le cmaci jdaske

la .varik. .VALefor.

April 12, 2024

Contents

Chapter 1

le me'oi .disclaimer.

ni'o na mulno .i la .varik. cu stidi lo nu na ci'au .ue sai lo nu vo'a mutce lo ka ce'u bixygau pe'a

Chapter 2

le vrici

```
open import Data.Fin
  using (
    Fin
  )
open import Data.Nat
  using (
    ℕ
  )
open import Function
  using (
    _∘₂_;
    _on_;
    flip;
    _∘_;
    _$_
  )
  renaming (
    _|>_ to _▷_
  )
open import Data.Bool
  using (
    not;
    T?
  )
open import Data.List
  as 𝕃
  using (
    map;
```

```
    zip;
    List
  )
  renaming (
    lookup to _!_
  )
open import Data.Maybe
  using (
    nothing;
    Maybe;
    just
  )
  renaming (
    map to map_m
  )
open import Data.String
  using (
    String
  )
open import Data.Product
  using (
    Σ;
    _×_;
    _,_;
    proj_1;
    proj_2
  )
open import Data.Rational
  using (
    ℚ
  )
open import Relation.Binary
  using (
    Setoid
  )
open import Truthbrary.Record.Eq
  using (
    _≡^b_;
    _≟_;
    Eq
  )
open import Truthbrary.Record.LLC
  using (
```

```
    nu,iork;
    length;
    _∉_;
    _∈_;
    LL;
    UL
  )
open import Data.Empty.Polymorphic
  using (
    ⊥
  )
open import Relation.Binary.PropositionalEquality
  using (
    refl;
    _≡_
  )

import Data.Vec
  as 𝕍
import Data.List.Relation.Unary.All
  as LUA
  using (
    _∷_;
    All;
    []
  )
```

Chapter 3

le vrici je fancu

3.1 la .cmimakonk.

ni'o la .cmimakonk. cu ctaipe le su'u ro da poi ke'a co'e zo'u ro de poi ke'a co'e zo'u da cmima lo konkatena be da bei de

```
cmimakonk : ∀ {a} → {A : Set a}
          → ⦃ _ : Eq A ⦄
          → (x : A)
          → (xs : List A)
          → x ∈_ $ x 𝕃.:: xs
cmimakonk x xs = sym $ begin
  length (𝕃.take 1 $ 𝕃.filter (x ≟_) $ x 𝕃.:: _) ≡⟨ refl ⟩
  _ ≡⟨ x≡1↑f[x::xs'] x xs ▷ sym ▷ cong length ⟩
  length (x 𝕃.:: 𝕃.[]) ≡⟨ refl ⟩
  1 ■
  where
  open import Relation.Binary.PropositionalEquality
  x≡1↑f[x::xs'] : ∀ {a} → {A : Set a}
    → ⦃ _ : Eq A ⦄
    → (x : A)
    → (xs : List A)
    → (_≡_
         (x 𝕃.:: 𝕃.[])
         (𝕃.take 1 $ 𝕃.filter (x ≟_)
           (x 𝕃.::_ $ 𝕍.toList $ 𝕍.fromList xs)))
  x≡1↑f[x::xs'] = {!!}
  open ≡-Reasoning
```

3.2 la .cmimapred.

ni'o la .cmimapred. cu ctaipe le su'u ro da poi ke'a co'e zo'u ro de poi ke'a co'e zo'u ro di poi ke'a co'e zo'u da cmima di naja lo konkatena be de bei di

```
cmimapred : ∀ {a} → {A : Set a}
          → ⦃ _ : Eq A ⦄
          → (x z : A)
          → (xs : List A)
          → x ∈ xs
          → x ∈_ $ z 𝕃.:: xs
cmimapred = {!!}
```

3.3 la'oi .Dun.

ni'o xu sarcu fa lo nu ciksi bau la .lojban.

```
Dun : ∀ {a} → {A : Set a}
    → ⦃ Eq A ⦄
    → (x z : List A)
    → Set a
Dun x z = LUA.All (_∈ x) z × LUA.All (_∈ z) x
```

