Scope

```
{-# OPTIONS --safe --without-K --exact-split #-}

open import MLTT.Spartan
open import MLTT.Negation
open import MLTT.Plus
open import UF.FunExt
open import UF.Univalence
open import UF.Equiv
open import MLTT.List
open import UF.Subsingletons
open import Naturals.Order
open import UF.Subsingletons-FunExt
open import UF.PropTrunc
open import UF.Sets
open import UF.Base
import UF.ImageAndSurjection

open import Lists

module Scope (fe : Fun-Ext) (pt : propositional-truncations-exist) (Msg : 𝓤 ̇)
(Secret : 𝓤 ̇ ) where

open PropositionalTruncation pt
open UF.ImageAndSurjection pt

open import PredP
open Pred
open ΣPred
open import Definitions Msg Secret

restr : ∀{𝓤 𝓥} → {A : 𝓤 ̇} → (P : A → 𝓥 ̇) → Σ P → A
restr P x =  x .pr₁

_$₂_ : ∀{𝓤 𝓥} → {A : 𝓤 ̇} → {B : 𝓥 ̇} → (A → B) → A × A → B × B
f $₂ (a , b) = f a , f b

+→𝟚 : ∀{𝓤 𝓥} → {X : 𝓤 ̇} → {Y : 𝓥 ̇} → X + Y → 𝟚
+→𝟚 (inl x) = ₀
+→𝟚 (inr x) = ₁

scope-l1 : (x : Secret) → (ls : List Secret) → (A : 𝟚 → 𝓦 ̇)
         → is-decidable (x ∈ ls) → 𝓦 ̇
scope-l1 x ls A r = A (+→𝟚 r)


module BSet-scope (_∈?_ : ∀ s ls → is-decidable (s ∈ ls)) where

 Lim : 𝓥 ̇ → 𝟚 → Set 𝓥
 Lim P ₀ = 𝟘
 Lim P ₁ = P

 limitPr : Secret → 𝓥 ̇ → Pred S×Msg 𝓥
 limitPr s P mp@(ls , msg) = scope-l1 s ls (Lim P) (s ∈? ls)

 limit : Secret → BSet 𝓥 → BSet 𝓥
```

```
 limit s bs .pr₁ mp = limitPr s (< bs > mp) mp
 limit s bs .pr₂ = λ ascrs scrs x (a⊂s , a⊃s) → l1 ascrs scrs x a⊂s a⊃s (s ∈? ascrs)
(s ∈? scrs) , l2 ascrs scrs x a⊂s a⊃s (s ∈? scrs) (s ∈? ascrs) where
   l1 : ∀ ascrs scrs x a⊃s a⊂s → (deq : is-decidable (s ∈ ascrs)) → (deq2 : is-
decidable (s ∈  scrs)) → scope-l1 s ascrs (Lim (< bs > (ascrs , x))) deq → scope-l1 s
scrs (Lim (< bs > (scrs , x))) deq2
   l1 ascrs scrs x a⊃s a⊂s (inr neq) (inl eq2) cond = 𝟘-elim (neq (∈→∈ s scrs ascrs
a⊂s eq2))
   l1 ascrs scrs x a⊃s a⊂s (inr neq) (inr x₁) cond = bs .pr₂ ascrs scrs x (a⊃s ,
a⊂s) .pr₁ cond

   l2 : ∀ ascrs scrs x a⊃s a⊂s → (deq : is-decidable (s ∈ scrs)) → (deq2 : is-
decidable (s ∈ ascrs)) → scope-l1 s scrs (Lim (< bs > (scrs , x))) deq → scope-l1 s
ascrs (Lim (< bs > (ascrs , x))) deq2
   l2 ascrs scrs x a⊃s a⊂s (inr neq) (inl eq2) cond = 𝟘-elim (neq (∈→∈ s ascrs scrs
a⊃s eq2))
