Supervisor: Matthieu Sozeau[1,2]

[1] Preuves, Programmes et Systèmes (Paris 7)

[2] Équipe-Projet πr² (INRIA Rocquencourt)

Stage de M2 MPRI

Outline

Revisiting Curry-Howard

- in Coq / the predicative Calculus of Inductive Constructions (pCIC):

Prop	Type
(formal) proof	program
logic/specification	data
cut-elimination	computation
conj a b : $A \wedge B$	pair a b : $A * B$

Revisiting Curry-Howard

- in Coq / the predicative Calculus of Inductive Constructions (pCIC):

Prop	Type
(formal) proof	program
logic/specification	data
cut-elimination	computation
conj a b : $A \wedge B$	pair a b : $A * B$

- an *isomorphism*?

Revisiting Curry-Howard

- in Coq / the predicative Calculus of Inductive Constructions (pCIC):

Prop	Type
(formal) proof	program
logic/specification	data
cut-elimination	computation
conj a b : $A \wedge B$	pair a b : $A * B$

- an *isomorphism*?
  What are the *theorems* corresponding to these *proofs*?

$$0 : \mathbb{N}, \quad (+) : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$$

  What do the *programs* implementing these theorems *compute*?

$$p, q : m + n = n + m$$

Revisiting Curry-Howard

- in Coq / the predicative Calculus of Inductive Constructions (pCIC):

Prop	Type
(formal) proof	program
logic/specification	data
cut-elimination	computation
conj a b : $A \wedge B$	pair a b : $A * B$

- an *isomorphism*?
  What are the *theorems* corresponding to these *proofs*?

$$0 : \mathbb{N}, \quad (+) : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$$

  What do the *programs* implementing these theorems *compute*?

$$p, q : m + n = n + m$$

- a *correspondence*!

Putting Curry-Howard to Use

```
let rec rev = function
  | [] -> []
  | h::t -> (rev t) @ [h]

Definition vec (A : Type) (n : nat) := {l : list A | n = length l}.

Fixpoint vrev : ∀ A n (v : vec A n), vec A n := …
```

Theorem vrev_inv

$\forall\,(A : \mathrm{Type})\,(n : \mathrm{nat}),\ \forall\,(v : \mathrm{vec}\ A\ n),\ \mathrm{vrev}\,(\mathrm{vrev}\ v) = v.$

But we get stuck!

```
A : Type     n : nat     h : A
l, t, l_rev_rev : list A
l_case : l = h :: t
IHt : l = t
      → ∀ (n : nat) (e : n = length t),
        exist (λ l0 : list A, n = length l0) t e =
        rev' A n (rev' A n (exist (λ l0 : list A, n = length l0) t e))
eq_l_rev_rev : l = l_rev_rev
len_l : n = length l
len_rev_rev : n = length l
============================
 exist (λ l' : list A, n = length l') l len_l =
 exist (λ l' : list A, n = length l') l len_rev_rev

Abort.
```

What went wrong?

- comparing proofs structurally while we only care for their existence
- Prop not "faithful" to intuition about propositions

Poincaré Principle

Obvious computations do not require formal justification.

...but some "obvious" computations do *not* hold!

The Solution: Proof-Irrelevance

The missing reasoning principle:

Definitional Proof-Irrelevance

$$\forall\,(P : \mathsf{Prop}),\ \forall\,(p\ q : P),\ p \equiv q$$

- Any two proofs of the same proposition are indistinguishable.
- adding the *propositional* axiom can break progress of computations
- missing from Coq as a target language for Program

As an axiom: *reduction* gets stuck

```
A : Type     n : nat     h : A
l, t, l_rev_rev : list A
l_case : l = h :: t
IHt : l = t
      → ∀ (n : nat) (e : n = length t),
        exist (λ l0 : list A, n = length l0) t e =
        rev' A n (rev' A n (exist (λ l0 : list A, n = length l0) t e))
eq_l_rev_rev : l = l_rev_rev
len_l : n = length l
len_rev_rev : n = length l
proof_irrelevance : ∀ (P : Prop) (p q : P), p = q
============================
 exist (λ l' : list A, n = length l') l len_l =
 exist (λ l' : list A, n = length l') l len_rev_rev

rewrite (proof_irrelevance _ len_rev_rev len_l). reflexivity.
```