3.3.1 le ctaipe be zo'e ja le su'u la'oi .Dun. mapti

```
module DunVeritasJaZo'e where
  T : ∀ {a} → {A : Set a}
    → ⦃ _ : Eq A ⦄
    → {i j k : List A}
    → Dun i j
    → Dun j k
    → Dun i k
  T = {!!}
```

3.4 la'oi .nu,iorks.

ni'o ro da zo'u da cmima la'o zoi. nu,iorks x .zoi. jo cu du lo su'o cmima be la'oi .x.

```
nu,iorks : ∀ {a} → {A : Set a} → ⦃ _ : Eq A ⦄ → List A → List A
nu,iorks 𝕃.[] = 𝕃.[]
nu,iorks (x 𝕃.:: z) = x 𝕃.:: 𝕃.filter (_≢?_ x) (nu,iorks z)
  where
  _≢?_ = T? ∘₂ not ∘₂ _≡ᵇ_
```

3.4.1 le ctaipe be le su'u mapti

```
module Nu,iorksVeritas where
  pav : ∀ {a} → {A : Set a}
      → ⦃ _ : Eq A ⦄
      → (x : List A)
      → nu,iork $ nu,iorks x
  pav 𝕃.[] = refl
  pav (x 𝕃.:: xs) = nuk {!!} $ filnek x $ nu,iorks xs
    where
    filnek : ∀ {a} → {A : Set a}
           → ⦃ _ : Eq A ⦄
           → (e : A)
           → (x : List A)
           → e ∉ 𝕃.filter (T? ∘ not ∘ _≡ᵇ_ e) x
    filnek e 𝕃.[] = refl
    filnek e (x 𝕃.:: xs) = {!!}
    nuk : ∀ {a} → {A : Set a}
        → ⦃ _ : Eq A ⦄
        → {e : A}
        → {x : List A}
        → nu,iork x
        → e ∉ x
        → nu,iork $ e 𝕃.:: x
    nuk = {!!}

  rel : ∀ {a} → {A : Set a}
      → ⦃ _ : Eq A ⦄
      → (x : List A)
      → (e : A)
      → e ∈ x → e ∈ nu,iorks x
  rel = {!!}

  cib : ∀ {a} → {A : Set a}
      → ⦃ _ : Eq A ⦄
      → (x : List A)
      → (e : A)
      → e ∈ nu,iorks x → e ∈ x
  cib = {!!}
```

3.5 la girzu

ni'o ga jo la'oi .G. du la'o zoi. map proj$_1$ $ girzu g .zoi. gi ro da poi ke'a cmima la'oi .G. zo'u ga je cmima la'oi .g. fa lo te orsi be da gi lo ve .orsi be da cu co'e

ja nilzilcmi lo'i ro cmima be la'oi .G. poi ke'a du lo te .orsi be da

```
girzu : ∀ {a} → {A : Set a}
      → ⦃ _ : Eq A ⦄
      → List A
      → List $ A × ℕ
girzu L = zipmap (length ∘ flip F L) $ nu,iorks L
  where
  zipmap = λ f x → zip x $ map f x
  F = λ a → length ∘ 𝕃.filter (_≟ a)
```

3.5.1 le ctaipe be le su'u la **girzu** cu mapti

```
module GirzuVeritas where
  pav : ∀ {a} → {A : Set a}
      → ⦃ _ : Eq A ⦄
      → (L : List A)
      → (flip LUA.All (girzu L)
           (λ (x₁ , x₂) →
             (_×_
               (x₁ ∈ L)
               (x₂ ≡_ $ length $ 𝕃.filter (_≟ x₁) L))))
  pav 𝕃.[] = LUA.[]
  pav (x 𝕃.∷ xs) = (cmimakonk x xs , refl) LUA.∷ {!!}

  rel : ∀ {a} → {A : Set a}
      → ⦃ _ : Eq A ⦄
      → (L : List A)
      → flip LUA.All L $ _∈ 𝕃.map proj₁ (girzu L)
  rel 𝕃.[] = LUA.All.[]
  rel (x 𝕃.∷ xs) = girzu₁ x xs LUA.All.∷ {!!}
    where
    girzu₁ : ∀ {a} → {A : Set a}
           → ⦃ _ : Eq A ⦄
           → (x : A)
           → (xs : List A)
           → x ∈ map proj₁ (girzu $ x 𝕃.∷ xs)
    girzu₁ = {!!}

  sum : ∀ {a} → {A : Set a}
      → ⦃ _ : Eq A ⦄
      → (L : List A)
      → 𝕃.sum (𝕃.map proj₂ $ girzu L) ≡ length L
```

```
sum 𝕃.[] = refl
sum (x 𝕃.:: xs) = {!!}
```

