

Type Theories with Computation Rules for the Univalence Axiom

Robin Adams

January 21, 2016

```
module main where
```

1 Preliminaries

```
module Prelims where
```

```
postulate Level : Set
postulate zro : Level
postulate suc : Level → Level
{-# BUILTIN LEVEL Level #-}
{-# BUILTIN LEVELZERO zro #-}
{-# BUILTIN LEVELSUC suc #-}
```

1.1 Functions

We write id_A for the identity function on the type A , and $g \circ f$ for the composition of functions g and f .

```
id : ∀ (A : Set) → A → A
id A x = x

infix 75 _o_
_o_ : ∀ {A B C : Set} → (B → C) → (A → B) → A → C
(g ∘ f) x = g (f x)
```

1.2 Equality

We use the inductively defined equality $=$ on every datatype.

```
infix 50 _≡_
data _≡_ {A : Set} (a : A) : A → Set where
  ref : a ≡ a
```

```
subst : ∀ {i} {A : Set} (P : A → Set i) {a} {b} → a ≡ b → P a → P b
subst P ref Pa = Pa
```

```
subst2 : ∀ {A B : Set} (P : A → B → Set) {a a' b b'} → a ≡ a' → b ≡ b' → P a b → P a' b'
subst2 P ref ref Pab = Pab
```

```
sym : ∀ {A : Set} {a b : A} → a ≡ b → b ≡ a
sym ref = ref
```

```
trans : ∀ {A : Set} {a b c : A} → a ≡ b → b ≡ c → a ≡ c
trans ref ref = ref
```

```
wd : ∀ {A B : Set} (f : A → B) {a a' : A} → a ≡ a' → f a ≡ f a'
wd _ ref = ref
```

```
wd2 : ∀ {A B C : Set} (f : A → B → C) {a a' : A} {b b' : B} → a ≡ a' → b ≡ b' → f a b ≡ f a' b'
wd2 _ ref ref = ref
```

```
module Equational-Reasoning (A : Set) where
```

```
  infix 2 '·'_
  '·'_ : ∀ (a : A) → a ≡ a
  '·'_ = ref
```

```
  infix 1 '≡'_
  '≡'_ : ∀ {a b : A} → a ≡ b → ∀ c → b ≡ c → a ≡ c
  δ ≡ c [ δ' ] = trans δ δ'
```

```
  infix 1 '≡'_
  '≡'_ : ∀ {a b : A} → a ≡ b → ∀ c → c ≡ b → a ≡ c
  δ ≡ c [[ δ' ]] = trans δ (sym δ')
```

We also write $f \sim g$ iff the functions f and g are extensionally equal, that is, $f(x) = g(x)$ for all x .

```
infix 50 '∼'_
'∼'_ : ∀ {A B : Set} → (A → B) → (A → B) → Set
f ∼ g = ∀ x → f x ≡ g x
```

2 Datatypes

We introduce a universe **FinSet** of (names of) finite sets. There is an empty set $\emptyset : \mathbf{FinSet}$, and for every $A : \mathbf{FinSet}$, the type $A + 1 : \mathbf{FinSet}$ has one more element:

$$A + 1 = \{\perp\} \uplus \{\uparrow a : a \in A\}$$

```
data FinSet : Set where
  ∅ : FinSet
```

`Lift : FinSet → FinSet`

`data El : FinSet → Set where`
`⊥ : ∀ {V} → El (Lift V)`
`↑ : ∀ {V} → El V → El (Lift V)`

A *replacement* from U to V is simply a function $U \rightarrow V$.

`Rep : FinSet → FinSet → Set`
`Rep U V = El U → El V`

Given $f : A \rightarrow B$, define $f + 1 : A + 1 \rightarrow B + 1$ by

$$(f + 1)(\perp) = \perp$$

$$(f + 1)(\uparrow x) = \uparrow f(x)$$

`lift : ∀ {U} {V} → Rep U V → Rep (Lift U) (Lift V)`
`lift _ ⊥ = ⊥`
`lift f (↑ x) = ↑ (f x)`

`liftwd : ∀ {U} {V} {f g : Rep U V} → f ~ g → lift f ~ lift g`
`liftwd f-is-g ⊥ = ref`
`liftwd f-is-g (↑ x) = wd ↑ (f-is-g x)`

This makes $(-)+1$ into a functor $\mathbf{FinSet} \rightarrow \mathbf{FinSet}$; that is,

$$\text{id}_V + 1 = \text{id}_{V+1}$$

$$(g \circ f) + 1 = (g + 1) \circ (f + 1)$$

`liftid : ∀ {V} → lift (id (El V)) ~ id (El (Lift V))`
`liftid ⊥ = ref`
`liftid (↑ _) = ref`

`liftcomp : ∀ {U} {V} {W} {g : Rep V W} {f : Rep U V} → lift (g ∘ f) ~ lift g ∘ lift f`
`liftcomp ⊥ = ref`
`liftcomp (↑ _) = ref`

`data List (A : Set) : Set where`
`⟨⟩ : List A`
`_::_ : List A → A → List A`

`open import Prelims`

`module PL where`
`open import Prelims`

3 Propositional Logic

Fix sets of *proof variables* and *term variables*.

The syntax of the system is given by the following grammar.

Proof	δ	$::=$	$p \mid \delta\delta \mid \lambda p : \phi.\delta$
Proposition	ϕ	$::=$	$\perp \mid \phi \rightarrow \phi$
Proof Context	Δ	$::=$	$\langle \rangle \mid \Delta, p : \phi$
Judgement	\mathcal{J}	$::=$	$\Delta \vdash \delta : \phi$

where p ranges over proof variables and x ranges over term variables. The variable p is bound within δ in the proof $\lambda p : \phi.\delta$, and the variable x is bound within M in the term $\lambda x : A.M$. We identify proofs and terms up to α -conversion.

We write **Proof** (P) for the set of all proofs δ with $\text{FV}(\delta) \subseteq V$.

```

infix 75 _⇒_
data Prp : Set where
  ⊥ : Prp
  _⇒_ : Prp → Prp → Prp

infix 80 _,-_
data PContext : FinSet → Set where
  ⟨⟩ : PContext ∅
  _,-_ : ∀ {P} → PContext P → Prp → PContext (Lift P)

propof : ∀ {P} → El P → PContext P → Prp
propof ⊥ ( _ , φ ) = φ
propof (↑ p) (Γ , _) = propof p Γ

data Proof : FinSet → Set where
  var : ∀ {P} → El P → Proof P
  app : ∀ {P} → Proof P → Proof P → Proof P
  Λ : ∀ {P} → Prp → Proof (Lift P) → Proof P

```

Let $P, Q : \mathbf{FinSet}$. A *replacement* from P to Q is just a function $P \rightarrow Q$. Given a term $M : \mathbf{Proof}(P)$ and a replacement $\rho : P \rightarrow Q$, we write $M\{\rho\} : \mathbf{Proof}(Q)$ for the result of replacing each variable x in M with $\rho(x)$.

```

infix 60 _<_>
_<_> : ∀ {P Q} → Proof P → Rep P Q → Proof Q
var p < ρ > = var (ρ p)
app δ ε < ρ > = app (δ < ρ >) (ε < ρ >)
Λ φ δ < ρ > = Λ φ (δ < lift ρ >)

```

With this as the action on arrows, **Proof** (ρ) becomes a functor $\mathbf{FinSet} \rightarrow \mathbf{Set}$.

```

repwd : ∀ {P Q : FinSet} {ρ ρ' : El P → El Q} → ρ ~ ρ' → ∀ δ → δ < ρ > ≡ δ < ρ' >
repwd ρ-is-ρ' (var p) = wd var (ρ-is-ρ' p)
repwd ρ-is-ρ' (app δ ε) = wd2 app (repwd ρ-is-ρ' δ) (repwd ρ-is-ρ' ε)
repwd ρ-is-ρ' (Λ φ δ) = wd (Λ φ) (repwd (liftwd ρ-is-ρ') δ)

repid : ∀ {Q : FinSet} δ → δ < id (El Q) > ≡ δ
repid (var _) = ref
repid (app δ ε) = wd2 app (repid δ) (repid ε)
repid {Q} (Λ φ δ) = wd (Λ φ) (let open Equational-Reasoning (Proof (Lift Q)) in
  ∴ δ < lift (id (El Q)) >
  ≡ δ < id (El (Lift Q)) > [ repwd liftid δ ]
  ≡ δ [ repid δ ])

repcomp : ∀ {P Q R : FinSet} (ρ : El Q → El R) (σ : El P → El Q) M → M < ρ ∘ σ > ≡ M
repcomp ρ σ (var _) = ref
repcomp ρ σ (app δ ε) = wd2 app (repcomp ρ σ δ) (repcomp ρ σ ε)
repcomp {R = R} ρ σ (Λ φ δ) = wd (Λ φ) (let open Equational-Reasoning (Proof (Lift R)) in
  ∴ δ < lift (ρ ∘ σ) >
  ≡ δ < lift ρ ∘ lift σ > [ repwd liftcomp δ ]
  ≡ (δ < lift σ >) < lift ρ > [ repcomp _ _ δ ])

```

A *substitution* σ from P to Q , $\sigma : P \Rightarrow Q$, is a function $\sigma : P \rightarrow \mathbf{Proof}(Q)$.

```

Sub : FinSet → FinSet → Set
Sub P Q = El P → Proof Q

```

The identity substitution $\text{id}_Q : Q \Rightarrow Q$ is defined as follows.

```

idSub : ∀ Q → Sub Q Q
idSub _ = var

```

Given $\sigma : P \Rightarrow Q$ and $M : \mathbf{Proof}(P)$, we want to define $M[\sigma] : \mathbf{Proof}(Q)$, the result of applying the substitution σ to M . Only after this will we be able to define the composition of two substitutions. However, there is some work we need to do before we are able to do this.