Section 2

Towards A Proof-Irrelevant Calculus of Constructions (πCIC)

Lift: From Prop to Type...

$$\frac{\Gamma \vdash A : \mathsf{Prop}}{\Gamma \vdash \{A\} : \mathsf{Type}_0}\ \textsc{Lift-form} \qquad \frac{\Gamma \vdash A : \mathsf{Prop} \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathsf{prf}\, a : \{A\}}\ \textsc{Lift-intro}$$

$$\frac{\Gamma \vdash a : \{A\}}{\Gamma \vdash a.\mathsf{prf} : A}\ \textsc{Lift-elim}$$

- Prop: a universe separated from Type
- *explicit* inclusion through the Lift type (Coq: implicit/inferred)
- proofs are *trivial* data

Lift: From Prop to Type...

$$\frac{\Gamma \vdash A : \mathsf{Prop}}{\Gamma \vdash \{A\} : \mathsf{Type}_0}\ \textsc{Lift-form} \qquad \frac{\Gamma \vdash A : \mathsf{Prop} \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathsf{prf}\, a : \{A\}}\ \textsc{Lift-intro}$$

$$\frac{\Gamma \vdash a : \{A\}}{\Gamma \vdash a.\mathsf{prf} : A}\ \textsc{Lift-elim}$$

- Prop: a universe separated from Type
- *explicit* inclusion through the Lift type (Coq: implicit/inferred)
- proofs are *trivial* data

Prop has Proof-Irrelevance

$$\frac{\Gamma \vdash u : \{A\} \qquad \Gamma \vdash v : \{A\}}{\Gamma \vdash u \equiv v : \{A\}}\ \textsc{Lift-irrel}$$

- implements definitional proof-irrelevance
- does The Right Thing™:
  $A : \mathsf{Prop}$ by inversion
- the conversion algorithm will be *type directed*

...and from Type to Prop: Truncation

- start from a relevant ("informative") type A

$$\frac{\Gamma \vdash A : \mathsf{Type}_i}{\Gamma \vdash \|A\| : \mathsf{Prop}}\ \textsc{Trunc-form}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash A : \mathsf{Type}_i}{\Gamma \vdash |a| : \|A\|}\ \textsc{Trunc-intro}$$

$$\frac{\Gamma \vdash x : \|A\| \qquad \Gamma \vdash y : \|A\|}{\Gamma \vdash x \equiv y : \|A\|}\ \textsc{Trunc-irrel}$$

...and from Type to Prop: Truncation

- start from a relevant ("informative") type A
- reduce the information to its mere *inhabitation*

$$\frac{\Gamma \vdash A : \mathsf{Type}_i}{\Gamma \vdash \|A\| : \mathsf{Prop}}\ \textsc{Trunc-form}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash A : \mathsf{Type}_i}{\Gamma \vdash |a| : \|A\|}\ \textsc{Trunc-intro}$$

$$\frac{\Gamma \vdash x : \|A\| \qquad \Gamma \vdash y : \|A\|}{\Gamma \vdash x \equiv y : \|A\|}\ \textsc{Trunc-irrel}$$

...and from Type to Prop: Truncation

- start from a relevant ("informative") type A
- reduce the information to its mere *inhabitation*
- the exact proof is thus *irrelevant*

$$\frac{\Gamma \vdash A : \mathsf{Type}_i}{\Gamma \vdash \|A\| : \mathsf{Prop}}\ \textsc{Trunc-form}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash A : \mathsf{Type}_i}{\Gamma \vdash |a| : \|A\|}\ \textsc{Trunc-intro}$$

$$\frac{\Gamma \vdash x : \|A\| \qquad \Gamma \vdash y : \|A\|}{\Gamma \vdash x \equiv y : \|A\|}\ \textsc{Trunc-irrel}$$

$$\frac{\Gamma \vdash a : \|A\| \qquad \Gamma \vdash P : \mathsf{Prop} \qquad \Gamma, x : A \vdash p : P}{\Gamma \vdash \mathsf{let}\ |x| := a\ \mathsf{in}\ p : P}\ \textsc{Trunc-elim}$$

