From the theory of vulnerability to verified policy advice

Nicola Botta with P. Jansson and C. Ionescu

Part 1, 2024-04-22

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

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

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

- 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?

State, *Evolution* and deterministic systems

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

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

- 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.

State, *Evolution* and deterministic systems

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

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

Exercise 4.7

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

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

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

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 $\ggg\!\!=_{List}$, one can also compute all the possible trajectories

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

Exercise 4.12

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

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

Non-deterministic systems

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

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

Exercise 4.13

Show that

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

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

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

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

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

From the theory of vulnerability to verified policy advice

Nicola Botta with P. Jansson and C. Ionescu

Part 1, The computational structure of *possible*: Monadic dynamical systems, 2024-04-29

Plan

Done:

- The computational structure of *possible*: Monadic dynamical systems
  - Recap vulnerability theory
  - *State*, *Evolution* and deterministic systems
  - Non-deterministic systems

Today:

- Monadic systems
- Background: climate science, climate policy under uncertainty

Week 19:

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

Recap computational structure of *possible*

$$DetSys : Set \to Set$$
$$DetSys\ X = X \to X$$

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

$$detFlow\ f\ zero = id$$
$$detFlow\ f\ (suc\ n) = detFlow\ f\ n \circ f$$

$$nonDetFlow\ f\ zero = \eta_{List}$$
$$nonDetFlow\ f\ (suc\ n) = f >\!\!=\!\!>_{List} nonDetFlow\ f\ n$$

$$detTrj\ f\ zero\ x = x :: []$$
$$detTrj\ f\ (suc\ n)\ x = x :: detTrj\ f\ n\ (f\ x)$$

$$nonDetTrj\ f\ zero\ x = fmap_{List}\ (x ::\_)\ (\eta_{List}\ [])$$
$$nonDetTrj\ f\ (suc\ n)\ x = fmap_{List}\ (x ::\_)\ ((f >\!\!=\!\!>_{List} (nonDetTrj\ f\ n))\ x)$$

Recap computational structure of *possible*

$$
\begin{aligned}
&\eta_{List} &&: \\
&{>\!\!=\!\!>}_{List} &&: \\
&fmap_{List} &&: (A \to B) \to List\ A \to List\ B \\
&\mu_{List} &&: List\ (List\ A) \to List\ A \\
&{>\!\!>\!\!=}_{List} &&:
\end{aligned}
$$

$$
\begin{aligned}
\eta_{List}\ x &= [x] \\
fmap_{List} &= map \\
nonDetToDet\ f\ xs &= xs\ {>\!\!>\!\!=}_{List}\ f
\end{aligned}
$$

$$
\begin{aligned}
&\forall\ (f : A \to B)\ (a : A) \to && fmap_{List}\ f\ (\eta_{List}\ a) \equiv \eta_{List}\ (f\ a) \\
&\forall\ (as : List\ A) \to && \mu_{List}\ (\eta_{List}\ as) \equiv as
\end{aligned}
$$

Monadic systems

Monadic systems

- Deterministic, non-deterministic, stochastic, etc. systems are instances of monadic systems

$$
\begin{array}{lll}
M & : & Set \to Set \\
fmap_M & : & \{A\ B : Set\} \to (A \to B) \to M\ A \to M\ B \\
\eta_M & : & \{A : Set\} \to A \to M\ A \\
\mu_M & : & \{A : Set\} \to M\ (M\ A) \to M\ A
\end{array}
$$

$$
\begin{array}{lll}
\_\gg=_M\_ & : & \{B\ C : Set\} \to M\ B \to (B \to M\ C) \to M\ C \\
\_>=>_M\_ & : & \{A\ B\ C : Set\} \to (A \to M\ B) \to (B \to M\ C) \to (A \to M\ C)
\end{array}
$$

$$
\begin{array}{lll}
mb \gg=_M f & = & \mu_M\ (fmap_M\ f\ mb) \\
f >=>_M g & = & \lambda\ a \to (f\ a) \gg=_M g
\end{array}
$$

$$
\begin{array}{lll}
MonSys & : & Set \to Set \\
MonSys\ X & = & X \to M\ X
\end{array}
$$

Monadic systems

- All results extend to monadic systems systems

$$
\begin{aligned}
&\mathit{monFlow} : \{X : \mathit{Set}\} \to \mathit{MonSys}\ X \to \mathbb{N} \to \mathit{MonSys}\ X\\
&\mathit{monFlow}\ f\ \ \mathit{zero} \quad = \eta_M\\
&\mathit{monFlow}\ f\ (\mathit{suc}\ n) = f \ {>}{=}{>}_M\ \mathit{monFlow}\ f\ n
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{monTrj} : \{X : \mathit{Set}\} \to \mathit{MonSys}\ X \to (n : \mathbb{N}) \to X \to M\ (\mathit{Vec}\ X\ (\mathit{suc}\ n))\\
&\mathit{monTrj}\ f\ \mathit{zero} \quad\ x = \mathit{fmap}_M\ (x :: \_)\ (\eta_M\ [])\\
&\mathit{monTrj}\ f\ (\mathit{suc}\ n)\ x = \mathit{fmap}_M\ (x :: \_)\ (f\ x \gg=_M (\mathit{monTrj}\ f\ n))
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{detToMon} : \{X : \mathit{Set}\} \to \mathit{DetSys}\ X \to \mathit{MonSys}\ X\\
&\mathit{detToMon}\ f = \eta_M \circ f
\end{aligned}
$$

Monadic systems

$$monToDet : \{X : Set\} \to MonSys\ X \to DetSys\ (M\ X)$$
$$monToDet\ f\ mx = mx \gg\!=_M f$$

$$Det{\equiv}Mon : \{X : Set\} \to (f : DetSys\ X) \to (n : \mathbb{N}) \to (x : X) \to$$
$$\eta_M\ (detFlow\ f\ n\ x) \equiv monFlow\ (detToMon\ f)\ n\ x$$

$$Mon{\equiv}Det : \{X : Set\} \to (f : MonSys\ X) \to (n : \mathbb{N}) \to (mx : M\ X) \to$$
$$monToDet\ (monFlow\ f\ n)\ mx \equiv detFlow\ (monToDet\ f)\ n\ mx$$

- And more ...

The computational structure of *possible*: Monadic dynamical systems

The computational structure of *possible*: Monadic dynamical systems

- The bottom line is that, when the functor F is also a monad, $possible\ s\ :\ F\ Evolution$ can be defined in terms of computations like $monFlow$, $monTrj$ and their combinations

- Example 1: $Evolution\ =\ State,\ \ possible\ =\ monFlow\ next\ 5$

- Example 2: $Evolution\ =\ Vec\ State\ 5,\ \ possible\ =\ monTrj\ next\ 4$

- Example 3: $Evolution\ =\ State^2,\ \ possible\ s\ =\ fmap_M\ (\lambda\ s'\ \rightarrow\ (s\ ,\ s'))\ (monFlow\ next\ 5\ s)$

Monadic systems (extra)

Monadic systems (extra)

- We have seen that the monadic operations fulfil certain equations, for example

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

- For arbitrary A, B, C : Set one has

$$\forall\ (f : A \to F\ B) \to \qquad (\_ \gg= f) \doteq \mu \circ fmap\ f$$
$$\forall\ (f : A \to F\ B) \to (g : B \to F\ C) \to f >=> g \doteq \mu \circ fmap\ g \circ f$$
$$\mu \circ \eta \doteq id$$
$$\mu \circ fmap\ \eta \doteq id$$
$$\mu \circ \mu \doteq \mu \circ fmap\ \mu$$
$$\forall\ (f : A \to B) \to fmap\ f \circ \eta \doteq \eta \circ f$$
$$\forall\ (f : A \to B) \to fmap\ f \circ \mu \doteq \mu \circ fmap\ (fmap\ f)$$