We can define the composition of a substitution and a replacement as follows.

```

infix 75 _•_1_
_•_1_ : ∀ {P} {Q} {R} → Rep Q R → Sub P Q → Sub P R
(ρ •_1 σ) u = σ u < ρ >

```

(On the other side, given $\rho : P \rightarrow Q$ and $\sigma : Q \Rightarrow R$, the composition is just function composition $\sigma \circ \rho : P \Rightarrow R$.)

Given a substitution $\sigma : P \Rightarrow Q$, define the substitution $\sigma + 1 : P + 1 \Rightarrow Q + 1$ as follows.

```

liftSub : ∀ {P} {Q} → Sub P Q → Sub (Lift P) (Lift Q)
liftSub _ ⊥ = var ⊥

```

$\text{liftSub } \sigma \ (\uparrow x) = \sigma \ x < \uparrow >$

$\text{liftSub-wd} : \forall \{P \ Q\} \{\sigma \ \sigma' : \text{Sub } P \ Q\} \rightarrow \sigma \sim \sigma' \rightarrow \text{liftSub } \sigma \sim \text{liftSub } \sigma'$
 $\text{liftSub-wd } \sigma\text{-is-}\sigma' \ \perp = \text{ref}$
 $\text{liftSub-wd } \sigma\text{-is-}\sigma' \ (\uparrow x) = \text{wd } (\lambda x \rightarrow x < \uparrow >) (\sigma\text{-is-}\sigma' \ x)$

Lemma 1. *The operations \bullet and $(-)+1$ satisfied the following properties.*

1. $\text{id}_Q + 1 = \text{id}_{Q+1}$
2. For $\rho : Q \rightarrow R$ and $\sigma : P \Rightarrow Q$, we have $(\rho \bullet \sigma) + 1 = (\rho + 1) \bullet (\sigma + 1)$.
3. For $\sigma : Q \Rightarrow R$ and $\rho : P \rightarrow Q$, we have $(\sigma \circ \rho) + 1 = (\sigma + 1) \circ (\rho + 1)$.

$\text{liftSub-id} : \forall \{Q : \text{FinSet}\} \rightarrow \text{liftSub } (\text{idSub } Q) \sim \text{idSub } (\text{Lift } Q)$
 $\text{liftSub-id } \perp = \text{ref}$
 $\text{liftSub-id } (\uparrow x) = \text{ref}$

$\text{liftSub-comp}_1 : \forall \{P \ Q \ R : \text{FinSet}\} (\sigma : \text{Sub } P \ Q) (\rho : \text{Rep } Q \ R) \rightarrow$
 $\text{liftSub } (\rho \bullet_1 \sigma) \sim \text{lift } \rho \bullet_1 \text{liftSub } \sigma$
 $\text{liftSub-comp}_1 \ \sigma \ \rho \ \perp = \text{ref}$
 $\text{liftSub-comp}_1 \ \{R = R\} \ \sigma \ \rho \ (\uparrow x) = \text{let open Equational-Reasoning (Proof (Lift R)) in}$
 $\quad \because \sigma \ x < \rho > < \uparrow >$
 $\quad \equiv \sigma \ x < \uparrow \circ \rho > \quad [[\text{repcomp } \uparrow \rho \ (\sigma \ x)]]$
 $\quad \equiv \sigma \ x < \uparrow > < \text{lift } \rho > [\text{repcomp } (\text{lift } \rho) \ \uparrow \ (\sigma \ x)]$
 $--\text{because } \text{lift } \rho \ (\uparrow x) = \uparrow (\rho \ x)$

$\text{liftSub-comp}_2 : \forall \{P \ Q \ R : \text{FinSet}\} (\sigma : \text{Sub } Q \ R) (\rho : \text{Rep } P \ Q) \rightarrow$
 $\text{liftSub } (\sigma \circ \rho) \sim \text{liftSub } \sigma \circ \text{lift } \rho$
 $\text{liftSub-comp}_2 \ \sigma \ \rho \ \perp = \text{ref}$
 $\text{liftSub-comp}_2 \ \sigma \ \rho \ (\uparrow x) = \text{ref}$

Now define $M[\sigma]$ as follows.

$\text{infix } 60 \ _[\![\]\!]$
 $_[\![\]\!] : \forall \{P \ Q : \text{FinSet}\} \rightarrow \text{Proof } P \rightarrow \text{Sub } P \ Q \rightarrow \text{Proof } Q$
 $(\text{var } x) \quad [\![\sigma]\!] = \sigma \ x$
 $(\text{app } \delta \ \epsilon) \quad [\![\sigma]\!] = \text{app } (\delta \ [\![\sigma]\!]) \ (\epsilon \ [\![\sigma]\!])$
 $(\Lambda \ A \ \delta) \quad [\![\sigma]\!] = \Lambda \ A \ (\delta \ [\![\text{liftSub } \sigma]\!])$

$\text{subwd} : \forall \{P \ Q : \text{FinSet}\} \{\sigma \ \sigma' : \text{Sub } P \ Q\} \rightarrow \sigma \sim \sigma' \rightarrow \forall \delta \rightarrow \delta \ [\![\sigma]\!] \equiv \delta \ [\![\sigma']\!]$
 $\text{subwd } \sigma\text{-is-}\sigma' \ (\text{var } x) = \sigma\text{-is-}\sigma' \ x$
 $\text{subwd } \sigma\text{-is-}\sigma' \ (\text{app } \delta \ \epsilon) = \text{wd2 app (subwd } \sigma\text{-is-}\sigma' \ \delta) (\text{subwd } \sigma\text{-is-}\sigma' \ \epsilon)$
 $\text{subwd } \sigma\text{-is-}\sigma' \ (\Lambda \ A \ \delta) = \text{wd } (\Lambda \ A) (\text{subwd } (\text{liftSub-wd } \sigma\text{-is-}\sigma') \ \delta)$

This interacts with our previous operations in a good way:

Lemma 2.

1. $M[\text{id}_Q] \equiv M$
2. $M[\rho \bullet \sigma] \equiv \delta[\sigma]\{\rho\}$
3. $M[\sigma \circ \rho] \equiv \delta < \rho > [\sigma]$

```

subid : ∀ {Q : FinSet} (δ : Proof Q) → δ [ idSub Q ] ≡ δ
subid (var x) = ref
subid (app δ ε) = wd2 app (subid δ) (subid ε)
subid {Q} (Λ φ δ) = let open Equational-Reasoning (Proof Q) in
  ∴ Λ φ (δ [ liftSub (idSub Q) ])
  ≡ Λ φ (δ [ idSub (Lift Q) ]) [ wd (Λ φ) (subwd liftSub-id δ) ]
  ≡ Λ φ δ [ wd (Λ φ) (subid δ) ]

```

```

rep-sub : ∀ {P} {Q} {R} (σ : Sub P Q) (ρ : Rep Q R) (δ : Proof P) → δ [ σ ] < ρ > ≡ δ [
rep-sub σ ρ (var x) = ref
rep-sub σ ρ (app δ ε) = wd2 app (rep-sub σ ρ δ) (rep-sub σ ρ ε)
rep-sub {R = R} σ ρ (Λ φ δ) = let open Equational-Reasoning (Proof R) in
  ∴ Λ φ ((δ [ liftSub σ ]) < lift ρ >)
  ≡ Λ φ (δ [ lift ρ •1 liftSub σ ]) [ wd (Λ φ) (rep-sub (liftSub σ) (lift ρ) δ) ]
  ≡ Λ φ (δ [ liftSub (ρ •1 σ) ]) [[ wd (Λ φ) (subwd (liftSub-comp1 σ ρ) δ) ]]