   l2 ascrs scrs x a⊃s a⊂s (inr neq) (inr x₁) cond = bs .pr₂ ascrs scrs x (a⊃s ,
a⊂s) .pr₂ cond

 limitMPr : Secret → List Secret → 𝒱 ˙ → Pred S×Msg 𝒱
 limitMPr s [] bs mp = limitPr s bs mp
 limitMPr s (l ∷ ls) w mp = let w2 = limitPr s w mp
                               w3 = limitMPr l ls w2 mp
                           in w3

 limitPr-𝟘 : ∀ s mp → limitPr {𝒱} s 𝟘 mp ＝ 𝟘
 limitPr-𝟘 s  mp@(scr , _) with (s ∈? scr)
 ... | inl x = refl
 ... | inr x = refl

 limitMPr-𝟘 : ∀ s ls mp → limitMPr {𝒱} s ls 𝟘 mp ＝ 𝟘
 limitMPr-𝟘 s [] mp@(scr , _) = limitPr-𝟘 s mp
 limitMPr-𝟘 s (l ∷ ls) mp = ap (λ z → limitMPr l ls z mp) (limitPr-𝟘 s mp) ∙
limitMPr-𝟘 l ls mp

 limitM : Secret → List Secret → BSet 𝒱 → BSet 𝒱
 limitM s ls bs .pr₁ mp = limitMPr s ls (< bs > mp) mp
 limitM s [] bs .pr₂ = limit s bs .pr₂
 limitM s (l ∷ ls) bs .pr₂ = limitM l ls (limit s bs) .pr₂

 limitM' : List Secret → BSet 𝒱 → BSet 𝒱
 limitM' [] bs = bs
 limitM' (s ∷ ls) bs = limitM s ls bs

-- limitM is a restriction, so it fits where bs fits.
 lim-rec : ∀{𝓦} → {A : 𝓦 ˙} → ∀ s ls {bs mp} → < (limitM {𝒱} s ls bs) > mp → (< bs >
mp → A) → A
 lim-rec s [] {bs} {mp@(ws , msg)} c f = l1 (s ∈? ws) c where
  l1 : (w : (s ∈ ws) + (s ∈ ws → 𝟘)) →
       Lim (< bs > (ws , msg)) (+→𝟚 w) → _
  l1 (inr _) c = f c

 lim-rec {𝒱 = 𝒱} s (l ∷ ls) {bs} {mp@(ws , msg)} c f = l1 (s ∈? ws) c where
  l1 : (w : (s ∈ ws) + (s ∈ ws → 𝟘)) →
       limitMPr l ls (Lim (< bs > (ws , msg)) (+→𝟚 w)) (ws , msg) → _
  l1 (inl x) c with limitMPr {𝒱} l ls 𝟘 mp | (limitMPr-𝟘 {𝒱} l ls mp)
```

```
  l1 (inl x) () | r | refl
  l1 (inr x) c = lim-rec l ls {bs} {mp} c f


 lim-rec' : ∀{𝑾} → {A : 𝑾 ̇} → ∀ ls bs {mp} → < (limitM' {𝓥} ls bs) > mp → (< bs > mp
→ A) → A
 lim-rec' [] _ c f = f c
 lim-rec' (x ∷ ls) bs {mp} = lim-rec x ls {bs}


 module &PSet-scope {𝓥} where

  limit&P : Secret → &PSet 𝓥 𝑾 → &PSet 𝓥 (𝓤 ⊔ 𝓥 ⁺ ⊔ 𝑾)
  limit&P s ps .pr₁ v = v ∈image λ x → (λ (a , bs) → a , limit s bs) (restr < ps > x)
  limit&P s ps .pr₂ = cons-is-non-empty

  limit&PM : Secret → List Secret → &PSet 𝓥 𝑾 → &PSet 𝓥 (𝓤 ⊔ 𝓥 ⁺ ⊔ 𝑾)
  limit&PM s ls ps .pr₁ v = v ∈image λ x → (λ (a , bs) → a , limitM s ls bs) (restr <
ps > x)
  limit&PM s ls ps .pr₂ = cons-is-non-empty
```