- In this specification, $f \doteq g$ means that f is extensionally equal to g:

$$f \doteq g = (x : dom\ f) \to f\ x \equiv g\ x$$

Monadic systems (extra)

- Monadic laws are best understood diagrammatically

$$
\begin{tikzcd}
F\ C & M\ (M\ C) \arrow[l, "\mu_M"'] & M\ B \arrow[l, "fmap_M\ g"'] & A \arrow[l, "f"'] \arrow[lll, bend right, "f \ggg_M g"']
\end{tikzcd}
$$

$$
\begin{tikzcd}
M\,X \arrow[r, "\eta_M"] \arrow[dr, "id"'] & M\,(M\,X) \arrow[d, "\mu_M"] & M\,X \arrow[l, "fmap_M\ \eta_M"'] \arrow[dl, "id"] \\
 & M\,X &
\end{tikzcd}
$$

$$
\begin{tikzcd}
M\,(M\,(M\,X)) \arrow[r, "fmap_M\ \mu_M"] \arrow[d, "\mu_M"'] & M\,(M\,X) \arrow[d, "\mu_M"] \\
M\,(M\,X) \arrow[r, "\mu_M"'] & M\,X
\end{tikzcd}
$$

$$
\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "\eta_M"'] & Y \arrow[d, "\eta_M"] \\
M\,X \arrow[r, "fmap_M\ f"'] & M\,Y
\end{tikzcd}
$$

$$
\begin{tikzcd}
M\,(M\,X) \arrow[r, "fmap_M\ (fmap_M\ f)"] \arrow[d, "\mu_M"'] & M\,(M\,Y) \arrow[d, "\mu_M"] \\
M\,X \arrow[r, "fmap_M\ f"'] & M\ Y
\end{tikzcd}
$$

Monadic systems (extra)

Exercise 4.14

Postulate the monadic laws in Agda.

Exercise 4.15

Using the postulated monadic laws, prove $Det{\equiv}Mon$. (It should be very similar to the earlier proof of $Det{\equiv}NonDet$, but now for an arbitrary monad.)

From the theory of vulnerability to verified policy advice

Nicola Botta with P. Jansson and C. Ionescu

Part 2, Background: climate science, climate policy under uncertainty, 2024-04-29

Plan

Done:

- The computational structure of *possible*: Monadic dynamical systems
  - Recap vulnerability theory
  - *State*, *Evolution* and deterministic systems
  - Non-deterministic systems
  - Monadic systems

Now:

- Background: climate science, climate policy under uncertainty

Next week:

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

Background: climate science, climate policy under uncertainty

- Basic notions
- Example 1: emission reduction policies
- Optimality, policies
- Example 2: a generation dilemma (Heitzig et al. 2018)
- Examples 1 and 2: common traits
- Towards sequential decision problems

Basic notions

- We expect climate science to improve our understanding of the climate system ...
- But also ... inform climate decisions that are transparent, accountable and yield possible evolutions of the climate-economic-social system that are safe and manageable
- It follows that climate decisions cannot be informed by climate science alone!
- Because we cannot make systematic climate-economic-social experiments, the problem of finding accountable climate decisions cannot be tackled empirically, see “Formal methods as a surrogate for empirical evidences” in the *Climate science and verified programming* note

Basic notions

- In the theory of vulnerability, the impact of decisions were encoded in *State* and *possible*
- Value predicates (what is safe, what is manageable) were encoded in *harm* and in *measure*
- To extend the theory to assist climate policy advice, we need to
- 1) model how climate decisions affect possible climate-economic-social evolutions
- 2) Given value predicates on evolutions, compute decisions that provably fulfill those predicates

Basic notions

- We have started working on such an extension in 2011

8-12.1

SDP, DP, optimal control, RL ...

X, Y state and control sets

$\Gamma : X \to \mathcal{P}\, Y$ feasible controls

$\sigma : (x : X) \to \Gamma\, x \to F\, X$ transition function

$\rho : (x : X) \to \Gamma\, x \to F\, \mathbb{R}$ payoff function

$F = \mathrm{Id}, \mathrm{Prob}, \ldots$

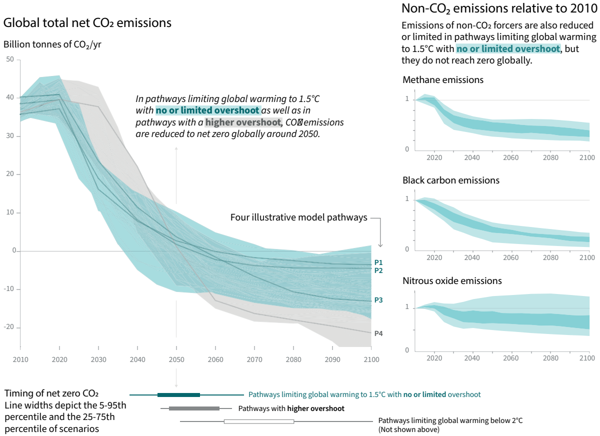
- 2014: *Sequential decision problems, dependent types and generic solutions*

- 2017: *Contributions to a computational theory of policy advice and avoidability*

Basic notions

- To motivate/explain the approach, we start by looking at a specific example
- The goal is to get an idea of the uncertainties that affect climate decision making and of ...
- ...how decision making can be accounted for in monadic systems
- The example is also an introduction to *The impact of uncertainty on optimal emission policies*

- Global GHG emissions have to be reduced to negative by about 2050



IPCC Special Report - Global Warming of 1.5 °C, Oct. 2018

Example 1: emission reduction policies

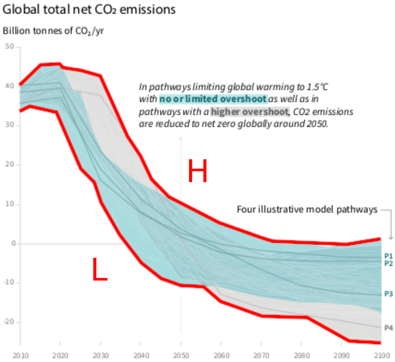
- Too fast reductions may compromise the wealth of upcoming generations but ...
- ... they may promote a transition to societies that are more wealthy, safe and fair
- Technologies that allow emission reductions at low costs may become available soon
- Rules and regulations may not be implemented or they may be implemented with delays

Example 1: emission reduction policies

- Because of these uncertainties, emission corridors like the one of the IPCC Special Report are useful but . . . also raise a number of questions:
- How to make good plans for the next few decades?
- Which plans are good under uncertainty?
- How safe is the corridor recommended by the IPCC Special Report?

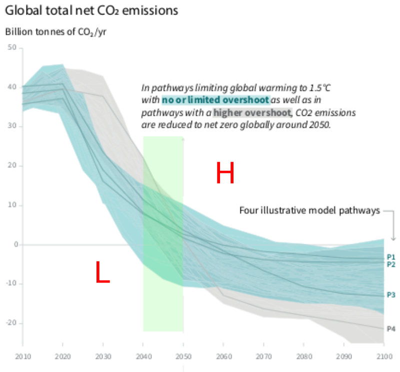
Example 1: emission reduction policies

- What are the odds of paths along the boundaries of the emissions corridor?