```

```

sub-rep : ∀ {P} {Q} {R} (σ : Sub Q R) (ρ : Rep P Q) δ → δ < ρ > [ σ ] ≡ δ [ σ ∘ ρ ]
sub-rep σ ρ (var x) = ref
sub-rep σ ρ (app δ ε) = wd2 app (sub-rep σ ρ δ) (sub-rep σ ρ ε)
sub-rep {R = R} σ ρ (Λ φ δ) = let open Equational-Reasoning (Proof R) in
  ∴ Λ φ ((δ < lift ρ >) [ liftSub σ ])
  ≡ Λ φ (δ [ liftSub σ ∘ lift ρ ]) [ wd (Λ φ) (sub-rep (liftSub σ) (lift ρ) δ) ]
  ≡ Λ φ (δ [ liftSub (σ ∘ ρ) ]) [[ wd (Λ φ) (subwd (liftSub-comp2 σ ρ) δ) ]]

```

We define the composition of two substitutions, as follows.

```

infix 75 _•_
_•_ : ∀ {P Q R : FinSet} → Sub Q R → Sub P Q → Sub P R
(σ • ρ) x = ρ x [ σ ]

```

Lemma 3. *Let $\sigma : Q \Rightarrow R$ and $\rho : P \Rightarrow Q$.*

1. $(\sigma \bullet \rho) + 1 = (\sigma + 1) \bullet (\rho + 1)$
2. $M[\sigma \bullet \rho] \equiv \delta[\rho][\sigma]$

```

liftSub-comp : ∀ {P} {Q} {R} (σ : Sub Q R) (ρ : Sub P Q) →
  liftSub (σ • ρ) ~ liftSub σ • liftSub ρ
liftSub-comp σ ρ ⊥ = ref
liftSub-comp σ ρ (↑ x) = trans (rep-sub σ ↑ (ρ x)) (sym (sub-rep (liftSub σ) ↑ (ρ x)))

```

```

subcomp : ∀ {P} {Q} {R} (σ : Sub Q R) (ρ : Sub P Q) δ → δ [ σ • ρ ] ≡ δ [ ρ ] [ σ ]

```

```

subcomp  $\sigma$   $\rho$  (var x) = ref
subcomp  $\sigma$   $\rho$  (app  $\delta$   $\epsilon$ ) = wd2 app (subcomp  $\sigma$   $\rho$   $\delta$ ) (subcomp  $\sigma$   $\rho$   $\epsilon$ )
subcomp  $\sigma$   $\rho$  ( $\Lambda$   $\phi$   $\delta$ ) = wd ( $\Lambda$   $\phi$ ) (trans (subwd (liftSub-comp  $\sigma$   $\rho$ )  $\delta$ ) (subcomp (liftSub  $\sigma$   $\rho$ )  $\delta$ ))

```

Lemma 4. *The finite sets and substitutions form a category under this composition.*

```

assoc :  $\forall$  {P Q R S} { $\rho$  : Sub R S} { $\sigma$  : Sub Q R} { $\tau$  : Sub P Q}  $\rightarrow$ 
   $\rho \bullet (\sigma \bullet \tau) \sim (\rho \bullet \sigma) \bullet \tau$ 
assoc {P} {Q} {R} {X} { $\rho$ } { $\sigma$ } { $\tau$ } x = sym (subcomp  $\rho$   $\sigma$  ( $\tau$  x))

subunitl :  $\forall$  {P} {Q} { $\sigma$  : Sub P Q}  $\rightarrow$  idSub Q  $\bullet$   $\sigma \sim \sigma$ 
subunitl {P} {Q} { $\sigma$ } x = subid ( $\sigma$  x)

subunitr :  $\forall$  {P} {Q} { $\sigma$  : Sub P Q}  $\rightarrow$   $\sigma \bullet$  idSub P  $\sim \sigma$ 
subunitr _ = ref

```

Replacement is a special case of substitution, in the following sense:

Lemma 5. *For any replacement ρ ,*

$$\delta\{\rho\} \equiv \delta[\rho]$$

```

rep-is-sub :  $\forall$  {P} {Q} { $\rho$  : El P  $\rightarrow$  El Q}  $\delta \rightarrow \delta < \rho > \equiv \delta$  [ $\text{var} \circ \rho$ ]
rep-is-sub (var x) = ref
rep-is-sub (app  $\delta$   $\epsilon$ ) = wd2 app (rep-is-sub  $\delta$ ) (rep-is-sub  $\epsilon$ )
rep-is-sub {Q = Q} { $\rho$ } ( $\Lambda$   $\phi$   $\delta$ ) = let open Equational-Reasoning (Proof Q) in
   $\therefore \Lambda$   $\phi$  ( $\delta < \text{lift } \rho >$ )
   $\equiv \Lambda$   $\phi$  ( $\delta$  [ $\text{var} \circ \text{lift } \rho$ ]) [wd ( $\Lambda$   $\phi$ ) (rep-is-sub  $\delta$ )]
   $\equiv \Lambda$   $\phi$  ( $\delta$  [ $\text{liftSub var} \circ \text{lift } \rho$ ]) [[wd ( $\Lambda$   $\phi$ ) (subwd ( $\lambda x \rightarrow \text{liftSub-id} (\text{lift } \rho$  x))  $\delta$ )]
   $\equiv \Lambda$   $\phi$  ( $\delta$  [ $\text{liftSub} (\text{var} \circ \rho)$ ]) [[wd ( $\Lambda$   $\phi$ ) (subwd (liftSub-comp2 var  $\rho$ )  $\delta$ )]

```

Given $\delta : \mathbf{Proof}(P)$, let $[\perp := \delta] : P + 1 \Rightarrow P$ be the substitution that maps \perp to δ , and $\uparrow x$ to x for $x \in P$. We write $\delta[\epsilon]$ for $\delta[\perp := \epsilon]$.

```

botsub :  $\forall$  {Q}  $\rightarrow$  Proof Q  $\rightarrow$  Sub (Lift Q) Q
botsub  $\delta$   $\perp$  =  $\delta$ 
botsub _ ( $\uparrow$  x) = var x

```

```

subbot :  $\forall$  {P}  $\rightarrow$  Proof (Lift P)  $\rightarrow$  Proof P  $\rightarrow$  Proof P
subbot  $\delta$   $\epsilon$  =  $\delta$  [ $\text{botsub } \epsilon$ ]

```

Lemma 6. *Let $\delta : \mathbf{Proof}(P)$ and $\sigma : P \Rightarrow Q$. Then*

$$\sigma \bullet [\perp := \delta] \sim [\perp := \delta[\sigma]] \circ (\sigma + 1)$$

```

sub-botsub :  $\forall$  {P} {Q} ( $\sigma$  : Sub P Q) ( $\delta$  : Proof P)  $\rightarrow$ 
   $\sigma \bullet \text{botsub } \delta \sim \text{botsub } (\delta$  [ $\sigma$ ])  $\bullet$  liftSub  $\sigma$ 

```



```

sub-botsub  $\sigma$   $\delta$   $\perp$  = ref
sub-botsub  $\sigma$   $\delta$  ( $\uparrow$   $x$ ) = let open Equational-Reasoning (Proof  $\_$ ) in
 $\because$   $\sigma$   $x$ 
 $\equiv \sigma$   $x$   $\llbracket$  idSub  $\_$   $\rrbracket$  [[ subid ( $\sigma$   $x$ ) ]]
 $\equiv \sigma$   $x$   $< \uparrow > \llbracket$  botsub ( $\delta$   $\llbracket$   $\sigma$   $\rrbracket$ )  $\rrbracket$  [[ sub-rep (botsub ( $\delta$   $\llbracket$   $\sigma$   $\rrbracket$ ))  $\uparrow$  ( $\sigma$   $x$ ) ]]

```

We write $\delta \twoheadrightarrow \epsilon$ iff δ β -reduces to ϵ in zero or more steps, $\delta \twoheadrightarrow^+ \epsilon$ iff δ β -reduces to ϵ in one or more steps, and $\delta \simeq \epsilon$ iff the terms δ and ϵ are β -convertible.