- allow *some* unboxing to actually use truncated values
- but irrelevance mandates a restricted elimination
- as Prop is closed for irrelevance, eliminating to $P : \mathsf{Prop}$ is okay:

$$\forall\ (a\ b : A),\ p[a/x] \equiv p[b/x]\ :\ P \text{ anyways}$$

- similar to the construction by Awodey and Bauer, but with decidable premises

Example: Elimination of a truncation

let $bool : \mathsf{Type}_0$.

we can construct a function $f : \|A + B\| \to \|bool\|$

$$\pi\mathrm{CIC} \vdash f = \lambda x : \{\|A + B\|\} . \mid \mathsf{let}\ |c| := x\ \mathsf{in}$$

$$\mathsf{case}\ c\ \mathsf{as}\ z\ \mathsf{return}\ \mathsf{bool}\ \mathsf{of}$$

$$\mid \mathsf{inl}\ \_ \Rightarrow \mathsf{false}$$

$$\mid \mathsf{inr}\ \_ \Rightarrow \mathsf{true}$$

$$\mid$$

Example: Elimination of a truncation

let $bool : \mathsf{Type}_0$.

we can construct a function $f : \|A + B\| \to \|bool\|$

$$\pi\mathrm{CIC} \vdash f = \lambda x : \{\|A + B\|\} . \; | \; \mathsf{let}\ |c| := x\ \mathsf{in}$$

$$\mathsf{case}\ c\ \mathsf{as}\ z\ \mathsf{return}\ \mathsf{bool}\ \mathsf{of}$$

$$| \ \mathsf{inl}\ \_ \Rightarrow \mathsf{false}$$

$$| \ \mathsf{inr}\ \_ \Rightarrow \mathsf{true}$$

$$|$$

but not without truncation:

$$\pi\mathrm{CIC} \not\vdash \|A + B\| \to bool$$

Eliminating Contradictions

- True, False : Prop
  → indicates (ir-) relevance
  → False does not have any values, cannot be used in any relevant way, even inside a program
- False-ELIM to any sort: information flows back from Prop to Type
- $\Gamma \vdash f$: False indicates contradictory context Γ
- computational interpretation of a contradiction: dead code

$$\frac{\ldots}{A : \mathsf{Type}_i,\ t : \{\mathsf{False}\} \vdash !_A\ t.\mathsf{prf} : A}\ \text{False-ELIM}$$

Uninformative Equalities: Elimination

- a type for equality: $Eq\ (A : \mathsf{Type})(x : A) : A \to \mathsf{Type}$
- equality is *relevant* by default
- truncated equalities have UIP / Streicher's K by TRUNC-IRREL

$$\frac{\Gamma \vdash e : \{\|a =_A b\|\} \qquad \Gamma, x : A, y : A, p : \{\|x =_A y\|\} \vdash C : s \qquad \Gamma, z : A \vdash c : C\left[z/x, z/y, \mathsf{prf}\left|\mathsf{refl}_A z\right|/p\right]}{\Gamma \vdash \mathsf{ind}_{\|=\|}\ x.y.p.C\ z.c\ a\ b\ e : C[a/x, b/y, e/p]}\ \textsc{trunc-=-elim}$$

Uninformative Equalities: Elimination

- a type for equality: $Eq\ (A : \mathsf{Type})(x : A) : A \to \mathsf{Type}$
- equality is *relevant* by default
- truncated equalities have UIP / Streicher's K by TRUNC-IRREL

$$\frac{\Gamma \vdash e : \{\|a =_A b\|\} \qquad \Gamma, x:A, y:A, p:\{\|x =_A y\|\} \vdash C : s \qquad \Gamma, z:A \vdash c : C\left[z/x, z/y, \mathsf{prf}\left|\mathsf{refl}_A z\right|/p\right]}{\Gamma \vdash \mathsf{ind}_{\|=\|}\ x.y.p.C\ z.c\ a\ b\ e : C[a/x, b/y, e/p]}\ \textsc{trunc-=-elim}$$

- the proof remains opaque!
- we can still *rewrite* with irrelevant equalities
- essential for dependent pattern matching and defining inductive families

Uninformative Equalities: Reduction

$$\frac{\Gamma \vdash a \equiv b : A}{\Gamma \vdash \mathsf{ind}_{\|=\|}\ x.y.p.C\ z.c\ a\ b\ e \equiv c[a/z] : C[a/x, a/y, e/p]}\ \textsc{trunc-=-comp}$$