Example 1: emission reduction policies

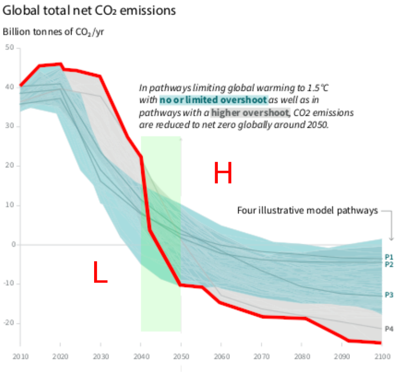
- If new technologies to reduce GHG emissions become available around 2050 ...



- ... how could optimal emission plans look like?

Example 1: emission reduction policies

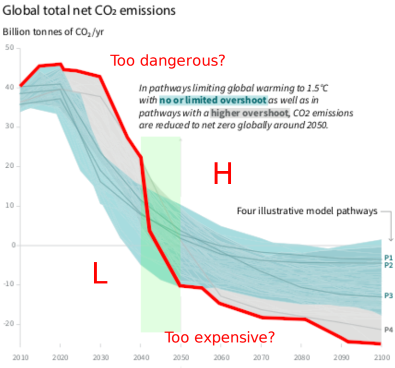
- Minimizing costs requires delaying reductions until the technologies are available



- But is this an optimal emission plan? In which sense?

Example 1: emission reduction policies

- Are H emissions until 2040 perhaps too dangerous?



- Are L emissions after 2050 possibly too expensive?

Optimality, policies

- Studying these questions requires understanding a simple but fundamental idea

- When the evolution of a system is uncertain, the notion of an optimal sequence (plan, path) of decisions becomes problematic, no matter whether F is $List$, $Maybe$, SP, or something else

- This is because, under uncertainty, more than one evolution is possible, for example

$$possible\ x \ = \ nonDetTrj\ next\ 1\ x$$

- How could a second decision possibly be optimal for all states in $nonDetFlow\ next\ 1\ x$? These could be very different from each other!

Optimality, policies

Exercise 5.1

Not every decision can be applied in every state. Decisions (controls) that can be applied in a given state are said to be feasible for that state. Give an example of a simple control problem in which certain controls are not feasible. What could be the type of a *feasible* predicate?

Exercise 5.2

Even a two-steps decision plan could be unfeasible. Explain why.

Exercise 5.3

Under stochastic uncertainty, it is generally not a good idea to take decisions which are optimal for expected states. Explain why. Give an example in which this is in fact the worst that one can do!

Optimality, policies

- Taking $1+n$ optimal decisions from s requires finding one optimal decision for s and ...
- ... one for every possible state at decision step $2, 3, \ldots 1 + n$
- In control theory, functions that map states to decisions are called policies (in economics *contingency* plans, decision *rules*, ...)
- Thus, taking $1+n$ optimal decisions under uncertainty requires computing n optimal policies
- For finite state space X and finite control (decision) space Y, this means computing at most $1 + n \cdot |Y|^{|X|}$ optimal decisions

Exercise 5.4

Explain the at most $1 + n \cdot |Y|^{|X|}$ estimate.

From the theory of vulnerability to verified policy advice

Nicola Botta with P. Jansson and C. Ionescu

Part 2, Background: climate science, climate policy under uncertainty, 2024-05-06

Plan

Done:

- The computational structure of *possible*: Monadic dynamical systems
- Background: climate science, climate policy under uncertainty
  - Basic notions
  - Example 1: emission reduction policies
  - Optimality, policies

Today:

- Background: climate science, climate policy under uncertainty
  - Example 2: a generation dilemma (Heitzig et al. 2018)
  - Examples 1 and 2: common traits
  - Towards sequential decision problems
- Sequential decision problems

Plan

Today:

- Background: climate science, climate policy under uncertainty
    - Example 2: a generation dilemma (Heitzig et al. 2018)
    - Examples 1 and 2: common traits
    - Towards sequential decision problems
- Sequential decision problems

Next week:

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

Optimality, policies

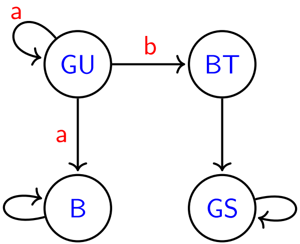
- With the understanding that what can be optimal under uncertainty are policies and with a notion of optimality, we can formulate the questions from the emission reduction example consistently
- How do optimal policies change if we account for the fact that technological innovation could become available later or earlier?
- How do optimal policies change if there is a non-zero probability of exceeding critical thresholds even if we stay within the IPCC emission corridor?
- How do optimal policies change if we account for the fact that climate decisions may not be implemented, for example, because of political instability or because of external shocks?

Example 2: a generation dilemma (Heitzig et al. 2018)

- The world can be in one of four states: GU, GS, B and BT
- B is a bad state, one in which resources are depleted and the wealth of the societies is low
- GS is a good, safe state. In GS, plenty of resources are available, societies are wealthy and there is no risk to turn into B, GU or BT
- GU is a good but unsafe state. In GU, plenty of resources are available, societies are wealthy but there is a significant risk to turn into B
- BT is a bad but temporary state
- In BT, societies are poor but it is certain that the next state will be good and safe: GS

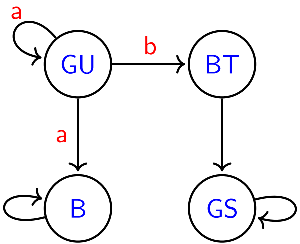
Example 2: a generation dilemma (Heitzig et al. 2018)

- A generation in B, BT or GS has no options: the next states will be B, GS and GS
- A generation in GU has two options: a and b
- If it picks a, the next generation will possibly be in GU again. But it can also end up in B
- If it picks b, the next generation will be in BT with certainty



- What should a generation in GU do? a or b?

Example 2: a generation dilemma (Heitzig et al. 2018)



Exercise 5.5

Should a generation in GU do a or b? The answer is: it depends. Explain on what it might depend.

Exercise 5.6

Put consistent probabilities on the edges of the transition graph above.

Examples 1 and 2: common traits

- Both decision problems have the form of a dilemma
- In both cases, the consequences of decisions are uncertain
- Decisions are taken sequentially, one after the other, see *Incorporating path dependency into decision-analytic methods: an application to global climate-change policy*
- Can we exploit these similarities? Can we develop a method for specifying and solving these and similar decisions problems rigorously? What does this mean?

Towards sequential decision problems

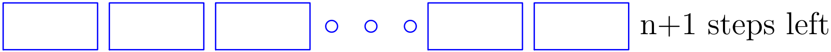
- We tackle these questions in three steps:

  1. Abstract away the details of specific decision problems
  2. Formulate a class of decision problems rigorously
  3. Derive generic, verified solution methods for this class

Towards sequential decision problems: step 1

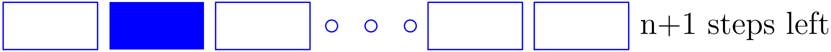
Towards sequential decision problems: step 1

There are $n + 1$ decision steps to go . . .