3.5.2 la .prefif.

ni'o xu sarcu fa lo nu ciksi bau la .lojban.

```
prefif : ∀ {a} → {A : Set a}
       → {{ _ : Eq A }}
       → (x : A)
       → (xs : List A)
       → LUA.All (_∈ (x 𝕃.:: xs)) xs
prefif x 𝕃.[] = LUA.[]
prefif e (x 𝕃.:: xs) = {!!}
```

3.6 la .refif.

ni'o xu sarcu fa lo nu ciksi bau la .lojban.

```
refif : ∀ {a} → {A : Set a}
      → {{ _ : Eq A }}
      → (x : List A)
      → LUA.All (_∈ x) x
refif 𝕃.[] = LUA.[]
refif (x 𝕃.:: z) = cmimakonk x z LUA.:: prefif x z
```

Chapter 4

le jicmu

4.1 la'oi .Multiset.

ni'o ga jo ko'a goi la'oi .x. ctaipe la'o zoi. Multiset X .zoi. gi ga je ko'a me'oi .multiset. gi lo'i ro se cmima be ko'a cu du lo'i ro se cmima be la'o zoi. Multiset.liste x .zoi.

.i lo me'oi .multiset. cu smimlu lo liste .i ku'i ro da poi ke'a me'oi .multiset. zo'u ro de poi ke'a .multiset. zo'u jitfa fa le du'u lo co'e ja se porsi cu vajni fi lo nu facki lo jei da dunli de

```
record Multiset {a} (A : Set a) : Set a
  where
  field
    liste : List A
```

4.1.1 le krinu be tu'a zo'oi .record.

ni'o la .varik. cu me'oi .record. ciksi la'oi .Multiset. jenai cu gasnu lo nu la'oi .Multiset. du la'oi .List. .i krinu la'e di'u fa le su'u la .varik. cu toldji lo nu frili fa lo nu vukna ja co'e lo ctaipe be la'o zoi. Multiset x .zoi. lo ctaipe be la'o zoi. List x .zoi.

4.1.2 le me'oi .record. co'e

la'o zoi. setoidMultiset .zoi.

```
setoidMultiset : ∀ {a} → {A : Set a} → ⦃ Eq A ⦄ → Setoid a a
setoidMultiset {A = A} = record {
  Carrier = Multiset A;
  _≈_ = Dun on GL;
```

```
  isEquivalence = record {
    refl = (λ x → x , x) $ refif _;
    sym = Data.Product.swap;
    trans = T}}
  where
  open DunVeritasJaZo'e using (T)
  GL = girzu ∘ Multiset.liste
```

4.2 la'oi .Selcmima.

ni'o la'oi .Selcmima. smimlu la'oi .Multiset. .i ku'i ga jo la'oi .*a*. ctaipe la'o zoi. Selcmima _.zoi. gi la'o zoi. Selcmima.narpanra *a* .zoi. ctaipe lo su'u ro da poi ke'a cmima ja co'e ko'a goi la'o zoi. Selcmima.liste *a* .zoi. zo'u li pa nilzilcmi lo'i ro co'e poi da du lo meirmoi be ke'a bei fo ko'a

```
record Selcmima {a} (A : Set a) ⦃ _ : Eq A ⦄ : Set a
  where
  field
    liste : List A
    narpanra : nu,iork liste
```

4.2.1 le me'oi .record. co'e

la'o zoi. setoidSelcmima .zoi.