- but relevant and irrelevant equalities coexist in ΠCIC!
- proof is not unboxed/inspected
  → more efficient reduction than trying to reduce equality-proofs to refl
- no big penalty for specifying programs
- like Werner's proof-erased equality and substitution in Observational Type Theory

Representing Predicate Logic

False, True, and truncations are in Prop:

$$
\begin{aligned}
[\![\bot]\!] &:\equiv \mathsf{False} \\
[\![\top]\!] &:\equiv \mathsf{True} \\
[\![A \wedge B]\!] &:\equiv \left\| \{[\![A]\!]\} * \{[\![B]\!]\} \right\| \\
[\![A \vee B]\!] &:\equiv \left\| \{[\![A]\!]\} + \{[\![B]\!]\} \right\| \\
[\![A \Rightarrow B]\!] &:\equiv \{[\![A]\!]\} \to [\![B]\!] \\
[\![\forall x : A, B]\!] &:\equiv \prod_{(x:A)} [\![B]\!] \\
[\![\exists x : A, B]\!] &:\equiv \left\| \sum_{(x:A)} [\![B]\!] \right\| \\
[\![a =_A b]\!] &:\equiv \left\| a =_A b \right\|
\end{aligned}
$$

- interpretation of any formula of 1st order pred. log. into a p. irr. universe
- “singleton elimination” for conjunction: projections are definable
- similar to HoTT

Proof-Irrelevance at work

```
A : Type     n : nat     h : A
l, t, l_rev_rev : list A
l_case : l = h :: t
IHt : l = t
      → ∀ (n : nat) (e : n = length t),
        exist (λ l0 : list A, n = length l0) t e =
        rev' A n (rev' A n (exist (λ l0 : list A, n = length l0) t e))
eq_l_rev_rev : l = l_rev_rev
len_l : {‖n = length l‖}
len_rev_rev : {‖n = length l‖}
============================
 exist (λ l' : list A, n = length l') l len_l =
 exist (λ l' : list A, n = length l') l len_rev_rev

reflexivity. Qed.
```

Our Contribution

- designed a Calculus of Inductive Constructions with definitional proof-irrelevance expanding on ideas from *The rooster and the syntactic bracket* (Spiwack, Herbelin '13)
- we expect to be able to interpret almost all of current Coq
- related work: Pfenning, Awodey-Bauer *bracket-types*; NuPrl *squash-types*; Homotopy Type Theory (-1)-truncations; Abel's Agda

Benefits

- no duplication of *type formers* in both Prop and Type (c.f. Coq)
- Prop is generated by False, True, truncations and products
- no more need to look at (reduce) proofs during computations of programs
  → to duplicate proofs as programs: `Defined.` vs `Qed.` (c.f. Program)
- maintain compatibility with consistent axioms
  - axioms defined in Prop do not interfere with computation
  - axiomatic equalities reduce only when the "endpoints" are convertible

Future Work

- design an algorithm for definitional equality
  - based on Abel's work on normalisation-by-evaluation for pCIC, other forms of irrelevance
  - modify Π-impredicativity?
- prove the correctness of such an algorithm and the meta-theory of the calculus
- build an elaboration procedure from the pCIC to πCIC

Future Work

- design an algorithm for definitional equality
  - based on Abel's work on normalisation-by-evaluation for pCIC, other forms of irrelevance
  - modify Π-impredicativity?
- prove the correctness of such an algorithm and the meta-theory of the calculus
- build an elaboration procedure from the pCIC to πCIC

Thank you! Questions?

