From the theory of vulnerability to verified policy advice

Nicola Botta[1]

Potsdam Institute for climate impact research

Part 1, 2024-04-22

[1] Joint work with P Jansson, C Ionescu

Where we are

Where we are

First half (mostly done):

15 Vulnerability modelling with functional programming and dependent types

16 Testing versus proving in climate impact research

17 Dependently-Typed Programming in Scientific Computing - Examples from Economic Modelling

18 Towards a Computational Theory of GSS: a Case for Domain-Specific Languages

Second half (upcoming):

19 Sequential decision problems, dependent types and generic solutions

20 Contributions to a computational theory of policy advice and avoidability

21 The impact of uncertainty on optimal emission policies

22 Responsibility Under Uncertainty: Which Climate Decisions Matter Most?

Plan

Plan

Today:

- The computational structure of *possible*: Monadic dynamical systems

Next week (18 or 19):

- Background: climate science, climate policy under uncertainty

Week 19 or 20:

- Sequential decision problems
- Bellman's equation, backward induction
- Verified policy advice in a nutshell

The computational structure of *possible*: Monadic dynamical systems

The computational structure of *possible*: Monadic dynamical systems

- Recap vulnerability theory
- *State*, *Evolution* and deterministic systems
- Non-deterministic systems
- Monadic systems

Recap vulnerability theory

Recap vulnerability theory

$$
\begin{array}{lll}
\textbf{postulate}\ \mathit{State}\ \mathit{Evolution}\ V\ W & : & \mathit{Set} \\
\textbf{postulate}\ F & : & \mathit{Set} \to \mathit{Set} \\
\textbf{postulate}\ \mathit{fmap} & : & \{A\ B : \mathit{Set}\} \to (A \to B) \to F\ A \to F\ B \\
\textbf{postulate}\ \mathit{possible} & : & \mathit{State} \to F\ \mathit{Evolution} \\
\textbf{postulate}\ \mathit{harm} & : & \mathit{Evolution} \to V \\
\textbf{postulate}\ \mathit{measure} & : & F\ V \to W
\end{array}
$$

$$
\begin{array}{lll}
\mathit{vulnerability} & : & \mathit{State} \to W \\
\mathit{vulnerability} & = & \mathit{measure} \circ \mathit{fmap}\ \mathit{harm} \circ \mathit{possible}
\end{array}
$$

- The theory is applied (instantiated) by defining *State*, *Evolution*, etc.

State, *Evolution* and deterministic systems

State, *Evolution* and deterministic systems

- Values of type *State* represent the state of a system
- Example 1: a reduced climate system as in SURFER
- Example 2: a simplified climate-economy system as in DICE simplified
- Example 3: a detailed climate system like in EMICs, GCMs, etc.
- $possible\ s : F\ Evolution$ are the possible evolutions starting in $s : State$
- Thus, either *State* or *possible* have to entail some representation of both natural and anthropogenic forcing on the system, for example global GHG emissions

State, *Evolution* and deterministic systems

- Remember that *Evolution* is the type of evolutions or scenarios
- In a deterministic, time continuous setting, evolutions are certain
- Often, they can be described by differential equations, for example ODE

$$\dot{x}\ t = f\ (x\ t) = (f \circ x)\ t$$

- In this case $F = Id$ and the evolution starting in (t_0, x_0) is obtained by integration:

$$\varphi\ t\ (t_0, x_0) = (t_0 + \int_{t_0}^{t_0+t} d\tau,\ x_0 + \int_{t_0}^{t_0+t} f\ (x\ \tau)\ d\tau) = (t_0 + t,\ x\ (t_0 + t))$$

Exercise 4.1

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

Exercise 4.2

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

- The evolution of time continuous, deterministic systems on time discretizations $\hat{t} : \mathbb{R}_+ \to \mathbb{N} \to \mathbb{R}_+$, $\hat{t}\ \Delta t\ k = k * \Delta t$ is also described by endo-functions

$$\hat{\varphi}\ \Delta t\ k = \varphi\ (\hat{t}\ \Delta t\ k)$$

Exercise 4.3

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

- ...and also that of time time discrete deterministic systems, e.g., given by difference equations

$$x\ (t+1) = x\ t + g\ (x\ t, x\ (t+1))$$

State, *Evolution* and deterministic systems

- In general, one can model deterministic systems as endo-functions

$$
\begin{aligned}
&DetSys &&: \quad Set \to Set \\
&DetSys\ X &&= \quad X \to X
\end{aligned}
$$

- The evolutions of a system is then obtained by iterating that system:

$$
\begin{aligned}
&detFlow &&: \{X : Set\} \to DetSys\ X \to \mathbb{N} \to DetSys\ X \\
&detFlow\ f\ \ zero &&= id \\
&detFlow\ f\ (suc\ n) &&= detFlow\ f\ n \circ f
\end{aligned}
$$

Exercise 4.4

Let $next : State \to State$ and $Evolution = Vec\ State\ 5$. Define $possible : State \to Evolution$ such that $possible\ s$ is the trajectory under $next$ starting in s: $possible\ s = [s, next\ s, ..., next^4\ s]$.

State, *Evolution* and deterministic systems

Exercise 4.5

Encode the mathematical specification

$$\forall\ m, n \in \mathbb{N}.\ \forall\ f : \mathit{DetSys}\ X.\ \forall\ x \in X.\ \mathit{detFlow}\ f\ (m + n)\ x = \mathit{detFlow}\ f\ n\ (\mathit{detFlow}\ f\ m\ x)$$

in Agda through a function *detFlowP1*.

Exercise 4.6

Implement (prove) *detFlowP1* by induction on m.