Given substitutions ρ and σ , we write $\rho \twoheadrightarrow \sigma$ iff $\rho(x) \twoheadrightarrow \sigma(x)$ for all x , and $\rho \simeq \sigma$ iff $\rho(x) \simeq \sigma(x)$ for all x .

```

data  $\_ \rightarrow_1 \_$  :  $\forall \{P\} \rightarrow$  Proof  $P \rightarrow$  Proof  $P \rightarrow$  Set where
 $\beta$  :  $\forall \{P\} \{ \phi \} \{ \delta \} \{ \epsilon : \text{Proof } P \} \rightarrow$  app ( $\Lambda$   $\phi$   $\delta$ )  $\epsilon \rightarrow_1$  subbot  $\delta$   $\epsilon$ 
 $\xi$  :  $\forall \{P\} \{ \phi \} \{ \delta \} \{ \epsilon : \text{Proof } (\text{Lift } P) \} \rightarrow$   $\delta \rightarrow_1 \epsilon \rightarrow \Lambda$   $\phi$   $\delta \rightarrow_1 \Lambda$   $\phi$   $\epsilon$ 
appl :  $\forall \{P\} \{ \delta \} \{ \delta' \} \{ \epsilon : \text{Proof } P \} \rightarrow$   $\delta \rightarrow_1 \delta' \rightarrow$  app  $\delta$   $\epsilon \rightarrow_1$  app  $\delta'$   $\epsilon$ 
appr :  $\forall \{P\} \{ \delta \in \epsilon' : \text{Proof } P \} \rightarrow$   $\epsilon \rightarrow_1 \epsilon' \rightarrow$  app  $\delta$   $\epsilon \rightarrow_1$  app  $\delta$   $\epsilon'$ 

data  $\_ \twoheadrightarrow \_$  :  $\forall \{Q\} \rightarrow$  Proof  $Q \rightarrow$  Proof  $Q \rightarrow$  Set where
 $\beta$  :  $\forall \{Q\} \phi (\delta : \text{Proof } (\text{Lift } Q)) \epsilon \rightarrow$  app ( $\Lambda$   $\phi$   $\delta$ )  $\epsilon \twoheadrightarrow$  subbot  $\delta$   $\epsilon$ 
ref :  $\forall \{P\} \{ \delta : \text{Proof } P \} \rightarrow$   $\delta \twoheadrightarrow \delta$ 
 $\twoheadrightarrow$ trans :  $\forall \{Q\} \{ \gamma \delta \epsilon : \text{Proof } Q \} \rightarrow$   $\gamma \twoheadrightarrow \delta \rightarrow \delta \twoheadrightarrow \epsilon \rightarrow \gamma \twoheadrightarrow \epsilon$ 
app :  $\forall \{Q\} \{ \delta \delta' \epsilon \epsilon' : \text{Proof } Q \} \rightarrow$   $\delta \twoheadrightarrow \delta' \rightarrow \epsilon \twoheadrightarrow \epsilon' \rightarrow$  app  $\delta$   $\epsilon \twoheadrightarrow$  app  $\delta'$   $\epsilon'$ 
 $\xi$  :  $\forall \{Q\} \{ \delta \epsilon : \text{Proof } (\text{Lift } Q) \} \{ \phi \} \rightarrow$   $\delta \twoheadrightarrow \epsilon \rightarrow \Lambda$   $\phi$   $\delta \twoheadrightarrow \Lambda$   $\phi$   $\epsilon$ 

data  $\_ \twoheadrightarrow^+ \_$  :  $\forall \{Q\} \rightarrow$  Proof  $Q \rightarrow$  Proof  $Q \rightarrow$  Set where
 $\beta$  :  $\forall \{Q\} \phi (\delta : \text{Proof } (\text{Lift } Q)) \epsilon \rightarrow$  app ( $\Lambda$   $\phi$   $\delta$ )  $\epsilon \twoheadrightarrow^+$  subbot  $\delta$   $\epsilon$ 
 $\twoheadrightarrow^+$ trans :  $\forall \{Q\} \{ \gamma \delta \epsilon : \text{Proof } Q \} \rightarrow$   $\gamma \twoheadrightarrow^+ \delta \rightarrow \delta \twoheadrightarrow^+ \epsilon \rightarrow \gamma \twoheadrightarrow^+ \epsilon$ 
appl :  $\forall \{Q\} \{ \delta \delta' \epsilon \epsilon' : \text{Proof } Q \} \rightarrow$   $\delta \twoheadrightarrow^+ \delta' \rightarrow \epsilon \twoheadrightarrow^+ \epsilon' \rightarrow$  app  $\delta$   $\epsilon \twoheadrightarrow^+$  app  $\delta'$   $\epsilon'$ 
appr :  $\forall \{Q\} \{ \delta \delta' \epsilon \epsilon' : \text{Proof } Q \} \rightarrow$   $\delta \twoheadrightarrow^+ \delta' \rightarrow \epsilon \twoheadrightarrow^+ \epsilon' \rightarrow$  app  $\delta$   $\epsilon \twoheadrightarrow^+$  app  $\delta$   $\epsilon'$ 
 $\xi$  :  $\forall \{Q\} \{ \delta \epsilon : \text{Proof } (\text{Lift } Q) \} \{ \phi \} \rightarrow$   $\delta \twoheadrightarrow^+ \epsilon \rightarrow \Lambda$   $\phi$   $\delta \twoheadrightarrow^+ \Lambda$   $\phi$   $\epsilon$ 

data  $\_ \simeq \_$  :  $\forall \{Q\} \rightarrow$  Proof  $Q \rightarrow$  Proof  $Q \rightarrow$  Set1 where
 $\beta$  :  $\forall \{Q\} \{ \phi \} \{ \delta : \text{Proof } (\text{Lift } Q) \} \{ \epsilon \} \rightarrow$  app ( $\Lambda$   $\phi$   $\delta$ )  $\epsilon \simeq$  subbot  $\delta$   $\epsilon$ 
ref :  $\forall \{Q\} \{ \delta : \text{Proof } Q \} \rightarrow$   $\delta \simeq \delta$ 
 $\simeq$ sym :  $\forall \{Q\} \{ \delta \epsilon : \text{Proof } Q \} \rightarrow$   $\delta \simeq \epsilon \rightarrow \epsilon \simeq \delta$ 
 $\simeq$ trans :  $\forall \{Q\} \{ \delta \epsilon \in P : \text{Proof } Q \} \rightarrow$   $\delta \simeq \epsilon \rightarrow \epsilon \simeq P \rightarrow \delta \simeq P$ 
app :  $\forall \{Q\} \{ \delta M' \in N' : \text{Proof } Q \} \rightarrow$   $\delta \simeq M' \rightarrow \epsilon \simeq N' \rightarrow$  app  $\delta$   $\epsilon \simeq$  app  $M'$   $N'$ 
 $\Lambda$  :  $\forall \{Q\} \{ \delta \epsilon : \text{Proof } (\text{Lift } Q) \} \{ \phi \} \rightarrow$   $\delta \simeq \epsilon \rightarrow \Lambda$   $\phi$   $\delta \simeq \Lambda$   $\phi$   $\epsilon$ 

```

Lemma 7. 1. If $\delta \twoheadrightarrow \epsilon$ then $\delta[\sigma] \twoheadrightarrow \epsilon[\sigma]$.

2. If $\sigma \twoheadrightarrow \tau$ then $\delta[\sigma] \twoheadrightarrow \delta[\tau]$.

Proof. For part 2, we first prove that if $\sigma \twoheadrightarrow \tau$ then $\sigma + 1 \twoheadrightarrow \tau + 1$ using part 1. \square

```

sub1redl : ∀ {P} {Q} {ρ : Sub P Q} {δ ε : Proof P} → δ →1 ε → δ [ ρ ] →1 ε [ ρ ]
sub1redl {P} {Q} {ρ} (β .{P} {φ} {δ} {ε}) = subst (λ x → app (Λ φ (δ [ liftSub ρ ]))) (ε [ ρ ])
  (let open Equational-Reasoning (Proof Q) in
  ∴ (δ [ liftSub ρ ] [ botsub (ε [ ρ ]) ])
  ≡ δ [ botsub (ε [ ρ ]) • liftSub ρ ] [[ subcomp (botsub (ε [ ρ ])) (liftSub ρ) δ ]]
  ≡ δ [ ρ • botsub ε ] [[ subwd (sub-botsub ρ ε) δ ]]
  ≡ (δ [ botsub ε ] [ ρ ]) [ subcomp ρ (botsub ε) δ ])
  β
sub1redl (ξ δ→1ε) = ξ (sub1redl δ→1ε)
sub1redl (appl δ→1ε) = appl (sub1redl δ→1ε)
sub1redl (appr δ→1ε) = appr (sub1redl δ→1ε)

subredl : ∀ {P} {Q} {ρ : Sub P Q} {δ ε : Proof P} → δ → ε → δ [ ρ ] → ε [ ρ ]
subredl {Q = Q} {ρ = ρ} (β φ δ ε) = subst (λ x → app (Λ φ (δ [ liftSub ρ ]))) (ε [ ρ ])
  (let open Equational-Reasoning (Proof Q) in
  ∴ δ [ liftSub ρ ] [ botsub (ε [ ρ ]) ]
  ≡ δ [ botsub (ε [ ρ ]) • liftSub ρ ] [[ subcomp (botsub (ε [ ρ ])) (liftSub ρ) δ ]
  ≡ δ [ ρ • botsub ε ] [[ subwd (sub-botsub ρ ε) δ ]
  ≡ δ [ botsub ε ] [ ρ ] [ subcomp ρ (botsub ε) δ ]
  (β _ _ _))
subredl (→trans r r1) = →trans (subredl r) (subredl r1)
subredl (app r r1) = app (subredl r) (subredl r1)
subredl (ξ r) = ξ (subredl r)
subredl ref = ref

sub+redl : ∀ {P} {Q} {ρ : Sub P Q} {δ ε : Proof P} → δ →+ ε → δ [ ρ ] →+ ε [ ρ ]
sub+redl {Q = Q} {ρ = ρ} (β φ δ ε) = subst (λ x → app (Λ φ (δ [ liftSub ρ ]))) (ε [ ρ ])
  (let open Equational-Reasoning (Proof Q) in
  ∴ δ [ liftSub ρ ] [ botsub (ε [ ρ ]) ]
  ≡ δ [ botsub (ε [ ρ ]) • liftSub ρ ] [[ subcomp (botsub (ε [ ρ ])) (liftSub ρ) δ ]
  ≡ δ [ ρ • botsub ε ] [[ subwd (sub-botsub ρ ε) δ ]
  ≡ δ [ botsub ε ] [ ρ ] [ subcomp ρ (botsub ε) δ ]
  (β _ _ _))
sub+redl (→+trans r r1) = →+trans (sub+redl r) (subredl r1)
sub+redl (appl r r1) = appl (sub+redl r) (subredl r1)
sub+redl (appr r r1) = appr (subredl r) (sub+redl r1)
sub+redl (ξ r) = ξ (sub+redl r)

liftSub-red : ∀ {P} {Q} {ρ σ : Sub P Q} → (∀ x → ρ x → σ x) → (∀ x → liftSub ρ x → liftSub σ x)
liftSub-red ρ→σ ⊥ = ref
liftSub-red {ρ = ρ} ρ→σ (↑ x) = subst2 _→_ (sym (rep-is-sub _)) (sym (rep-is-sub _)) (ρ x)

subredr : ∀ {P} {Q} {ρ σ : Sub P Q} (δ : Proof P) → (∀ x → ρ x → σ x) → δ [ ρ ] → δ [ σ ]
subredr (var x) ρ→σ = ρ→σ x
subredr (app δ ε) ρ→σ = app (subredr δ ρ→σ) (subredr ε ρ→σ)
subredr (Λ φ δ) ρ→σ = ξ (subredr δ (liftSub-red ρ→σ))

