

# Seminar 4 - From the theory of vulnerability to verified policy advice

Adrian Perez Keilty

Chalmers University of Technology, Göteborg, Sweden

Note: Check out the attached file

`from-vulnerability-to-verified-policy-advice.agda`

for type-checking.

## Exercise 4.1:

Let  $x : \mathbb{R} \rightarrow \mathbb{R}$ . What are the types of  $\dot{x}, f, \varphi$  in the expressions above?

$$\dot{x} : \mathbb{R} \rightarrow \mathbb{R}$$

$$\varphi : \mathbb{R} \rightarrow \mathbb{R}^2 \rightarrow \mathbb{R}^2$$

$$f : (\mathbb{R} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow \mathbb{R}$$

## Exercise 4.2:

Which function is  $\varphi 0$ ? Which function is  $\varphi (t_1 + t_2)$ ?

$$\varphi 0 (t_0, x_0) = (t_0 + 0, x(t_0 + 0)) = (t_0, x_0) \implies \varphi 0 \equiv id$$

$$\varphi (t_1 + t_2) = \varphi t_2 \circ \varphi t_1$$

## Exercise 4.3:

What is the type of  $\hat{\varphi} \Delta t k$ ?

$$\hat{\varphi} : \mathbb{R}_+ \rightarrow \mathbb{N} \rightarrow \mathbb{R}^2$$

## Exercise 4.4:

Let

$$\text{next} : \text{State} \rightarrow \text{State}$$

and

$$\text{Evolution} = \text{Vec State } 5.$$

Define

$$\text{possible} : \text{State} \rightarrow \text{Evolution}$$

such that  $\text{possible } s$  is the trajectory under  $\text{next}$  starting in  $s$ :

$$\text{possible } s = [s, \text{next } s, \dots, \text{next}^{(4)} s].$$

Given the functions

```

-- deterministic system
DetSys : Set → Set
DetSys X = X → X

-- deterministic evolution, iterating over a deterministic system
detFlow : {X : Set } → DetSys X → Nat → DetSys X
detFlow f zero = id
detFlow f (suc n) = detFlow f n ∘ f

```

we can define

```

postulate next1 : State -> State
possible1 : State -> Vec State 5
possible1 s = map (λ n -> detFlow next1 n s)
              (0 :: 1 :: 2 :: 3 :: 4 :: [])

```

## Exercises 4.5 and 4.6:

Encode the mathematical specification

$$\forall m, n \in N, \forall f: \text{DetSys } X, \forall x \in X,$$

$$\text{detFlow } f \ (m + n) \ x = \text{detFlow } f \ n \ (\text{detFlow } f \ m \ x)$$

in Agda through a function `detFlowP1` and implement (prove) `detFlowP1` by induction on `m`:

```

-----
detFlowP1 : {X : Set} (f : DetSys X) (m n : Nat) (x : X) →
  detFlow f (m + n) x ≡ detFlow f n (detFlow f m x)
detFlowP1 f zero n x =
  begin
    detFlow f (zero + n) x
  =⟨⟩
    detFlow f n x
  =⟨⟩ -- apply id
    detFlow f n (id x)
  =⟨⟩ -- apply detFlow first clause
    detFlow f n (detFlow f zero x)
  end
detFlowP1 f (suc m) n x =
  begin
    detFlow f (suc m + n) x
  =⟨⟩ -- suc apply def
    detFlow f (suc (m + n)) x
  =⟨⟩ -- detFlow apply second clause
    detFlow f (m + n) (f x)
  =⟨ detFlowP1 f m n (f x) ⟩ -- induction hypothesis
    detFlow f n (detFlow f m (f x))
  =⟨⟩ -- undo detFlow second clause

```

```

    detFlow f n (detFlow f (suc m) x)
  end
-----

```

### Exercise 4.7:

`detTrj` fulfills a specification similar to `detFlowP1`. Encode this specification in the type of a function `detTrjP1` using only `detTrj`, `detFlow`, `tail` : `Vec X (1 + n) → Vec X n` and vector concatenation `++`:

```

postulate detTrjP1 : {X : Set} (f : DetSys X) (m n : Nat) (x : X) →
detTrj f (m + n) x ≡ detTrj f m x ++ tail (detTrj f n (detFlow f m x))