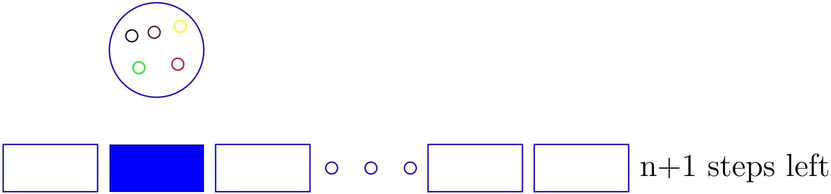
Towards sequential decision problems: step 1

. . . here is the current state,



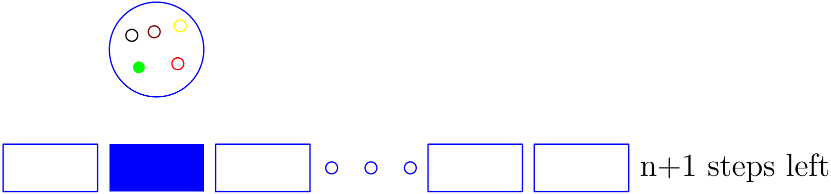
Towards sequential decision problems: step 1

. . . here are your options.



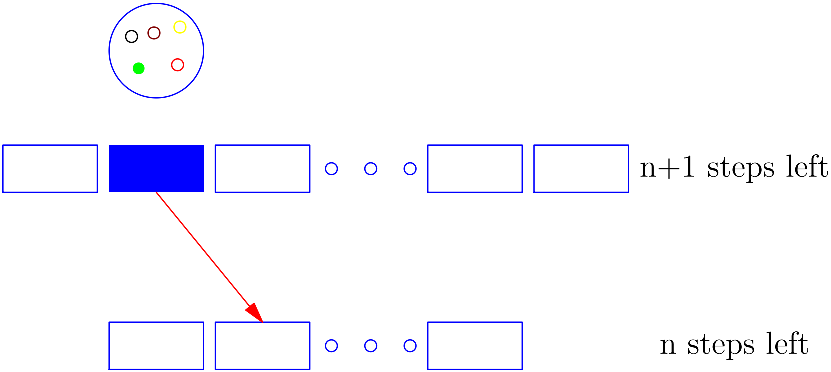
Towards sequential decision problems: step 1

Pick one!



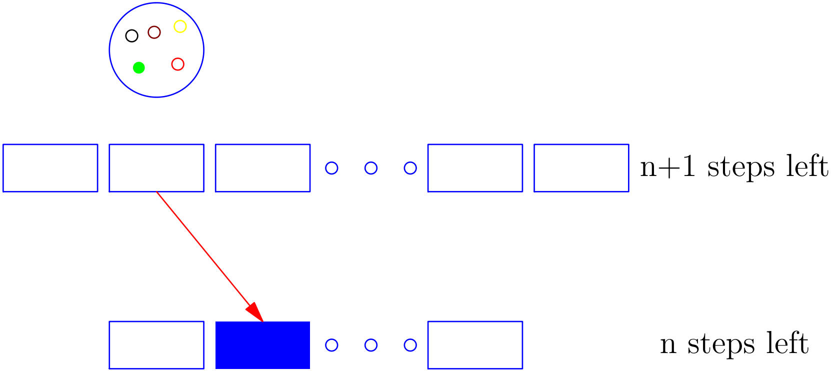
Towards sequential decision problems: step 1

Move to a new state and . . .



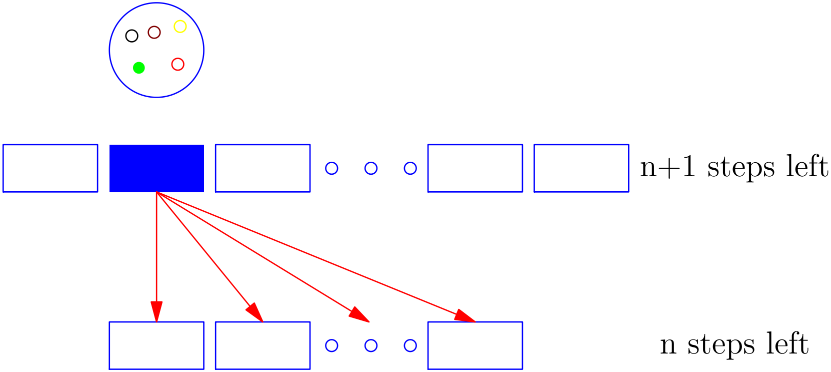
Towards sequential decision problems: step 1

. . . collect rewards and face the next decision step!



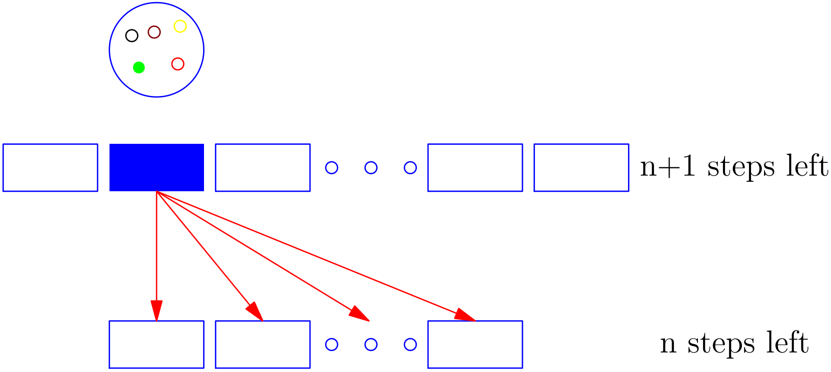
Towards sequential decision problems: step 1

What if there are more than one next possible states?



Towards sequential decision problems: step 1

Apply monadic systems theory!



Next week

- Steps 1-3: Sequential decision problems
- Bellman's equation, backward induction
- Verified policy advice in a nutshell

Exercise 5.7

Try to formalize the cartoon of step 1 (abstract away the details of specific decision problems) in Agda

From the theory of vulnerability to verified policy advice

Nicola Botta with P. Jansson and C. Ionescu

Part 3, 2024-05-06

Plan

Done:

- The computational structure of *possible*: Monadic dynamical systems
- Background: climate science, climate policy under uncertainty

Today:

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

Sequential decision problems

Sequential decision problems

- As in the vulnerability theory, we build a theory for specifying and solving *finite horizon sequential decision problems* (SDP) in terms of a number of postulates or partial definitions
- These are the problem specification components of the theory/library
- The rest are problem solution components
- The theory is applied by fully defining the specification components

Specification: monadic uncertainty, number of decision steps

- The problem is affected by monadic uncertainty

$$
\begin{array}{lll}
M & : & Set \to Set \\
fmap_M & : & \{A\ B : Set\} \to (A \to B) \to M\ A \to M\ B \\
\eta_M & : & \{A : Set\} \to A \to M\ A \\
\mu_M & : & \{A : Set\} \to M\ (M\ A) \to M\ A
\end{array}
$$

- We want to make a finite number of decision steps

- At decision step $t : \mathbb{N}$, we have already taken t decisions

Specification: states, controls, transition function

- The set of states of the problem can be different at different decision steps

$$X \,:\, \mathbb{N} \to \mathit{Set}$$

- The set of controls can be different at different decision steps and in different states