```

The *strongly normalizable* terms are defined inductively as follows.

```
data SN {P} : Proof P → Set1 where
  SNI : ∀ {φ} → (∀ ψ → φ →1 ψ → SN ψ) → SN φ
```

Lemma 8. 1. If $\delta\epsilon \in SN$ then $\delta \in SN$ and $\epsilon \in SN$.

2. If $\delta[\perp := N] \in SN$ then $\delta \in SN$.

3. If $\delta \in SN$ and $\delta \rightarrow \epsilon$ then $\epsilon \in SN$.

4. If $\delta[x := \epsilon] \in SN$ and $\epsilon \in SN$ then $(\lambda x : \phi.\delta)\epsilon \in SN$.

```
SNappl : ∀ {Q} {δ ε : Proof Q} → SN (app δ ε) → SN δ
SNappl {Q} {δ} {ε} (SNI δε-is-SN) = SNI (λ δ' δ→1δ' → SNappl (δε-is-SN (app δ' ε)) (appl
```

```
SNappr : ∀ {Q} {δ ε : Proof Q} → SN (app δ ε) → SN ε
SNappr {Q} {δ} {ε} (SNI δε-is-SN) = SNI (λ ε' ε→1ε' → SNappr (δε-is-SN (app δ ε')) (appr ε
```

```
SNsub : ∀ {Q} {δ : Proof (Lift Q)} {ε} → SN (subbot δ ε) → SN δ
SNsub {Q} {δ} {ε} (SNI δε-is-SN) = SNI (λ δ' δ→1δ' → SNsub (δε-is-SN (δ' [ botsub ε ])) (S
```

The rules of deduction of the system are as follows.

$$\frac{\Gamma \text{ valid}}{\Gamma \vdash p : \phi} (p : \phi \in \Gamma)$$

$$\frac{\Gamma \vdash \delta : \phi \rightarrow \psi}{\Gamma \vdash \delta\epsilon : \psi} \quad \Gamma \vdash \epsilon : \phi$$

$$\frac{\Gamma, p : \phi \vdash \delta : \psi}{\Gamma \vdash \lambda p : \phi.\delta : \phi \rightarrow \psi}$$

```
data _⊢_::_ : ∀ {P} → PContext P → Proof P → Prp → Set1 where
  var : ∀ {P} {Γ : PContext P} {p} → Γ ⊢ var p :: propof p Γ
  app : ∀ {P} {Γ : PContext P} {δ} {ε} {φ} {ψ} → Γ ⊢ δ :: φ ⇒ ψ → Γ ⊢ ε :: φ → Γ ⊢ ap
  Λ : ∀ {P} {Γ : PContext P} {φ} {δ} {ψ} → Γ , φ ⊢ δ :: ψ → Γ ⊢ Λ φ δ :: φ ⇒ ψ
```

```
module PHOPL where
open import Prelims
```

4 Predicative Higher-Order Propositional Logic

Fix sets of *proof variables* and *term variables*.

The syntax of the system is given by the following grammar.

Proof	$\delta ::= p \mid \delta\delta \mid \lambda p : \phi.\delta$
Term	$M, \phi ::= x \mid \perp \mid MM \mid \phi \rightarrow \phi \mid \lambda x : A.M$
Type	$A ::= \Omega \mid A \rightarrow A$
Term Context	$\Gamma ::= \langle \rangle \mid \Gamma, x : A$
Proof Context	$\Delta ::= \langle \rangle \mid \Delta, p : \phi$
Judgement	$\mathcal{J} ::= \Gamma \text{ valid} \mid \Gamma \vdash M : A \mid \Gamma, \Delta \text{ valid} \mid \Gamma, \Delta \vdash \delta : \phi$

where p ranges over proof variables and x ranges over term variables. The variable p is bound within δ in the proof $\lambda p : \phi.\delta$, and the variable x is bound within M in the term $\lambda x : A.M$. We identify proofs and terms up to α -conversion.

In the implementation, we write **Term**(V) for the set of all terms with free variables a subset of V , where $V : \mathbf{FinSet}$.

```

infix 80 _=>_
data Type : Set where
  Ω : Type
  _=>_ : Type → Type → Type

--Context V P is the set of all contexts whose domain consists of the term variables in V
infix 80 _,_
data TContext : FinSet → Set where
  ⟨⟩ : TContext ∅
  _,_ : ∀ {V} → TContext V → Type → TContext (Lift V)

--Term V is the set of all terms M with FV(M) ⊆ V
data Term : FinSet → Set where
  var : ∀ {V} → El V → Term V
  ⊥ : ∀ {V} → Term V
  app : ∀ {V} → Term V → Term V → Term V
  Λ : ∀ {V} → Type → Term (Lift V) → Term V
  _=>_ : ∀ {V} → Term V → Term V → Term V

data PContext (V : FinSet) : FinSet → Set where
  ⟨⟩ : PContext V ∅
  _,_ : ∀ {P} → PContext V P → Term V → PContext V (Lift P)

--Proof V P is the set of all proofs with term variables among V and proof variables among P
data Proof (V : FinSet) : FinSet → Set₁ where
  var : ∀ {P} → El P → Proof V P
  app : ∀ {P} → Proof V P → Proof V P → Proof V P
  Λ : ∀ {P} → Term V → Proof V (Lift P) → Proof V P

```

Let $U, V : \mathbf{FinSet}$. A *replacement* from U to V is just a function $U \rightarrow V$. Given a term $M : \mathbf{Term}(U)$ and a replacement $\rho : U \rightarrow V$, we write $M\{\rho\} : \mathbf{Term}(V)$ for the result of replacing each variable x in M with $\rho(x)$.

```

infix 60 _<_>
_<_> : ∀ {U V} → Term U → Rep U V → Term V
(var x) < ρ > = var (ρ x)
⊥ < ρ > = ⊥
(app M N) < ρ > = app (M < ρ >) (N < ρ >)
(Λ A M) < ρ > = Λ A (M < lift ρ >)
(φ ⇒ ψ) < ρ > = (φ < ρ >) ⇒ (ψ < ρ >)