```

### Exercise 4.8:

Implementation of `detFlowTrjP1` (type-checks without `lastLemma`):

```

-----
detFlowTrjP1 : {X : Set} → (n : Nat) → (f : DetSys X) →
              (x : X) → last (detTrj f n x) ≡ detFlow f n x
detFlowTrjP1 zero f x =
  begin
    last (detTrj f zero x)
  =<> -- detTrj first clause
    last (x :: [])
  =<> -- last def
    x
  =<> -- apply id
    id x
  =<> -- detFlow first clause
    detFlow f zero x
  end
detFlowTrjP1 (suc n) f x =
  begin
    last (detTrj f (suc n) x)
  =<>
    last (detTrj f n (f x))
  =< detFlowTrjP1 n f (f x) > -- induction step
    detFlow f n (f x)
  =<>
    detFlow f (suc n) x
  end
-----

```

### Exercises 4.9 and 4.10:

What are the types of  $\eta_{List}$  and  $\Rightarrow_{List}$  in the definition of `nonDetFlow`?

Since we have

```

NonDetSys : Set → Set
NonDetSys X = X → List X,

```

then according to the definitions

```

nonDetFlow : {X : Set} → NonDetSys X → Nat → NonDetSys X
nonDetFlow f zero = ηList
nonDetFlow f (suc n) = f >=>List nonDetFlow f n

```

$\eta_{List}$  must be of type List and

```

_>=>List_ : {A B C : Set} → (A → List B) → (B → List C) → (A → List C)
f >=>List g = μList ∘ ((fmapList g) ∘ f)

```

where  $\mu_{List}$  is list concatenation and  $fmap_{List}$  is the pointwise function map for lists.

### Exercise 4.11:

The following equality type-checks:

```

ηList NatTrans : {A B : Set} → (f : A → B) → (a : A) →
    fmapList f (ηList a) ≡ ηList (f a)
ηList NatTrans f a = refl

```

### Exercise 4.12:

Compute `nonDetFlow rw n zero` and `nonDetTrj rw n zero` for  $n = 0, 1, 2$  for the random walk

```

rw : N → List N
rw zero
= zero :: suc zero :: []
rw (suc m) = m :: suc m :: suc (suc m) :: []

```

Using (ctrl+c+n) to normalize we get:

```

-----
nonDetFlow rw 0 zero = 0 :: []
nonDetFlow rw 1 zero = 0 :: 1 :: []
nonDetFlow rw 2 zero = 0 :: 1 :: 0 :: 1 :: 2 :: []
-----

```

and

```

-----
nonDetTrj rw 0 zero = (0 :: []) :: []
nonDetTrj rw 1 zero = (0 :: 0 :: []) :: (0 :: 1 :: []) :: []
nonDetTrj rw 2 zero = (0 :: 0 :: 0 :: []) ::
    (0 :: 0 :: 1 :: []) ::
    (0 :: 1 :: 0 :: []) :: (0 :: 1 :: 1 :: []) ::
    (0 :: 1 :: 2 :: []) :: []
-----

```

### Exercise 4.13:

Show that  $\text{Det} \equiv \text{NonDet}$  by induction on  $n$  and using  $\eta_{\text{List}}$   $\text{NatTrans}$  and postulate `triangleLeftList`:

```
-----
detToNonDet : {X : Set} → DetSys X → NonDetSys X
detToNonDet f =  $\eta_{\text{List}}$  ∘ f

postulate triangleLeftList : {A : Set} → (as : List A) →  $\mu_{\text{List}}$  ( $\eta_{\text{List}}$  as)  $\equiv$  as

Det≡NonDet : {X : Set} → (f : DetSys X) → (n : Nat) → (x : X) →
   $\eta_{\text{List}}$  (detFlow f n x)  $\equiv$  nonDetFlow (detToNonDet f) n x
Det≡NonDet f zero x =
  begin
     $\eta_{\text{List}}$  (detFlow f zero x)
  =⟨⟩
     $\eta_{\text{List}}$  x
  =⟨⟩
    nonDetFlow (detToNonDet f) zero x
  end
Det≡NonDet f (suc n) x =
  begin
     $\eta_{\text{List}}$  (detFlow f (suc n) x)
  =⟨⟩
     $\eta_{\text{List}}$  ((detFlow f n ∘ f) x)
  =⟨⟩
     $\eta_{\text{List}}$  (detFlow f n (f x))
  =⟨ Det≡NonDet f n (f x) ⟩
    nonDetFlow (detToNonDet f) n (f x)
  =⟨ triangleLeftList2 (nonDetFlow (detToNonDet f) n (f x)) ⟩
     $\mu_{\text{List}}$  ( $\eta_{\text{List}}$  (nonDetFlow (detToNonDet f) n) (f x)))
  =⟨ --  $\eta_{\text{ListNatTrans}}$  : fmapList f ( $\eta_{\text{List}}$  a)  $\equiv$   $\eta_{\text{List}}$  (f a)
     $\mu_{\text{List}}$  (fmapList (nonDetFlow (detToNonDet f) n) ( $\eta_{\text{List}}$  (f x)))
  =⟨⟩
    (nonDetFlow (detToNonDet f) (suc n)) x
  end
-----
```