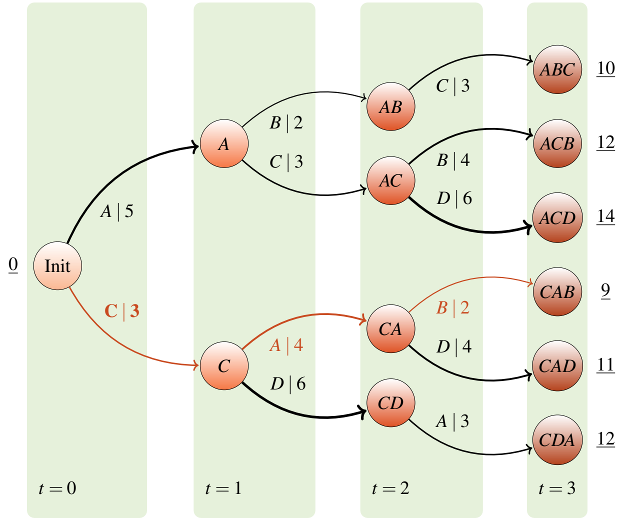
$$Y \,:\, (t \,:\, \mathbb{N}) \to X\, t \to \mathit{Set}$$

- Selecting a control $y : Y\ t\ x$ in $x : X\ t$ yields an M-structure of possible next states

$$\mathit{next} \,:\, (t \,:\, \mathbb{N}) \to (x \,:\, X\, t) \to Y\, t\, x \to M\, (X\, (\mathit{suc}\ t))$$

- *next* describes an infinite, layered DAG with states in the nodes

Specification: states, controls, transition function



Specification: values, reward function

- Each decision step yields a reward in a value set Val

$$Val \ : \ Set$$

- As in vulnerability theory, we require Val to be a preorder, here a total one

$$
\begin{aligned}
\_\leqslant\_ \ &: \ Val \ \to \ Val \ \to \ Set \\
refl_{\leqslant} \ &: \ (x \ : \ Val) \ \to \ x \ \leqslant \ x \\
trans_{\leqslant} \ &: \ (x \ y \ z \ : \ Val) \ \to \ x \ \leqslant \ y \ \to \ y \ \leqslant \ z \ \to \ x \ \leqslant \ z \\
total_{\leqslant} \ &: \ (x \ y \ : \ Val) \ \to \ Either \ (x \ \leqslant \ y) \ (y \ \leqslant \ x)
\end{aligned}
$$

- We will also need Val to have a reference "zero" element and an "addition"

$$
\begin{aligned}
0_{Val} \ &: \ Val \\
\_\oplus\_ \ &: \ Val \ \to \ Val \ \to \ Val
\end{aligned}
$$

Specification: reward function, solving a SDP

- Different combinations of current state, control and next state can lead to different rewards

$$reward \;:\; (t \;:\; \mathbb{N}) \;\to\; (x \;:\; X\;t) \;\to\; Y\;t\;x \;\to\; X\;(suc\;t) \;\to\; Val$$

- Solving a SDP means finding a sequence of policies that maximizes a measure of the $\oplus$-sum of the rewards along all possible trajectories

Exercise 6.1

Make sure that you fully understand what solving a SDP means.

Exercise 6.2

Define M, X, Y and *next* for the generation dilemma. What could be Val and *reward* for this problem?

Solution: policies, policy sequences

- ...finding a sequence of policies that maximizes a measure of the $\oplus$-sum of the rewards along all possible trajectories
- We need to formulate the problem precisely
- We start with policies and policy sequences

$$
\begin{aligned}
&\mathit{Policy} \;:\; (t : \mathbb{N}) \to \mathit{Set}\\
&\mathit{Policy}\; t \;=\; (x : X\; t) \to Y\; t\; x
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{data}\; \mathit{PolicySeq} : (t\; n : \mathbb{N}) \to \mathit{Set}\; \textbf{where}\\
&\quad \mathit{Nil} \;:\; \{t : \mathbb{N}\} \to \mathit{PolicySeq}\; t\; \mathit{zero}\\
&\quad \_{::}\_ \;:\; \{t\; n : \mathbb{N}\} \to \mathit{Policy}\; t \to \mathit{PolicySeq}\; (\mathit{suc}\; t)\; n \to \mathit{PolicySeq}\; t\; (\mathit{suc}\; n)
\end{aligned}
$$

Exercise 6.3

Explain the $(\mathit{suc}\; t)\; n$ - $t\; (\mathit{suc}\; n)$ pattern in the definition of *PolicySeq*.

Solution: state-control sequences

- ... a pol. seq. that max. a meas. of the $\oplus$-sum of the rewards along all possible trajectories
- We want the possible trajectories of a policy sequence to be sequences of state-control pairs

$$
\begin{array}{l}
\textbf{data}\ \mathit{XYSeq} : (t\ n : \mathbb{N}) \to \mathit{Set}\ \textbf{where} \\
\quad \mathit{Last} : \{t : \mathbb{N}\} \quad \to X\ t \to \mathit{XYSeq}\ t\ (\mathit{suc}\ \mathit{zero}) \\
\quad \_{||}\_ \ : \{t\ n : \mathbb{N}\} \to \Sigma\ (X\ t)\ (Y\ t) \to \mathit{XYSeq}\ (\mathit{suc}\ t)\ (\mathit{suc}\ n) \to \mathit{XYSeq}\ t\ (\mathit{suc}\ (\mathit{suc}\ n))
\end{array}
$$

Exercise 6.4

A value of type $\mathit{XYSeq}\ t\ n$ is like a vector. What is its length? Can n be zero? Why is the first constructor of XYSeq called Last?

Solution: possible trajectories

- ... a pol. seq. that max. a meas. of the $\oplus$-sum of the rewards along all possible trajectories
- We compute the possible trajectories of a policy sequence as we did for monadic systems

$$
\begin{aligned}
& trj \; : \; \{t \; n \; : \; \mathbb{N}\} \to PolicySeq\; t\; n \to X\; t \to M\; (XYSeq\; t\; (suc\; n)) \\
& trj\; \{t\}\; Nil \qquad\quad x \;=\; \eta_M\; (Last\; x) \\
& trj\; \{t\}\; (p :: ps)\; x \;=\; \mathbf{let}\; y \quad\;\; = \; p\; x\; \mathbf{in} \\
& \qquad\qquad\qquad\qquad\quad\;\; \mathbf{let}\; mx' \;=\; next\; t\; x\; y\; \mathbf{in} \\
& \qquad\qquad\qquad\qquad\quad\;\; fmap_M\; ((x\; ,\; y)\; {\shortparallel}\_)\; (mx' \;\gg\!=_M\; trj\; ps)
\end{aligned}
$$

Exercise 6.5

Make sure that you understand the computation of possible trajectories. What are the types of y, mx' in the **let-in** clauses?

Solution: possible trajectories

- ... a pol. seq. that max. a meas. of the $\oplus$-sum of the rewards along all possible trajectories
- Now we can compute the $\oplus$-sum of the rewards along all possible trajectories ...

$$
\begin{aligned}
&sumR : \{t\ n : \mathbb{N}\} \to XYSeq\ t\ n \to Val \\
&sumR\ \{t\}\ (Last\ x) = 0_{Val} \\
&sumR\ \{t\}\ ((x\ ,\ y) \,{:}{:}\, xys) = reward\ t\ x\ y\ (head\ xys) \oplus sumR\ xys
\end{aligned}
$$

- ... and the value of taking n decisions according to a policy sequence in an initial state

$$
\begin{aligned}
&val : \{t\ n : \mathbb{N}\} \to (ps : PolicySeq\ t\ n) \to (x : X\ t) \to Val \\
&val\ ps = measure \circ fmap_M\ sumR \circ trj\ ps
\end{aligned}
$$

Exercise 6.6

Notice that *val ps* is a vulnerability measure! What are *possible* and *harm* here?