More inference rules!

judgements

$$\Gamma\ \mathsf{ctx} \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u \equiv v : A \qquad \mathsf{sp}_x A$$

$$\Gamma \vdash \text{guarded } f\ x_1 \ldots x_n \Rightarrow t$$

Sorts

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathsf{Prop} : \mathsf{Type}_1}\ \textsc{Prop-Ax} \qquad \frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathsf{Type}_i : \mathsf{Type}_{i+1}}\ \textsc{Type-Ax}$$

$$\frac{\Gamma \vdash A : \mathsf{Type}_i}{\Gamma \vdash A : \mathsf{Type}_{i+1}}\ \textsc{Type-cumul}$$

contexts

$$\frac{}{\cdot\ \mathsf{ctx}}\ \textsc{ctx-emp} \qquad \frac{x_1 : A_1,\ \ldots,\ x_{n-1} : A_{n-1} \vdash A_n\ :\ \mathsf{Type}_i}{(x_1 : A_1,\ \ldots,\ x_{n-1} : A_{n-1},\ x_n : A_n)\ \mathsf{ctx}}\ \textsc{ctx-ext}$$

variables

$$\frac{(x_1 : A_1, \ldots, x_n : A_n)\ \mathsf{ctx} \qquad 1 \le i \le n}{x_1 : A_1, \ldots, x_n : A_n \vdash x_i\ :\ A_i}\ \mathsf{Vble}$$

False

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash \mathsf{False} : \mathsf{Prop}}\ \textsc{False-form} \qquad \frac{\Gamma \vdash C : s \qquad \Gamma \vdash t : \mathsf{False}}{\Gamma \vdash !_C\, t : C}\ \textsc{False-elim}$$

$$\frac{\Gamma \vdash u : \mathsf{False} \qquad \Gamma \vdash v : \mathsf{False}}{\Gamma \vdash u \equiv v : \mathsf{False}}\ \textsc{False-irrel}$$

True

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash \mathsf{True} : \mathsf{Prop}}\ \textsc{True-form} \qquad \frac{\Gamma \text{ ctx}}{\Gamma \vdash \mathrm{I} : \mathsf{True}}\ \textsc{True-intro}$$

$$\frac{\Gamma \vdash u : \mathsf{True}}{\Gamma \vdash u \equiv \mathrm{I} : \mathsf{True}}\ \textsc{True-uniq}$$

$$\frac{\Gamma \vdash A : \mathsf{Type}_i \qquad \Gamma \vdash B : \mathsf{Type}_j \qquad (\mathsf{Type}_i, \mathsf{Type}_j, s_3) \in \mathscr{R}}{\Gamma \vdash A + B : s_3} \; +\text{-FORM}$$

$$\frac{\Gamma \vdash A : \mathsf{Type}_i \qquad \Gamma \vdash B : \mathsf{Type}_j \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathsf{inl}\ a : A + B} \; +\text{-INTRO}_1$$

$$\frac{\begin{array}{c}\Gamma, z : A + B \vdash C : s \qquad \Gamma, x : A \vdash u : C[\mathsf{inl}\ x/z] \\ \Gamma, y : B \vdash v : C[\mathsf{inr}\ y/z] \qquad \Gamma \vdash t : A + B\end{array}}{\Gamma \vdash \mathsf{case}\, t\, \mathsf{as}\, z\, \mathsf{return}\, C\, \mathsf{of}\, \mathsf{inl}\ x \Rightarrow u \mid \mathsf{inr}\ y \Rightarrow v : C[t/z]} \; +\text{-ELIM}$$

$$\frac{\begin{array}{c}\Gamma, z : A + B \vdash C : s \qquad \Gamma, x : A \vdash u : C[\mathsf{inl}\ x/z] \\ \Gamma, y : B \vdash v : C[\mathsf{inr}\ y/z] \qquad \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathsf{case}\, \mathsf{inl}\ a\, \mathsf{as}\, z\, \mathsf{return}\, C\, \mathsf{of}\, \mathsf{inl}\ x \Rightarrow u \mid \mathsf{inr}\ y \Rightarrow v \equiv u[a/x] : C[\mathsf{inl}\ a/z]} \; +\text{-COMP}_1$$

$$\frac{\Gamma \vdash A : \mathsf{Type}_i \qquad \Gamma, x : A \vdash B : s_2 \qquad (\mathsf{Type}_i, s_2, s_3) \in \mathscr{R}_\Pi}{\Gamma \vdash \prod_{(x:A)} B : s_3}\ \Pi\text{-form}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.\, t : \prod_{(x:A)} B}\ \Pi\text{-intro}$$

$$\frac{\Gamma \vdash u : \prod_{(x:A)} B \qquad \Gamma \vdash v : A}{\Gamma \vdash u\ v : B[v/x]}\ \Pi\text{-elim}$$