```

With this as the action on arrows, **Term**() becomes a functor **FinSet** → **Set**.

```

repwd : ∀ {U V : FinSet} {ρ ρ' : El U → El V} → ρ ~ ρ' → ∀ M → M < ρ > ≡ M < ρ' >
repwd ρ-is-ρ' (var x) = wd var (ρ-is-ρ' x)
repwd ρ-is-ρ' ⊥ = ref
repwd ρ-is-ρ' (app M N) = wd2 app (repwd ρ-is-ρ' M) (repwd ρ-is-ρ' N)
repwd ρ-is-ρ' (Λ A M) = wd (Λ A) (repwd (liftwd ρ-is-ρ') M)
repwd ρ-is-ρ' (φ ⇒ ψ) = wd2 _⇒_ (repwd ρ-is-ρ' φ) (repwd ρ-is-ρ' ψ)

repid : ∀ {V : FinSet} M → M < id (El V) > ≡ M
repid (var x) = ref
repid ⊥ = ref
repid (app M N) = wd2 app (repid M) (repid N)
repid (Λ A M) = wd (Λ A) (trans (repwd liftid M) (repid M))
repid (φ ⇒ ψ) = wd2 _⇒_ (repid φ) (repid ψ)

repcomp : ∀ {U V W : FinSet} (σ : El V → El W) (ρ : El U → El V) M → M < σ ∘ ρ > ≡ M
repcomp ρ σ (var x) = ref
repcomp ρ σ ⊥ = ref
repcomp ρ σ (app M N) = wd2 app (repcomp ρ σ M) (repcomp ρ σ N)
repcomp ρ σ (Λ A M) = wd (Λ A) (trans (repwd liftcomp M) (repcomp (lift ρ) (lift σ) M))
repcomp ρ σ (φ ⇒ ψ) = wd2 _⇒_ (repcomp ρ σ φ) (repcomp ρ σ ψ)

```

A *substitution* σ from U to V , $\sigma : U \Rightarrow V$, is a function $\sigma : U \rightarrow \mathbf{Term}(V)$.

```

Sub : FinSet → FinSet → Set
Sub U V = El U → Term V

```

The identity substitution $\text{id}_V : V \Rightarrow V$ is defined as follows.

```

idSub : ∀ V → Sub V V
idSub _ = var

```

Given $\sigma : U \Rightarrow V$ and $M : \mathbf{Term}(U)$, we want to define $M[\sigma] : \mathbf{Term}(V)$, the result of applying the substitution σ to M . Only after this will we be able to define the composition of two substitutions. However, there is some work we need to do before we are able to do this.

We can define the composition of a substitution and a replacement as follows.

```

infix 75 _•₁_
_•₁_ : ∀ {U} {V} {W} → Rep V W → Sub U V → Sub U W
(ρ •₁ σ) u = σ u < ρ >

```

(On the other side, given $\rho : U \rightarrow V$ and $\sigma : V \Rightarrow W$, the composition is just function composition $\sigma \circ \rho : U \Rightarrow W$.)

Given a substitution $\sigma : U \Rightarrow V$, define the substitution $\sigma + 1 : U + 1 \Rightarrow V + 1$ as follows.

```

liftSub : ∀ {U} {V} → Sub U V → Sub (Lift U) (Lift V)
liftSub _ ⊥ = var ⊥
liftSub σ (↑ x) = σ x < ↑ >

```

```

liftSub-wd : ∀ {U V} {σ σ' : Sub U V} → σ ~ σ' → liftSub σ ~ liftSub σ'
liftSub-wd σ-is-σ' ⊥ = ref
liftSub-wd σ-is-σ' (↑ x) = wd (λ x → x < ↑ >) (σ-is-σ' x)

```

Lemma 9. *The operations ffl_1 and $(-)+1$ satisfiesd the following properties.*

1. $\text{id}_V + 1 = \text{id}_{V+1}$
2. For $\rho : V \rightarrow W$ and $\sigma : U \Rightarrow V$, we have $(\rho \bullet \sigma) + 1 = (\rho + 1) \bullet (\sigma + 1)$.
3. For $\sigma : V \Rightarrow W$ and $\rho : U \rightarrow V$, we have $(\sigma \circ \rho) + 1 = (\sigma + 1) \circ (\rho + 1)$.

```

liftSub-id : ∀ {V : FinSet} → liftSub (idSub V) ~ idSub (Lift V)
liftSub-id ⊥ = ref
liftSub-id (↑ x) = ref

```

```

liftSub-comp₁ : ∀ {U V W : FinSet} (σ : Sub U V) (ρ : Rep V W) →
  liftSub (ρ •₁ σ) ~ lift ρ •₁ liftSub σ
liftSub-comp₁ σ ρ ⊥ = ref
liftSub-comp₁ {W = W} σ ρ (↑ x) = let open Equational-Reasoning (Term (Lift W)) in
  ∴ σ x < ρ > < ↑ >
  ≡ σ x < ↑ ∘ ρ > [[ repcomp ↑ ρ (σ x) ]]
  ≡ σ x < ↑ > < lift ρ > [ repcomp (lift ρ) ↑ (σ x) ]
--because lift ρ (↑ x) = ↑ (ρ x)

```

```

liftSub-comp₂ : ∀ {U V W : FinSet} (σ : Sub V W) (ρ : Rep U V) →
  liftSub (σ ∘ ρ) ~ liftSub σ ∘ lift ρ
liftSub-comp₂ σ ρ ⊥ = ref
liftSub-comp₂ σ ρ (↑ x) = ref

```

Now define $M[\sigma]$ as follows.

```

--Term is a monad with unit var and the following multiplication
infix 60 _⟦_⟧
_⟦_⟧ : ∀ {U V : FinSet} → Term U → Sub U V → Term V

```

```

(var x)    [ [ σ ] ] = σ x
⊥         [ [ σ ] ] = ⊥
(app M N) [ [ σ ] ] = app (M [ [ σ ] ]) (N [ [ σ ] ])
(Λ A M)   [ [ σ ] ] = Λ A (M [ [ liftSub σ ] ])
(φ ⇒ ψ)   [ [ σ ] ] = (φ [ [ σ ] ]) ⇒ (ψ [ [ σ ] ])

```

```

subwd : ∀ {U V : FinSet} {σ σ' : Sub U V} → σ ~ σ' → ∀ M → M [ [ σ ] ] ≡ M [ [ σ' ] ]
subwd σ-is-σ' (var x) = σ-is-σ' x
subwd σ-is-σ' ⊥ = ref
subwd σ-is-σ' (app M N) = wd2 app (subwd σ-is-σ' M) (subwd σ-is-σ' N)
subwd σ-is-σ' (Λ A M) = wd (Λ A) (subwd (liftSub-wd σ-is-σ') M)
subwd σ-is-σ' (φ ⇒ ψ) = wd2 _⇒_ (subwd σ-is-σ' φ) (subwd σ-is-σ' ψ)

```

This interacts with our previous operations in a good way:

Lemma 10. 1. $M[id_V] \equiv M$

2. $M[\rho \bullet \sigma] \equiv M[\sigma]\{\rho\}$

3. $M[\sigma \circ \rho] \equiv M < \rho > [\sigma]$

```

subid : ∀ {V : FinSet} (M : Term V) → M [ [ idSub V ] ] ≡ M
subid (var x) = ref
subid ⊥ = ref
subid (app M N) = wd2 app (subid M) (subid N)
subid {V} (Λ A M) = let open Equational-Reasoning (Term V) in
  ∴ Λ A (M [ [ liftSub (idSub V) ] ])
  ≡ Λ A (M [ [ idSub (Lift V) ] ])      [ wd (Λ A) (subwd liftSub-id M) ]
  ≡ Λ A M                             [ wd (Λ A) (subid M) ]
subid (φ ⇒ ψ) = wd2 _⇒_ (subid φ) (subid ψ)

```

```

rep-sub : ∀ {U} {V} {W} (σ : Sub U V) (ρ : Rep V W) (M : Term U) → M [ [ σ ] ] < ρ > ≡ M [ [ σ ∘ ρ ] ]
rep-sub σ ρ (var x) = ref
rep-sub σ ρ ⊥ = ref
rep-sub σ ρ (app M N) = wd2 app (rep-sub σ ρ M) (rep-sub σ ρ N)
rep-sub {W = W} σ ρ (Λ A M) = let open Equational-Reasoning (Term W) in
  ∴ Λ A ((M [ [ liftSub σ ] ]) < lift ρ >)
  ≡ Λ A (M [ [ lift ρ •1 liftSub σ ] ]) [ wd (Λ A) (rep-sub (liftSub σ) (lift ρ) M) ]
  ≡ Λ A (M [ [ liftSub (ρ •1 σ) ] ])   [[ wd (Λ A) (subwd (liftSub-comp1 σ ρ) M) ]]
rep-sub σ ρ (φ ⇒ ψ) = wd2 _⇒_ (rep-sub σ ρ φ) (rep-sub σ ρ ψ)

```

```

sub-rep : ∀ {U} {V} {W} (σ : Sub V W) (ρ : Rep U V) M → M < ρ > [ [ σ ] ] ≡ M [ [ σ ∘ ρ ] ]
sub-rep σ ρ (var x) = ref
sub-rep σ ρ ⊥ = ref
sub-rep σ ρ (app M N) = wd2 app (sub-rep σ ρ M) (sub-rep σ ρ N)
sub-rep {W = W} σ ρ (Λ A M) = let open Equational-Reasoning (Term W) in
  ∴ Λ A ((M < lift ρ >) [ [ liftSub σ ] ])