From the theory of vulnerability to verified policy advice

Nicola Botta with P. Jansson and C. Ionescu

Part 3, 2024-05-13

Plan

Done:

- The computational structure of *possible*: Monadic dynamical systems
- Background: climate science, climate policy under uncertainty
- Sequential decision problems (spec. and sol. components)

Today:

- Sequential decision problems (two more slides)
- Bellman's equation, backward induction
- Verified policy advice in a nutshell

Recap

$$
\begin{array}{lll}
M & : & \mathit{Set} \;\rightarrow\; \mathit{Set} \\
X & : & \mathbb{N} \;\rightarrow\; \mathit{Set} \\
Y & : & (t \;:\; \mathbb{N}) \;\rightarrow\; X\;t \;\rightarrow\; \mathit{Set} \\
\mathit{next} & : & (t \;:\; \mathbb{N}) \;\rightarrow\; (x \;:\; X\;t) \;\rightarrow\; Y\;t\;x \;\rightarrow\; M\,(X\,(\mathit{suc}\;t)) \\
\mathit{Val} & : & \mathit{Set} \\
\mathit{reward} & : & (t \;:\; \mathbb{N}) \;\rightarrow\; (x \;:\; X\;t) \;\rightarrow\; Y\;t\;x \;\rightarrow\; X\,(\mathit{suc}\;t) \;\rightarrow\; \mathit{Val} \\
\mathit{measure} & : & \;\; M\;\mathit{Val} \;\rightarrow\; \mathit{Val} \\
\_\oplus\_ & : & \;\; \mathit{Val} \;\rightarrow\; \mathit{Val} \;\rightarrow\; \mathit{Val} \\
\mathit{Policy}\;t & = & (x \;:\; X\;t) \;\rightarrow\; Y\;t\;x \\
\mathbf{data}\;\mathit{PolicySeq} & : & \;\; (t\;n \;:\; \mathbb{N}) \;\rightarrow\; \mathit{Set}\;\mathbf{where}\;\ldots \\
\mathbf{data}\;\mathit{XYSeq} & : & \;\; (t\;n \;:\; \mathbb{N}) \;\rightarrow\; \mathit{Set}\;\mathbf{where}\;\ldots \\
\mathit{trj} & : & \;\; \{t\;n \;:\; \mathbb{N}\} \;\rightarrow\; \mathit{PolicySeq}\;t\;n \;\rightarrow\; X\;t \;\rightarrow\; M\,(\mathit{XYSeq}\;t\,(\mathit{suc}\;n))
\end{array}
$$

Recap

- A solution of a SDP is a policy sequence that maximizes the measure of the $\oplus$-sum of the rewards along all possible trajectories

$$
\begin{aligned}
&\mathit{sumR} : \{t\ n : \mathbb{N}\} \to \mathit{XYSeq}\ t\ n \to \mathit{Val} \\
&\mathit{sumR}\ \{t\}\ (\mathit{Last}\ x) = 0_{\mathit{Val}} \\
&\mathit{sumR}\ \{t\}\ ((x\ ,\ y) \mathbin{::} \mathit{xys}) = \mathit{reward}\ t\ x\ y\ (\mathit{head}\ \mathit{xys}) \oplus \mathit{sumR}\ \mathit{xys}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{val} : \{t\ n : \mathbb{N}\} \to (\mathit{ps} : \mathit{PolicySeq}\ t\ n) \to (x : X\ t) \to \mathit{Val} \\
&\mathit{val}\ \mathit{ps} = \mathit{measure} \circ \mathit{fmap}_M\ \mathit{sumR} \circ \mathit{trj}\ \mathit{ps}
\end{aligned}
$$

Exercise 7.1

What are the types of *head* and *measure* in the definitions of *sumR* and *val*? Define *head*. How could the type of *measure* be generalized?

Solution: optimality of policy sequences

- A solution of a SDP is a policy sequence that maximizes the measure of the $\oplus$-sum of the rewards along all possible trajectories

- Now we can express what it means for a policy sequence to be a solution of a SDP precisely

$$
\begin{aligned}
&\mathit{OptPolicySeq} : \{t\ n : \mathbb{N}\} \to \mathit{PolicySeq}\ t\ n \to \mathit{Set} \\
&\mathit{OptPolicySeq}\ \{t\}\ \{n\}\ \mathit{ps} = \forall\ (\mathit{ps}' : \mathit{PolicySeq}\ t\ n) \to \mathit{val}\ \mathit{ps}' \leqslant_l \mathit{val}\ \mathit{ps}
\end{aligned}
$$

Exercise 7.2

What is the type of $\leqslant_l$ in the definition of *OptPolicySeq*? Define $\leqslant_l$ in terms of $\leqslant$.

Bellman's equation, backward induction

Bellman's equation

- We have understood what it means for a policy sequence to be optimal but ...
- ... how can one compute optimal policy sequences?
- For finite $X\ t$, $Y\ t\ x$, an obvious approach is brute-force:
- Generate all policy sequences pss, then pick up a $ps \in pss$ such that

$$\forall\ (ps' \in pss) \rightarrow val\ ps' \leqslant_l val\ ps$$
$$\equiv$$
$$\forall\ (ps' \in pss) \rightarrow measure \circ fmap_M\ sumR \circ trj\ ps' \leqslant_l measure \circ fmap_M\ sumR \circ trj\ ps$$

- Around 1954, Bellman came up with a much better idea: *dynamic programming*!

Bellman's equation

- Let's have a look at the value of $[p0\ ,\ p1]$ in $x_0 : X\ 0$ for $M = SP$, $SP\ X = List\ (X\ ,\ \mathbb{R}_{[0,1]})$ with $Val = \mathbb{R}$, $\oplus = +$, $0_{Val} = 0$ and the expected value measure
- Let α and β be arbitrary probabilities and

$$p_0\ x_0 = y_0 \qquad next\ 0\ x_0\ y_0 = [\ (x_1^0\ ,\ \alpha)\ ,\ (x_1^1\ ,\ 1 - \alpha)\ ] \qquad \begin{array}{l} reward\ 0\ x_0\ y_0\ x_1^0 = r_0^0 \\ reward\ 0\ x_1\ y_0\ x_1^1 = r_0^1 \end{array}$$

$$p_1\ x_1^0 = y_1^0 \qquad next\ 1\ x_1^0\ y_1^0 = [\ (x_2^{0,0}\ ,\ \beta)\ ,\ (x_2^{0,1}\ ,\ 1 - \beta)\ ] \qquad \begin{array}{l} reward\ 1\ x_1^0\ y_1^0\ x_2^{0,0} = r_1^{0,0} \\ reward\ 1\ x_1^0\ y_1^0\ x_2^{0,1} = r_1^{0,1} \end{array}$$

$$p_1\ x_1^1 = y_1^1 \qquad next\ 1\ x_1^1\ y_1^1 = [\ (x_2^{1,0}\ ,\ 1)\ ] \qquad reward\ 1\ x_1^1\ y_1^1\ x_2^{1,0} = r_1^{1,0}$$

Exercise 7.3

On the fly: How many trajectories are in $trj\ [p0\ ,\ p1]\ x_0$?