ni'o la .varik. cu stidi lo nu tcidu le velcki fa lo na jimpe…kei je cu stidi lo nu tadni la'oi .Agda. fa lo na jimpe be fi le velcki

```
setoidSelcmima : ∀ {a} → {A : Set a} → ⦃ Eq A ⦄ → Setoid a a
setoidSelcmima {A = A} = record {
  Carrier = Selcmima A;
  _≈_ = λ a b → Al (_∈ L b) (L a) × Al (_∈ L a) (L b);
  isEquivalence = record {
    refl = λ {x} → (λ x → x , x) $ refif $ Selcmima.liste x;
    sym = Data.Product.swap;
    trans = {!!} }}
  where
  L = Selcmima.liste
  Al = LUA.All
```

4.3 la'oi .Fasnu.

ni'o ro da zo'u da ctaipe la'oi .Fasnu. jo cu fasnu

```
postulate Fasnu : Set
```

4.3.1 le me'oi .instance.

```
postulate instance eqFasnu : Eq Fasnu
```

4.4 la'oi .Selpre.

ni'o ro da zo'u da ctaipe la'oi .Selpre. jo cu selpre

```
postulate Selpre : Set
```

4.4.1 le me'oi .instance.

```
postulate instance eqSelpre : Eq Selpre
```

Chapter 5

le srana be lo lijda ja co'e

5.1 la'oi .Jdanunza'omro.

ni'o ga jo ko'a goi la'oi .*a*. ctaipe la'oi .Jdanunza'omro. gi ga je ko'a jdanuza'omro gi...

- ga je ko'a selcme lo ro cmima be la'o zoi. Jdanunza'omro.cmene *a* .zoi. gi
- krici le du'u...
  - ga je ro da zo'u da selvau la'o zoi. Selcmima.liste $ Jdanunza'omro.velski *a* .zoi. jo cu jetnu je cu velski ko'a gi
  - ko'a jdanunza'omro lo ro prenu poi ke'a zukte lo cmima be la'o zoi. Selcmima.liste $ Jdanunza'omro.krinu *a* .zoi.

```
record Jdanunza'omro : Set
  where
  field
    cmene : Selcmima String
    velski : Selcmima Fasnu
    krinu : Selcmima Fasnu
```

5.1.1 le me'oi .instance.

```
postulate instance eqJdanunza'omro : Eq Jdanunza'omro
```

5.2 la'oi .Marde.

ni'o ga naja la'oi .f. ctaipe la'oi .Marde. gi la'o zoi. f *x* .zoi. ni la'oi .*x*. vrude la'oi .f. .i ro da zo'u ga jo la'oi .f. ctaipe la'oi .Marde. gi ga jo li no du lo me'oi .f. be

da gi na'e ke co'e ja krici fi da fa lo ro prenu poi la'oi .f. du lo ro marde be ke'a

```
Marde : Set
Marde = Fasnu → ℚ
```

5.2.1 le me'oi .instance.

```
postulate instance eqMarde : Eq Marde
```

5.2.2 le me'oi .record. co'e

la'oi .setoidMarde.

```
setoidMarde : Setoid _ _
setoidMarde = record {
  Carrier = Marde;
  _≈_ = λ a b → (z : Fasnu) → a z ≡ b z;
  isEquivalence = record {
    refl = λ z → refl;
    sym = λ {a b} x z → {!!} ;
    trans = {!!} }}
```

5.3 la'oi .Prenu.

ni'o ga jo ko'a goi la'oi .*a*. ctaipe la'oi .Prenu. gi...

- ga je cmene ko'a fa lo ro cmima be la'o zoi. Selcmima.liste $ Prenu.cmene *a* .zoi. gi
- ga je la'o zoi. Prenu.marde *a* .zoi. marde ko'a gi
- la'o zoi. Prenu.selpre *a* .zoi. selpre ko'a

```
record Prenu : Set
  where
  field
    cmene : Selcmima String
    marde : Marde
    selpre : Selpre
```

5.3.1 le su'u na me'oi .field. fa lo srana be lo lijda

ni'o la .varik. cu djica ko'a goi lo nu su'o da zo'u da me'oi .field. la'oi .Prenu. je cu ctaipe ja co'e la'oi .Lijda. .i ku'i la .varik. cu na birti lo du'u ma kau zabna je su'u rinka ja co'e ko'a .i ga je le velcki be la'oi .Prenu. cu lidne le velcki be la'oi .Lijda. gi zmadu fi le ka ce'u seldji la .varik. fa lo nu lo me'oi .field. be la'oi .Lijda. cu srana lo prenu kei fe lo nu lo me'oi .field. be la'oi .Prenu. cu srana lo lijda