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash v : A}{\Gamma \vdash (\lambda x : A.\, t)\ v \equiv t[v/x] : B[v/x]}\ \Pi\text{-comp}$$

$$\frac{\Gamma \vdash t : \prod_{(x:A)} B}{\Gamma \vdash t \equiv (\lambda x : A.\, t\ x) : \prod_{(x:A)} B}\ \Pi\text{-uniq}$$

$\mathscr{R} = \{ (s_1, s_2, \max(s_1, s_2)) \}$ $\mathscr{R}_{impred} = \{ (s, \mathsf{Prop}, \mathsf{Prop}) \}$ $\mathscr{R}_\Pi = \mathscr{R}_{impred} \cup \mathscr{R}$

$\max \mathsf{Type}_i\ \mathsf{Type}_j = \mathsf{Type}_{\max i\, j}$ $\max s\ \mathsf{Prop} = \max \mathsf{Prop}\ s = s$

$$\frac{\Gamma \vdash A : \mathsf{Type}_i \qquad \Gamma, x : A \vdash B : s_2 \qquad (\mathsf{Type}_i, s_2, s_3) \in \mathscr{R}}{\Gamma \vdash \sum_{(x:A)} B : s_3}\ \Sigma\text{-FORM}$$

$$\frac{\Gamma, x : A \vdash B : s \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : B[a/x]}{\Gamma \vdash (a, b) : \sum_{(x:A)} B}\ \Sigma\text{-INTRO}$$

$$\frac{\Gamma \vdash t : \sum_{(x:A)} B}{\Gamma \vdash \mathsf{pr}_1\, t : A}\ \Sigma\text{-ELIM-1} \qquad \frac{\Gamma \vdash t : \sum_{(x:A)} B}{\Gamma \vdash \mathsf{pr}_2\, t : B[\mathsf{pr}_1\, t/x]}\ \Sigma\text{-ELIM-2}$$

$$\frac{\Gamma \vdash (a, b) : \sum_{(x:A)} B}{\Gamma \vdash \mathsf{pr}_1(a, b) \equiv a : A}\ \Sigma\text{-COMP-1} \qquad \frac{\Gamma \vdash (a, b) : \sum_{(x:A)} B}{\Gamma \vdash \mathsf{pr}_2(a, b) \equiv b : B[a/x]}\ \Sigma\text{-COMP-2}$$

$$\frac{\Gamma \vdash t : \sum_{(x:A)} B}{\Gamma \vdash (\mathsf{pr}_1\, t, \mathsf{pr}_2\, t) \equiv t : \sum_{(x:A)} B}\ \Sigma\text{-UNIQ}$$

$$\frac{\Gamma \vdash A : \mathsf{Type}_i \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : A}{\Gamma \vdash a =_A b : \mathsf{Type}_i} =\text{-FORM}$$

$$\frac{\Gamma \vdash A : \mathsf{Type}_i \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathsf{refl}_A a : a =_A a} =\text{-INTRO}$$

$$\frac{\begin{array}{c}\Gamma, x{:}A, y{:}A, p{:}x =_A y \vdash C : \mathsf{Type}_i \\ \Gamma, z{:}A \vdash c : C[z/x, z/y, \mathsf{refl}_A z/p] \\ \Gamma \vdash a : A \qquad \Gamma \vdash b : A \qquad \Gamma \vdash e : a =_A b\end{array}}{\Gamma \vdash \mathsf{ind}_{=_A}\ x.y.p.C\ z.c\ a\ b\ e : C[a/x, b/y, e/p]} =\text{-ELIM}$$

$$\frac{\begin{array}{c}\Gamma, x{:}A, y{:}A, p{:}x =_A y \vdash C : \mathsf{Type}_i \\ \Gamma, z{:}A \vdash c : C[z/x, z/y, \mathsf{refl}_A z/p] \qquad \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathsf{ind}_{=_A}\ x.y.p.C\ z.c\ a\ a\ \mathsf{refl}_A a \equiv c[a/z] : C[a/x, a/y, \mathsf{refl}_A a/p]} =\text{-COMP}$$

$$\frac{\Gamma \vdash A : s \qquad \Gamma,\, X : A \to \mathsf{Type}_i \vdash F : A \to \mathsf{Type}_i \qquad \mathsf{sp}_X F}{\Gamma \vdash \mu\, X : A \to \mathsf{Type}_i .\, F : A \to \mathsf{Type}_i}\ \mu\text{-FORM}$$