- One can compute the trajectory obtained by iterating a system n times from an initial state:

$$\begin{aligned}
&detTrj : \{X : Set\} \to DetSys\ X \to (n : \mathbb{N}) \to X \to Vec\ X\ (suc\ n)\\
&detTrj\ f\ \ zero\ \ \ \ x = x :: []\\
&detTrj\ f\ (suc\ n)\ x = x :: detTrj\ f\ n\ (f\ x)
\end{aligned}$$

Exercise 4.7

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

State, *Evolution* and deterministic systems

- Perhaps not surprisingly, the last element of the trajectory of length $1 + n$ of $f : DetSys\ X$ starting in x is just $detFlow\ f\ n\ x$:

$$
\begin{aligned}
detFlowTrjP1 &: \{X : Set\} \to (n : \mathbb{N}) \to (f : DetSys\ X) \to \\
&\qquad (x : X) \to last\ (detTrj\ f\ n\ x) \equiv detFlow\ f\ n\ x
\end{aligned}
$$

Exercise 4.8

Implement *detFlowTrjP1* using

$$
\begin{aligned}
&lastLemma : \{A : Set\} \to \{n : \mathbb{N}\} \to \\
&\qquad (a : A) \to (as : Vec\ A\ (suc\ n)) \to last\ (a :: as) \equiv last\ as \\
&lastLemma\ a\ (x :: as) = refl
\end{aligned}
$$

Non-deterministic systems

Non-deterministic systems

- Remember that uncertainty can be represented functorially: $possible : State \to F\ Evolution$
- For $F = List$, we have non-deterministic uncertainty
- In this case, for a given initial state one can have zero, one or more possible next states
- One can iterate non-deterministic systems like deterministic ones

$$NonDetSys : Set \to Set$$
$$NonDetSys\ X = X \to List\ X$$

$$nonDetFlow : \{X : Set\} \to NonDetSys\ X \to \mathbb{N} \to NonDetSys\ X$$
$$nonDetFlow\ f\ \ zero \quad = \eta_{List}$$
$$nonDetFlow\ f\ (suc\ n) = f \ggg_{List} nonDetFlow\ f\ n$$

Non-deterministic systems

Exercise 4.9

What are the types of η_{List} and $>\!\!=\!\!>_{List}$ in the definition of *nonDetFlow*?

Exercise 4.10

Define $>\!\!=\!\!>_{List}$ in terms of $fmap_{List}$ and μ_{List} with

$$\begin{aligned} fmap_{List} &: \{A\ B : Set\} \to (A \to B) \to List\ A \to List\ B \\ \mu_{List} &: \{A : Set\} \to List\ (List\ A) \to List\ A \end{aligned}$$

Exercise 4.11

Verify that, for arbitrary types A and B, $\eta_{List} = [\_]$ and $fmap_{List} = map$ fulfill

$$(f : A \to B) \to (a : A) \to fmap_{List}\ f\ (\eta_{List}\ a) \equiv \eta_{List}\ (f\ a)$$

Non-deterministic systems

- With $fmap_{List}$, η_{List} and $\gg\!\!=\!\!>_{List}$, one can also compute all the possible trajectories

$$
\begin{aligned}
&nonDetTrj : \{X : Set\} \to NonDetSys\ X \to (n : \mathbb{N}) \to X \to List\ (Vec\ X\ (suc\ n))\\
&nonDetTrj\ f\ zero\quad x = fmap_{List}\ (x :: \_)\ (\eta_{List}\ [])\\
&nonDetTrj\ f\ (suc\ n)\ x = fmap_{List}\ (x :: \_)\ ((f \gg\!\!=\!\!>_{List} (nonDetTrj\ f\ n))\ x)
\end{aligned}
$$

Exercise 4.12

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

$$
\begin{aligned}
&rw : \mathbb{N} \to List\ \mathbb{N}\\
&rw\ zero \quad = zero :: suc\ zero :: []\\
&rw\ (suc\ m) = m :: suc\ m :: suc\ (suc\ m) :: []
\end{aligned}
$$

Non-deterministic systems

- Every deterministic system can be represented by a non-deterministic one:

$$detToNonDet : \{X : Set\} \to DetSys\ X \to NonDetSys\ X$$
$$detToNonDet\ f = \eta_{List} \circ f$$

Exercise 4.13

Show that

$$Det{\equiv}NonDet : \{X : Set\} \to (f : DetSys\ X) \to (n : \mathbb{N}) \to (x : X) \to$$
$$\eta_{List}\ (detFlow\ f\ n\ x) \equiv nonDetFlow\ (detToNonDet\ f)\ n\ x$$

by induction on n and using $\eta_{List} NatTrans$ and

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

Non-deterministic systems

- Perhaps surprisingly, the opposite is also true

$$nonDetToDet \;:\; \{X \;:\; Set\} \to NonDetSys\; X \to DetSys\;(List\; X)$$
$$nonDetToDet\; f \;=\; \mu_{List} \circ (fmap_{List}\; f)$$

- But the state of the resulting deterministic system is now much bigger!
- The function $\lambda\; xs \;\to\; \lambda\; f \;\to\; \mu_{List} \circ (fmap_{List}\; f\; xs)$ is usually denoted by the infix $\gg\!=_{List}$

$$nonDetToDet\; f\; xs \;=\; xs \gg\!=_{List}\; f$$

- Again, one has a representation theorem

$$NonDet{\equiv}Det \;:\; \{X \;:\; Set\} \to (f \;:\; NonDetSys\; X) \to (n \;:\; \mathbb{N}) \to (xs \;:\; List\; X) \to$$
$$nonDetToDet\;(nonDetFlow\; f\; n)\; xs \;\equiv\; detFlow\;(nonDetToDet\; f)\; n\; xs$$

Monadic systems