```

$$\begin{aligned}
&\equiv \Lambda A (M \llbracket \text{liftSub } \sigma \circ \text{lift } \rho \rrbracket) \quad [\text{wd } (\Lambda A) (\text{sub-rep } (\text{liftSub } \sigma) (\text{lift } \rho) M)] \\
&\equiv \Lambda A (M \llbracket \text{liftSub } (\sigma \circ \rho) \rrbracket) \quad [[\text{wd } (\Lambda A) (\text{subwd } (\text{liftSub-comp}_2 \sigma \rho) M)]] \\
\text{sub-rep } \sigma \rho (\phi \Rightarrow \psi) &= \text{wd2 } _ \Rightarrow _ (\text{sub-rep } \sigma \rho \phi) (\text{sub-rep } \sigma \rho \psi)
\end{aligned}$$

We define the composition of two substitutions, as follows.

```

infix 75 _•_
_•_ : ∀ {U V W : FinSet} → Sub V W → Sub U V → Sub U W
(σ • ρ) x = ρ x ⌊ σ ⌋

```

Lemma 11. *Let $\sigma : V \Rightarrow W$ and $\rho : U \Rightarrow V$.*

$$1. (\sigma \bullet \rho) + 1 = (\sigma + 1) \bullet (\rho + 1)$$

$$2. M[\sigma \bullet \rho] \equiv M[\rho][\sigma]$$

```

liftSub-comp : ∀ {U} {V} {W} (σ : Sub V W) (ρ : Sub U V) →
  liftSub (σ • ρ) ~ liftSub σ • liftSub ρ
liftSub-comp σ ρ ⊥ = ref
liftSub-comp σ ρ (↑ x) = trans (rep-sub σ ↑ (ρ x)) (sym (sub-rep (liftSub σ) ↑ (ρ x)))

```

```

subcomp : ∀ {U} {V} {W} (σ : Sub V W) (ρ : Sub U V) M → M ⌊ σ • ρ ⌋ ≡ M ⌊ ρ ⌋ ⌊ σ ⌋
subcomp σ ρ (var x) = ref
subcomp σ ρ ⊥ = ref
subcomp σ ρ (app M N) = wd2 app (subcomp σ ρ M) (subcomp σ ρ N)
subcomp σ ρ (Λ A M) = wd (Λ A) (trans (subwd (liftSub-comp σ ρ) M) (subcomp (liftSub σ) ρ M))
subcomp σ ρ (φ ⇒ ψ) = wd2 _⇒_ (subcomp σ ρ φ) (subcomp σ ρ ψ)

```

Lemma 12. *The finite sets and substitutions form a category under this composition.*

```

assoc : ∀ {U V W X} {ρ : Sub W X} {σ : Sub V W} {τ : Sub U V} →
  ρ • (σ • τ) ~ (ρ • σ) • τ
assoc {U} {V} {W} {X} {ρ} {σ} {τ} x = sym (subcomp ρ σ (τ x))

```

```

subunitl : ∀ {U} {V} {σ : Sub U V} → idSub V • σ ~ σ
subunitl {U} {V} {σ} x = subid (σ x)

```

```

subunitr : ∀ {U} {V} {σ : Sub U V} → σ • idSub U ~ σ
subunitr _ = ref

```

-- The second monad law

```

rep-is-sub : ∀ {U} {V} {ρ : El U → El V} M → M < ρ > ≡ M ⌊ var ∘ ρ ⌋
rep-is-sub (var x) = ref
rep-is-sub ⊥ = ref
rep-is-sub (app M N) = wd2 app (rep-is-sub M) (rep-is-sub N)
rep-is-sub {V = V} {ρ} (Λ A M) = let open Equational-Reasoning (Term V) in

```



```

    ∴ Λ A (M < lift ρ >)
    ≡ Λ A (M [ var ∘ lift ρ ]) [ wd (Λ A) (rep-is-sub M) ]
    ≡ Λ A (M [ liftSub var ∘ lift ρ ]) [[ wd (Λ A) (subwd (λ x → liftSub-id (lift ρ x)) M) ]
    ≡ Λ A (M [ liftSub (var ∘ ρ) ]) [[ wd (Λ A) (subwd (liftSub-comp2 var ρ) M) ]]
--wd (Λ A) (trans (rep-is-sub M) (subwd {!!} M))
rep-is-sub (ϕ ⇒ ψ) = wd2 _⇒_ (rep-is-sub ϕ) (rep-is-sub ψ)

typeof : ∀ {V} → El V → TContext V → Type
typeof ⊥ (_ , A) = A
typeof (↑ x) (Γ , _) = typeof x Γ

propof : ∀ {V} {P} → El P → PContext V P → Term V
propof ⊥ (_ , ϕ) = ϕ
propof (↑ p) (Γ , _) = propof p Γ

liftSub-var' : ∀ {U} {V} (ρ : El U → El V) → liftSub (var ∘ ρ) ~ var ∘ lift ρ
liftSub-var' ρ ⊥ = ref
liftSub-var' ρ (↑ x) = ref

botsub : ∀ {V} → Term V → Sub (Lift V) V
botsub M ⊥ = M
botsub _ (↑ x) = var x

sub-botsub : ∀ {U} {V} (σ : Sub U V) (M : Term U) (x : El (Lift U)) →
  botsub M x [ σ ] ≡ liftSub σ x [ botsub (M [ σ ]) ]
sub-botsub σ M ⊥ = ref
sub-botsub σ M (↑ x) = let open Equational-Reasoning (Term _) in
  ∴ σ x
  ≡ σ x [ idSub _ ] [ subid (σ x) ]
  ≡ σ x < ↑ > [ botsub (M [ σ ]) ] [ sub-rep (botsub (M [ σ ])) ↑ (σ x) ]

rep-botsub : ∀ {U} {V} (ρ : El U → El V) (M : Term U) (x : El (Lift U)) →
  botsub M x < ρ > ≡ botsub (M < ρ >) (lift ρ x)
rep-botsub ρ M x = trans (rep-is-sub (botsub M x))
  (trans (sub-botsub (var ∘ ρ) M x) (trans (subwd (λ x1 → wd (λ y → botsub y x1) (sym (
    wd (λ x → x [ botsub (M < ρ >)] (liftSub-var' ρ x))))
--TODO Inline this?

subbot : ∀ {V} → Term (Lift V) → Term V → Term V
subbot M N = M [ botsub N ]

```

We write $M \simeq N$ iff the terms M and N are β -convertible, and similarly for proofs.

```

data _→_ : ∀ {V} → Term V → Term V → Set where
  β : ∀ {V} A (M : Term (Lift V)) N → app (Λ A M) N → subbot M N

```

```

ref : ∀ {V} {M : Term V} → M → M
→trans : ∀ {V} {M N P : Term V} → M → N → N → P → M → P
app : ∀ {V} {M M' N N' : Term V} → M → M' → N → N' → app M N → app M' N'
Λ : ∀ {V} {M N : Term (Lift V)} {A} → M → N → Λ A M → Λ A N
imp : ∀ {V} {φ φ' ψ ψ' : Term V} → φ → φ' → ψ → ψ' → φ ⇒ ψ → φ' ⇒ ψ'

repred : ∀ {U} {V} {ρ : El U → El V} {M N : Term U} → M → N → M < ρ > → N < ρ >
repred {U} {V} {ρ} (β A M N) = subst (λ x → app (Λ A (M < lift ρ >)) (N < ρ >) → x) (
repred ref = ref
repred (→trans M→N N→P) = →trans (repred M→N) (repred N→P)
repred (app M→N M'→N') = app (repred M→N) (repred M'→N')
repred (Λ M→N) = Λ (repred M→N)
repred (imp φ→φ' ψ→ψ') = imp (repred φ→φ') (repred ψ→ψ')

liftSub-red : ∀ {U} {V} {ρ σ : Sub U V} → (∀ x → ρ x → σ x) → (∀ x → liftSub ρ x →
liftSub-red ρ→σ ⊥ = ref
liftSub-red ρ→σ (↑ x) = repred (ρ→σ x)

subred : ∀ {U} {V} {ρ σ : Sub U V} (M : Term U) → (∀ x → ρ x → σ x) → M [ ρ ] → M [
subred (var x) ρ→σ = ρ→σ x
subred ⊥ ρ→σ = ref
subred (app M N) ρ→σ = app (subred M ρ→σ) (subred N ρ→σ)
subred (Λ A M) ρ→σ = Λ (subred M (liftSub-red ρ→σ))
subred (φ ⇒ ψ) ρ→σ = imp (subred φ ρ→σ) (subred ψ ρ→σ)

subsub : ∀ {U} {V} {W} (σ : Sub V W) (ρ : Sub U V) M → M [ ρ ] [ σ ] ≡ M [ σ • ρ ]
subsub σ ρ (var x) = ref
subsub σ ρ ⊥ = ref
subsub σ ρ (app M N) = wd2 app (subsub σ ρ M) (subsub σ ρ N)
subsub σ ρ (Λ A M) = wd (Λ A) (trans (subsub (liftSub σ) (liftSub ρ) M)
(subwd (λ x → sym (liftSub-comp σ ρ x)) M))
subsub σ ρ (φ ⇒ ψ) = wd2 _⇒_ (subsub σ ρ φ) (subsub σ ρ ψ)

subredr : ∀ {U} {V} {σ : Sub U V} {M N : Term U} → M → N → M [ σ ] → N [ σ ]
subredr {U} {V} {σ} (β A M N) = subst (λ x → app (Λ A (M [ liftSub σ ])) (N [ σ ])) (
(sym (trans (subsub (botsub (N [ σ ])) (liftSub σ) M) (subwd (λ x → sym (sub-botsub σ
subredr ref = ref
subredr (→trans M→N N→P) = →trans (subredr M→N) (subredr N→P)
subredr (app M→M' N→N') = app (subredr M→M') (subredr N→N')
subredr (Λ M→N) = Λ (subredr M→N)
subredr (imp φ→φ' ψ→ψ') = imp (subredr φ→φ') (subredr ψ→ψ')

data _≈_ : ∀ {V} → Term V → Term V → Set1 where
β : ∀ {V} {A} {M : Term (Lift V)} {N} → app (Λ A M) N ≈ subbot M N
ref : ∀ {V} {M : Term V} → M ≈ M
≈sym : ∀ {V} {M N : Term V} → M ≈ N → N ≈ M