$$\frac{\Gamma \vdash \mu\, X : A \to \mathsf{Type}_i .\, F : A \to \mathsf{Type}_i}{\Gamma \vdash \mu\, X : A \to \mathsf{Type}_i .\, F \equiv F[\mu\, X : A \to \mathsf{Type}_i .\, F \,/\, X] : A \to \mathsf{Type}_i}\ \mu\text{-COMP}$$

$$\frac{\begin{array}{c}\Gamma \vdash \prod_{(x_1 : A_1)} \cdots \prod_{(x_n : A_n)} B : s \\ \Gamma,\ f : \prod_{(x_1 : A_1)} \cdots \prod_{(x_n : A_n)} B,\ x_1 : A_1,\ \ldots,\ x_n : A_n \vdash t : B \\ \Gamma \vdash \text{guarded } f\ x_1 \ldots x_n \Rightarrow t\end{array}}{\Gamma \vdash \text{fix } f\ (x_1 : A_1) \ldots (x_n : A_n) \Rightarrow t : \prod_{(x_1 : A_1)} \cdots \prod_{(x_n : A_n)} B}\ \textsc{fix-intro}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : s \qquad \Gamma \vdash A \leq B : s}{\Gamma \vdash t : B} \leq\text{-CONV}$$

$$\frac{\Gamma \vdash A_1 : s \qquad \Gamma \vdash A_1 \equiv A_2 : s \qquad \Gamma, x : A_1 \vdash B_1 \leq B_2 : s'}{\Gamma \vdash \prod_{(x:A_1)} B_1 \leq \prod_{(x:A_2)} B_2 : s'} \leq\text{-}\Pi$$

$$\frac{i \leq j}{\Gamma \vdash \mathsf{Type}_i \leq \mathsf{Type}_j : s} \leq\text{-Type}$$

$$\frac{\Gamma \vdash A \leq B : s \qquad \Gamma \vdash B \leq C : s}{\Gamma \vdash A \leq C : s} \leq\text{-TRANS} \qquad \frac{\Gamma \vdash A \equiv B : s}{\Gamma \vdash A \leq B : s} \equiv\text{-}\leq$$

$$\frac{\Gamma \vdash u \equiv v : A \qquad \Gamma \vdash B : s \qquad \Gamma \vdash A \leq B : s}{\Gamma \vdash u \equiv v : B} \equiv\text{-COMPAT}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t \equiv t : A} \equiv\text{-REFL} \qquad \frac{\Gamma \vdash u \equiv v : A}{\Gamma \vdash v \equiv u : A} \equiv\text{-SYM}$$

$$
\begin{aligned}
nat &:\equiv \mu\, X : \mathsf{Type}_0.\, \mathbf{1} + X \\
\mathrm{O} &:\equiv \mathsf{inl}\ \star\ :\ \mathbf{1} + nat \\
\mathrm{S} &:\equiv \lambda(n : nat).\ \mathsf{inr}\ n\ :\ \mathbf{1} + nat
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{nat\_rect} :\equiv\ & \lambda(P : nat \to \mathsf{Type})\ (f_0 : P\ 0)\ (f_S : \textstyle\prod_{(n:nat)} P\ n \to P\ (\mathrm{S}\ n)). \\
& \mathrm{fix}\ F\ (n : nat) \Rightarrow \\
& \mathrm{case}\ n\ \mathrm{as}\ x\ \mathrm{return}\ P\ x\ \mathrm{of} \\
& \quad \mid \mathsf{inl}\ \_ \Rightarrow f_0 \\
& \quad \mid \mathsf{inr}\ m \Rightarrow f_S\ m\ (F\ m) \\
\mathrm{nat\_ind} :\equiv\ & \lambda(P : nat \to \mathsf{Prop})\ (p_0 : \{P\ 0\})\ (p_S : \textstyle\prod_{(m:nat)} \{P\ m\} \to \{P\ (\mathrm{S}\ m)\}). \\
& \mathrm{nat\_rect}\ (\lambda n : nat.\ \{P n\})\ p_0\ p_S
\end{aligned}
$$