Bellman's equation

We compute

$$
\begin{aligned}
& \mathit{val}\ [p0\ ,\ p1]\ x_0 \\
& \quad = \{\ \mathit{step}_1\ \}\ = \\
& \mathit{ev}\ (\mathit{fmap}_{SP}\ \mathit{sumR}\ (\mathit{trj}\ [p0\ ,\ p1]\ x_0)) \\
& \quad = \{\ \mathit{step}_2\ \}\ = \\
& \mathit{ev}\ (\mathit{fmap}_{SP}\ \mathit{sumR}\ [\ (((x_0\ ,\ y_0)\ ::\ (x^0_1\ ,\ y^0_1)\ ::\ \mathit{Last}\ x^{0,0}_2)\ ,\ \alpha * \beta)\ , \\
& \qquad\qquad\qquad\qquad (((x_0\ ,\ y_0)\ ::\ (x^0_1\ ,\ y^0_1)\ ::\ \mathit{Last}\ x^{0,1}_2)\ ,\ \alpha * (1 - \beta))\ , \\
& \qquad\qquad\qquad\qquad (((x_0\ ,\ y_0)\ ::\ (x^1_1\ ,\ y^1_1)\ ::\ \mathit{Last}\ x^{1,0}_2)\ ,\ 1 - \alpha)\ ]) \\
& \quad = \{\ \mathit{step}_3\ \}\ = \\
& \mathit{ev}\ [\ (r^0_0 + r^{0,0}_1\ ,\ \alpha * \beta)\ ,\ (r^0_0 + r^{0,1}_1\ ,\ \alpha * (1 - \beta))\ ,\ (r^1_0 + r^{1,0}_1\ ,\ 1 - \alpha)\ ] \\
& \quad = \{\ \mathit{step}_4\ \}\ = \\
& r^0_0 * \alpha + r^{0,0}_1 * \beta * \alpha + r^{0,1}_1 * (1 - \beta) * \alpha + r^1_0 * (1 - \alpha) + r^{1,0}_1 * (1 - \alpha)
\end{aligned}
$$

Bellman's equation

Exercise 7.4

In $step_2$ we have applied the generic trajectory computation

$$
\begin{aligned}
&trj : \{t\ n : \mathbb{N}\} \to PolicySeq\ t\ n \to X\ t \to M\ (XYSeq\ t\ (suc\ n))\\
&trj\ \{t\}\ Nil \qquad x = \eta_M\ (Last\ x)\\
&trj\ \{t\}\ (p :: ps)\ x = \textbf{let}\ y \quad = p\ x\ \textbf{in}\\
&\qquad\qquad\qquad\qquad\ \ \textbf{let}\ mx' = next\ t\ x\ y\ \textbf{in}\\
&\qquad\qquad\qquad\qquad\ \ fmap_M\ ((x\ ,\ y)\ ||\_)\ (mx' \gg=_M\ trj\ ps)
\end{aligned}
$$

for $M = SP$. Define η_{SP}, $fmap_{SP}$ and $\gg=_{SP}$ such that $trj\ [p0\ ,\ p1]\ x_0$ yields the result of $step_2$.

Exercise 7.5

In $step_4$ we have applied a definition of the exp. value measure ev. Define ev consistently with $step_4$.

Bellman's equation

$$= \{ \, step_5 \, \} =$$

$$r_0^0 * \alpha + r_1^{0,0} * \beta * \alpha + r_1^{0,1} * (1 - \beta) * \alpha + r_0^1 * (1 - \alpha) + r_1^{1,0} * (1 - \alpha)$$

$$= \{ \, step_6 \, \} =$$

$$ev \; [\; (r_0^0 + r_1^{0,0} * \beta + r_1^{0,1} * (1 - \beta) \, , \alpha) \, , (r_0^1 + r_1^{1,0} \, , 1 - \alpha) \;]$$

$$= \{ \, step_7 \, \} =$$

$$ev \; [\; (r_0^0 + ev \; [\; (r_1^{0,0} \, , \beta) \, , (r_1^{0,1} \, , 1 - \beta) \;] \, , \alpha) \, , (r_0^1 + ev \; [\; (r_1^{1,0} \, , 1) \;] \, , 1 - \alpha) \;]$$

$$= \{ \, step_8 \, \} =$$

$$ev \; [\; (r_0^0 + ev \; (fmap_{SP} \; sumR \; [\; (((x_1^0 \, , y_1^0) \;{\shortparallel}\; Last \; x_2^{0,0}) \, , \beta) \, , (((x_1^0 \, , y_1^0) \;{\shortparallel}\; Last \; x_2^{0,1}) \, , 1 - \beta) \;]) \, , \alpha) \, ,$$
$$(r_0^1 + ev \; (fmap_{SP} \; sumR \; [\; (((x_1^1 \, , y_1^1) \;{\shortparallel}\; Last \; x_2^{1,0}) \, , 1) \;]) \, , 1 - \alpha) \;]$$

$$= \{ \, step_9 \, \} =$$

$$ev \; [\; (r_0^0 + ev \; (fmap_{SP} \; sumR \; (trj \; [p_1] \; x_1^0)) \, , \alpha) \, , (r_0^1 + ev \; (fmap_{SP} \; sumR \; (trj \; [p_1] \; x_1^1)) \, , 1 - \alpha) \;]$$

$$= \{ \, step_{10} \, \} =$$

$$ev \; [\; (reward \; 0 \; x_0 \; y_0 \; x_1^0 + val \; [p_1] \; x_1^0 \, , \alpha) \, , (reward \; 0 \; x_0 \; y_0 \; x_1^1 + val \; [p_1] \; x_1^1 \, , 1 - \alpha) \;]$$

Bellman's equation

$$= \{\, step_{11} \,\} =$$

$$ev\ [\ (reward\ 0\ x_0\ y_0\ x_1^0 + val\ [p_1]\ x_1^0\ ,\ \alpha)\ ,\ (reward\ 0\ x_0\ y_0\ x_1^1 + val\ [p_1]\ x_1^1\ ,\ 1 - \alpha)\ ]$$

$$= \{\, step_{12} \,\} =$$

$$ev\ (\ fmap_{SP}\ (reward\ 0\ x_0\ y_0\ \oplus_l\ val\ [p_1])\ [\ (x_1^0\ ,\ \alpha)\ ,\ (x_1^1\ ,\ 1 - \alpha)\ ])$$

$$= \{\, step_{13} \,\} =$$

$$ev\ (\ fmap_{SP}\ (reward\ 0\ x_0\ y_0\ \oplus_l\ val\ [p_1])\ (next\ 0\ x_0\ y_0))$$

Thus, we have computed

$$val\ [p0\ ,\ p1]\ x_0 = ev\ (fmap_{SP}\ (reward\ 0\ x_0\ y_0\ \oplus_l\ val\ [p_1])\ (next\ 0\ x_0\ y_0))$$

Bellman's equation

Exercise 7.6

Is the computation correct? Check it and report eventual errors!

Exercise 7.7

Redo the computation for the non-deterministic case with the canonical monadic operations for *List* and with $measure = sum$. Do you obtain the same computational pattern?

- For stochastic SDPs, one can generalize the result to

$$val\ (p :: ps)\ x \ = \ ev\ (fmap_{SP}\ (reward\ t\ x\ (p\ x)\ \oplus_l\ val\ ps)\ (next\ t\ x\ (p\ x)))$$

for arbitrary p, ps, $reward$ and $next$ of consistent types. This is *Bellman's equation*!