```

$\simeq\text{trans} : \forall \{V\} \{M N P : \text{Term } V\} \rightarrow M \simeq N \rightarrow N \simeq P \rightarrow M \simeq P$
 $\text{app} : \forall \{V\} \{M M' N N' : \text{Term } V\} \rightarrow M \simeq M' \rightarrow N \simeq N' \rightarrow \text{app } M N \simeq \text{app } M' N'$
 $\Lambda : \forall \{V\} \{M N : \text{Term } (\text{Lift } V)\} \{A\} \rightarrow M \simeq N \rightarrow \Lambda A M \simeq \Lambda A N$
 $\text{imp} : \forall \{V\} \{\phi \phi' \psi \psi' : \text{Term } V\} \rightarrow \phi \simeq \phi' \rightarrow \psi \simeq \psi' \rightarrow \phi \Rightarrow \psi \simeq \phi' \Rightarrow \psi'$

The *strongly normalizable* terms are defined inductively as follows.

$\text{data SN } \{V\} : \text{Term } V \rightarrow \text{Set}_1 \text{ where}$
 $\text{SNI} : \forall \{M\} \rightarrow (\forall N \rightarrow M \twoheadrightarrow N \rightarrow \text{SN } N) \rightarrow \text{SN } M$

Lemma 13. 1. If $MN \in \text{SN}$ then $M \in \text{SN}$ and $N \in \text{SN}$.

2. If $M[x := N] \in \text{SN}$ then $M \in \text{SN}$.

3. If $M \in \text{SN}$ and $M \triangleright N$ then $N \in \text{SN}$.

4. If $M[x := N]\vec{P} \in \text{SN}$ and $N \in \text{SN}$ then $(\lambda x M)N\vec{P} \in \text{SN}$.

$\text{SNappl} : \forall \{V\} \{M N : \text{Term } V\} \rightarrow \text{SN } (\text{app } M N) \rightarrow \text{SN } M$
 $\text{SNappl } \{V\} \{M\} \{N\} (\text{SNI } MN\text{-is-SN}) = \text{SNI } (\lambda P M \triangleright P \rightarrow \text{SNappl } (MN\text{-is-SN } (\text{app } P N)) (\text{app } M \triangleright P))$

$\text{SNappr} : \forall \{V\} \{M N : \text{Term } V\} \rightarrow \text{SN } (\text{app } M N) \rightarrow \text{SN } N$
 $\text{SNappr } \{V\} \{M\} \{N\} (\text{SNI } MN\text{-is-SN}) = \text{SNI } (\lambda P N \triangleright P \rightarrow \text{SNappr } (MN\text{-is-SN } (\text{app } M P)) (\text{app } \text{ref } P))$

$\text{SNsub} : \forall \{V\} \{M : \text{Term } (\text{Lift } V)\} \{N\} \rightarrow \text{SN } (\text{subbot } M N) \rightarrow \text{SN } M$
 $\text{SNsub } \{V\} \{M\} \{N\} (\text{SNI } MN\text{-is-SN}) = \text{SNI } (\lambda P M \triangleright P \rightarrow \text{SNsub } (MN\text{-is-SN } (P \ll \text{botsub } N \gg)) (\text{subbot } P))$

The rules of deduction of the system are as follows.

$$\begin{array}{c}
\frac{}{\langle \rangle \text{ valid}} \quad \frac{\Gamma \text{ valid}}{\Gamma, x : A \text{ valid}} \quad \frac{\Gamma \vdash \phi : \Omega}{\Gamma, p : \phi \text{ valid}} \\
\\
\frac{\Gamma \text{ valid}}{\Gamma \vdash x : A} (x : A \in \Gamma) \quad \frac{\Gamma \text{ valid}}{\Gamma \vdash p : \phi} (p : \phi \in \Gamma) \\
\\
\frac{\Gamma \text{ valid}}{\Gamma \vdash \perp : \Omega} \quad \frac{\Gamma \vdash \phi : \Omega \quad \Gamma \vdash \psi : \Omega}{\Gamma \vdash \phi \rightarrow \psi : \Omega} \\
\\
\frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \quad \frac{\Gamma \vdash \delta : \phi \rightarrow \psi \quad \Gamma \vdash \epsilon : \phi}{\Gamma \vdash \delta \epsilon : \psi} \\
\\
\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A. M : A \rightarrow B} \quad \frac{\Gamma, p : \phi \vdash \delta : \psi}{\Gamma \vdash \lambda p : \phi. \delta : \phi \rightarrow \psi} \\
\\
\frac{\Gamma \vdash \delta : \phi \quad \Gamma \vdash \psi : \Omega}{\Gamma \vdash \delta : \psi} (\phi \simeq \psi)
\end{array}$$

```

mutual
data Tvalid : ∀ {V} → TContext V → Set1 where
  ⟨⟩ : Tvalid ⟨⟩
  _,_ : ∀ {V} {Γ : TContext V} → Tvalid Γ → ∀ A → Tvalid (Γ , A)

data _⊢_:_ : ∀ {V} → TContext V → Term V → Type → Set1 where
  var : ∀ {V} {Γ : TContext V} {x} → Tvalid Γ → Γ ⊢ var x : typeof x Γ
  ⊥ : ∀ {V} {Γ : TContext V} → Tvalid Γ → Γ ⊢ ⊥ : Ω
  imp : ∀ {V} {Γ : TContext V} {ϕ} {ψ} → Γ ⊢ ϕ : Ω → Γ ⊢ ψ : Ω → Γ ⊢ ϕ ⇒ ψ : Ω
  app : ∀ {V} {Γ : TContext V} {M} {N} {A} {B} → Γ ⊢ M : A ⇒ B → Γ ⊢ N : A → Γ ⊢ app M N : B
  Λ : ∀ {V} {Γ : TContext V} {A} {M} {B} → Γ , A ⊢ M : B → Γ ⊢ Λ A M : A ⇒ B

data Pvalid : ∀ {V} {P} → TContext V → PContext V P → Set1 where
  ⟨⟩ : ∀ {V} {Γ : TContext V} → Tvalid Γ → Pvalid Γ ⟨⟩
  _,_ : ∀ {V} {P} {Γ : TContext V} {Δ : PContext V P} {ϕ : Term V} → Pvalid Γ Δ → Γ ⊢ ϕ : PContext V P

data _,,_⊢_:_ : ∀ {V} {P} → TContext V → PContext V P → Proof V P → Term V → Set1 where
  var : ∀ {V} {P} {Γ : TContext V} {Δ : PContext V P} {p} → Pvalid Γ Δ → Γ , Δ ⊢ var p : PContext V P
  app : ∀ {V} {P} {Γ : TContext V} {Δ : PContext V P} {δ} {ϵ} {ϕ} {ψ} → Γ , Δ ⊢ δ :: ϕ → Γ , Δ ⊢ ϵ :: ψ → Γ , Δ ⊢ app δ ϵ : PContext V P
  Λ : ∀ {V} {P} {Γ : TContext V} {Δ : PContext V P} {ϕ} {δ} {ψ} → Γ , Δ ⊢ ϕ :: ψ → Γ , Δ ⊢ δ :: ψ → Γ , Δ ⊢ Λ ϕ δ : PContext V P
  conv : ∀ {V} {P} {Γ : TContext V} {Δ : PContext V P} {δ} {ϕ} {ψ} → Γ , Δ ⊢ δ :: ϕ → Γ , Δ ⊢ ϕ :: ψ → Γ , Δ ⊢ conv δ ϕ : PContext V P

```