### Exercise 4.14:

Postulate the monadic laws in Agda and generalize the results to monads:

```
-----
postulate M      : Set → Set
postulate fmapM  : {A B : Set } → (A → B) → M A → M B
postulate  $\eta_M$     : {A : Set } → A → M A
postulate  $\mu_M$    : {A : Set } → M (M A) → M A

infixl 40 _>=>M_
```

```

_>=>M_ : { B C : Set } → M B → (B → M C) → M C
mb >=>M f = μM (fmapM f mb)

infixl 50 _>=>M_
_>=>M_ : {A B C : Set } → (A → M B) → (B → M C) → (A → M C)
f >=>M g = (λ a → (f a) >=>M g)

postulate leftTriangle      : {A : Set}      → (ma : M A)
→ ma ≡ (μM ∘ ηM) ma -- used in Det≡Mon proof

postulate lawTriangle      : {A : Set}      → (ma : M A)
→ (μM ∘ ηM) ma ≡ (μM ∘ fmapM ηM) ma

postulate lawRectangle1 : {A : Set}      → (ma : M (M (M A)))
→ μM (μM ma) ≡ μM (fmapM μM ma)

postulate lawRectangle2 : {X Y : Set}      → (x : X)      → (f : X → Y)
→ (ηM ∘ f) x ≡ ((fmapM f) ∘ ηM) x

postulate lawRectangle3 : {X Y : Set}      → (mx : M (M X)) → (f : X → Y)
→ μM (fmapM (fmapM f) mx) ≡ fmapM f (μM mx)

postulate lawBow          : {A B C : Set} → (a : A)
→ (f : A → M B) → (g : B → M C)
→ μM (fmapM g (f a)) ≡ (f >=>M g) a

postulate ηMNatTrans      : {X : Set} → (f : X → M X) → (x : X)
→ ηM (f x) ≡ fmapM f (ηM x)

postulate ηMNatTrans'     : {X : Set}      → (f : X → M X)      → (x : X)
→ μM (ηM (f x)) ≡ μM (fmapM f (ηM x)) -- used in Det≡Mon proof
-----

```

## Exercise 4.15:

Using the postulated monadic laws, prove Det≡Mon

```

-----
Det≡Mon : {X : Set} → (f : DetSys X) → (n : Nat) → (x : X) →
    ηM (detFlow f n x) ≡ monFlow (detToMon f) n x
Det≡Mon f zero x = -- refl
    begin
        ηM (detFlow f zero x)
    =⟨⟩
        ηM x
    =⟨⟩
        monFlow (detToMon f) zero x
    end
Det≡Mon f (suc n) x =

```

```

begin
   $\eta_M$  (detFlow f (suc n) x)
=⟨
   $\eta_M$  ((detFlow f n  $\circ$  f) x)
=⟨
   $\eta_M$  (detFlow f n (f x))
=⟨
  ( $\eta_M$  (detFlow f n (f x)))
=⟨ Det $\equiv$ Mon f n (f x) > -- induction hypothesis
  (monFlow (detToMon f) n) (f x)
=⟨ leftTriangle ((monFlow (detToMon f) n) (f x)) >
   $\mu_M$  ( $\eta_M$  ((monFlow (detToMon f) n) (f x))))
=⟨  $\eta_{M\text{NatTrans}}$  ' (monFlow (detToMon f) n) (f x) >
  --  $\mu_M$  ( $\eta_M$  (f x))  $\equiv$   $\mu_M$  (fmapM f ( $\eta_M$  x))
   $\mu_M$  (fmapM (monFlow (detToMon f) n) ( $\eta_M$  (f x)))
=⟨
  ((detToMon f)  $\Rightarrow_M$  monFlow (detToMon f) n) x
=⟨
  monFlow (detToMon f) (suc n) x
end

```

---