Bellman's equation

Exercise 7.8

Not surprisingly, Bellman's equation also holds for the "plain" deterministic case. Prove

$$\begin{aligned} BellmanEq \; : \; & (t \; n \; : \; \mathbb{N}) \to (p \; : \; Policy \; t) \to (ps \; : \; PolicySeq \; (suc \; t) \; n) \to (x \; : \; X \; t) \to \\ & val \; (p :: ps) \; x \; \equiv \; measure \; (fmap \; (reward \; t \; x \; (p \; x) \; \oplus_l \; val \; ps) \; (next \; t \; x \; (p \; x))) \end{aligned}$$

by induction on ps and with $M \; = \; Id, \;\; Id \; X \; = \; X, \;\; measure \; = \; id$, arbitrary $next$,

$$val \; ps \; = \; measure \circ fmap \; sumR \circ trj \; ps$$

and with $fmap_{Id} \; = \; \eta_{Id} \; = \; id$ and $x \; \gg\!=_{Id} \; f \; = \; f \; x$ for the identity monad. Apply

$$\begin{aligned} Lemma \; : \; & (t \; n \; : \; \mathbb{N}) \to (p \; : \; Policy \; t) \to (ps \; : \; PolicySeq \; (suc \; t) \; n) \to (x \; : \; X \; t) \to \\ & sumR \; (trj \; (p :: ps) \; x) \; \equiv \; reward \; t \; x \; (p \; x) \; (next \; t \; x \; (p \; x)) \; \oplus \; val \; ps \; (next \; t \; x \; (p \; x)) \end{aligned}$$

Bellman's equation

- If *measure*, $\oplus$ and M fulfill certain *compatibility conditions*, Bellman's equation can be generalized to the monadic case

- In this case one defines the value of a policy sequence through Bellman's equation

$$
\begin{array}{lll}
\mathit{val} : \{t\ n : \mathbb{N}\} \to \mathit{PolicySeq}\ t\ n \to X\ t \to \mathit{Val} & & \\
\mathit{val}\ \{t\}\ \mathit{Nil}\ x & = & 0_{\mathit{Val}} \\
\mathit{val}\ \{t\}\ (p :: ps)\ x & = & \mathbf{let}\ y \quad = \ p\ x\ \mathbf{in} \\
 & & \mathbf{let}\ mx' \ = \ \mathit{next}\ t\ x\ y\ \mathbf{in} \\
 & & \mathit{measure}\ (\mathit{fmap}_M\ (\mathit{reward}\ t\ x\ y\ \oplus_l\ \mathit{val}\ ps)\ mx')
\end{array}
$$

- This definition is the key for solving SDPs via backward induction

- Backward induction follows directly from Bellman's optimality principle

Bellman's principle, optimal extensions

- Optimal extensions of optimal policy sequences are optimal

$$
\begin{aligned}
&Bellman : \{t\ n : \mathbb{N}\} \to (p : Policy\ t) \to (ps : PolicySeq\ (suc\ t)\ n) \to \\
&\qquad OptExt\ ps\ p \to OptPolicySeq\ ps \to OptPolicySeq\ (p :: ps)
\end{aligned}
$$

- p is an optimal extension of ps iff p is at least as good as any other policy

$$
\begin{aligned}
&OptExt : \{t\ n : \mathbb{N}\} \to PolicySeq\ (suc\ t)\ n \to Policy\ t \to Set \\
&OptExt\ ps\ p = \forall\ p' \to val\ (p' :: ps) \leqslant_l val\ (p :: ps)
\end{aligned}
$$

Exercise 7.9

Thus, computing an optimal extension $p\ :\ Policy\ t$ of a policy sequence ps requires computing a control $p\ x\ :\ Y\ t\ x$ that maximizes $val\ (p :: ps)\ x$ for every $x\ :\ X\ t$. When $Y\ t\ x$ is non-empty and finite, this can be done easily. Implement

$$optExt\ :\ \{t\ n\ :\ \mathbb{N}\} \rightarrow PolicySeq\ (suc\ t)\ n \rightarrow Policy\ t$$

for this case by applying

$$\begin{aligned}
&Finite && :\ Set \rightarrow Set \\
&toList && :\ \{A\ :\ Set\} \rightarrow Finite\ A \rightarrow List\ A \\
&max && :\ \{A\ :\ Set\} \rightarrow (f\ :\ A \rightarrow Val) \rightarrow List\ A \rightarrow Val \\
&argmax && :\ \{A\ :\ Set\} \rightarrow (f\ :\ A \rightarrow Val) \rightarrow List\ A \rightarrow A
\end{aligned}$$

Bellman's principle, optimal extensions

Exercise 7.10

Formulate minimal requirements on *toList*, *max* and *argmax* for *optExt* to satisfy

$$optExtSpec \;:\; \{t\; n \;:\; \mathbb{N}\} \to (ps \;:\; PolicySeq\,(suc\; t)\; n) \to OptExt\; ps\,(optExt\; ps)$$

- To prove Bellman's optimality principle one needs two monotonicity conditions

$$\begin{aligned} measureMon \;:\; &\{A \;:\; Set\} \to (f\; g \;:\; A \to Val) \to (f \leqslant_l g) \to \\ &(ma \;:\; M\; A) \to measure\,(fmap_M\; f\; ma) \leqslant measure\,(fmap_M\; g\; ma) \end{aligned}$$

$$plusMon \;:\; \{a\; b\; c\; d \;:\; Val\} \to a \leqslant b \to c \leqslant d \to (a \oplus c) \leqslant (b \oplus d)$$

Exercise 7.11

Postulate *measureMon*, *plusMon* and implement *Bellman*.

Verified backward induction

- With *optExt* one can solve SDPs by backward induction

$$
\begin{aligned}
&bi : (t\ n : \mathbb{N}) \to PolicySeq\ t\ n \\
&bi\ t\ zero \quad = Nil \\
&bi\ t\ (suc\ n) = \mathbf{let}\ ps = bi\ (suc\ t)\ n\ \mathbf{in}\ optExt\ ps :: ps
\end{aligned}
$$

- With *Bellman* and *optExtSpec* one can verify that *bi* yields optimal policy sequences

$$
biOptVal : (t\ n : \mathbb{N}) \to OptPolicySeq\ (bi\ t\ n)
$$

Exercise 7.12

Implement *biOptVal* by induction on n

$$\begin{aligned} \mathit{biOptVal}\ t\ \mathit{zero} &= \\ \mathit{biOptVal}\ t\ (\mathit{suc}\ n) &= \end{aligned}$$

Notice that policy sequences for zero decision steps are optimal by reflexivity of $\leqslant$.

Verified policy advice: wrap-up, technical issues

- This was a simplified account of the theory from the 2017 LMCS and JFP papers as it is presented in the last two papers *ESD2018* and *JFP2023*
- The bottom line is that, if M, $\leqslant$ and $\oplus$ fulfil fairly natural conditions, one can compute verified optimal policies for arbitrary finite-horizon SDPs
- The simplified theory is easy to discuss but has a major flaw: what if a control set $Y\ t\ x$ is empty? Or if *next* returns an empty *M*-structure of possible next states?
- We can "fix" the theory by requiring $Y\ t\ x$, *next t x y* to contain at least one element or ...
- ...by building a more general theory, as done in *JFP2017* by restricting the domain of policies to states that are *viable* for a suitable number of decision steps!

Verified policy advice: limitations

- The theory is general enough to support verified optimal decision making under uncertainty for a finite number of decision steps
- What about optimal decision making for infinite many decision steps?
- What if one is required to provide decision makers with all optimal policy sequences rather than just one?
- There are also more "practical" limitations ...

Exercise 7.13

...which ones come up to your mind?