.i la .varik. cu djica curmi lo nu stidi

5.3.2 le me'oi .instance.

```
postulate instance eqPrenu : Eq Prenu
```

5.3.3 le me'oi .record. co'e

la'oi .setoidPrenu.

```
setoidPrenu : (λ x → Setoid x x) _
setoidPrenu = record {
  Carrier = Prenu;
  _≈_ = {!!};
  isEquivalence = {!!} }
```

5.4 la'oi .Lijda.

ni'o ga jo ko'a goi la'oi .*a*. ctaipe la'oi .Lijda. gi...

- ga je la'o zoi. Lijda.marde *a* .zoi. marde lo ro seljda be ko'a[1] gi
- ga je ga jonai la'oi .nothing. du ko'e goi la'o zoi. $\mathsf{map_m}$ Selcmima.liste $ Lijda.jdanunza'omro *a* .zoi. gi ga je su'o da zo'u da jdanunza'omro fi ko'a gi ko'e me'oi .just. lo'i jdanunza'omro be fi ko'a gi
- ga jonai...
  - la'oi .nothing. du ko'e goi la'o zoi. Lijda.cevni *a* .zoi. gi
  - ga je selcei fa lo ro seljda be ko'a gi...
    - ga je ko'e me'oi .just. lo'i cevni be ko'a gi

[1] .i jitfa fa le du'u ro da zo'u da srana lo marde be da be'o jo lo marde be lo ro seljda be lo lijda be da .i su'o da zo'u su'o de zo'u ga je de pagbu lo marde be da gi su'o di poi ke'a seljda lo lijda be da zo'u de pagbu lo marde be da be'o jenai lo marde be di

* ga jo la'oi .t. ctaipe la'o zoi. Is-just $ Lijda.cevni a .zoi. gi ga jo la'oi .Z. du la'o zoi. Data.Maybe.to-witness t .zoi. gi ga jo la'oi .G. du la'o zoi. proj_1 $ proj_1 Z .zoi. gi lo mu'oi zoi. (proj_2 Z) m n .zoi. be ko'a goi la'o zoi. G ! m .zoi. bei ko'e goi la'o zoi. G ! n .zoi. cu co'e ja ni ko'a nelci ko'e

```
record Lijda : Set
  where
  private
    𝔽L : ∀ {a b} → {A : Set a} → {B : A → Set b}
       → ⦃ LL A ⦄
       → Σ A B → Set
    𝔽L = Fin ∘ length ∘ proj₁
  field
    cevni : Maybe $ Σ (UL $ List Prenu) $ (λ X → X → X → ℚ) ∘ 𝔽L
    marde : Marde
    jdanunza'omro : Maybe $ Selcmima Jdanunza'omro
```

5.4.1 le me'oi .record. co'e

```
setoidLijda : (λ x → Setoid x x) _
setoidLijda = record {
  Carrier = Lijda;
  _≈_ = λ a b → Lijda.marde a ≡ Lijda.marde b
              × Setoid._≈_ SLJ (LJ a) (LJ b)
              × Setoid._≈_ SLC (Lijda.cevni a) (Lijda.cevni b);
  isEquivalence = {!!} }
  where
  LJ = Lijda.jdanunza'omro
  SLJ : (λ x → Setoid x x) _
  SLJ = record {
    Carrier = Maybe $ Selcmima Jdanunza'omro;
    _≈_ = SLJdu;
    isEquivalence = {!!} }
    where
    SLJdu : ∀ {a} → {A : Set a}
          → ⦃ _ : Eq A ⦄
          → Maybe $ Selcmima A
          → Maybe $ Selcmima A
          → Set a
    SLJdu a@nothing b@nothing = a ≡ b
    SLJdu (just a) (just b) = Setoid._≈_ setoidSelcmima a b
    SLJdu _ _ = ⊥
  SLC : (λ x → Setoid x x) _
```

```
SLC = record {
  Carrier = _;
  _≈_ = {!!};
  isEquivalence = {!!} }
```