Proving versus testing in climate impact research

Cezar Ionescu

PIK: Potsdam Institute for Climate Impact Research

"At PIK researchers in the natural and social sciences work together to study global change and its impacts on ecological, economic and social systems. They examine the Earth system's capacity for withstanding human interventions and devise strategies for a sustainable development of humankind and nature.

PIK research projects are interdisciplinary and undertaken by scientists from the following Research Domains: Earth System Analysis, Climate Impacts and Vulnerabilities, Sustainable Solutions and Transdisciplinary Concepts and Methods."

PIK: Potsdam Institute for Climate Impact Research

"At PIK researchers in the natural and social sciences work together to study global change and its impacts on ecological, economic and social systems. They examine the Earth system's capacity for withstanding human interventions and devise strategies for a sustainable development of humankind and nature.

PIK research projects are interdisciplinary and undertaken by scientists from the following Research Domains: Earth System Analysis, Climate Impacts and Vulnerabilities, Sustainable Solutions and Transdisciplinary Concepts and Methods."

Brave new world.

Lots of scope for conceptual analysis.

Examples:

- vulnerability
- resilience
- adaptive capacity
- mitigation
- sustainability
- . . .

Opinions differ. . .

physicist: partial differential equations

economist: utility functions

social scientist: subject interviews

mathematician: sets, lattices, categories

Example: "Vulnerability"

". . . a human condition or process resulting from physical, social and environmental factors which determine the likelihood and damage from the impact of a given hazard" (UNDP Annual Report, 2004)

"Vulnerability [. . .] is a way of conceptualizing what may happen to an identifiable population under conditions of particular risk and hazards." (Cannon et al. 2004)

". . . the degree to which a system is susceptible to and unable to cope with, adverse effects of climate change, including climate variability and extremes. " (The Intergovernmental Panel on Climate Change, 2007)

Basic elements:

S	-- set of states
$Trj = List\ S\ or\ Trj = T \to S$	-- trajectory
$h : Trj \to H$	-- harm along a trajectory

"Possible" future:

$[trj_1, ..., trj_n]$ scenarios

or

$[(trj_1, p_1), ..., (trj_n, p_n)]$ stochastic uncertainty

or

fuzzy, finitely additive probabilities, . . .

or

combinations thereof

"Possible" future: $F\ Trj$ for a functor F

$p : State \to F\ Trj$

$F\ h \circ p : State \to F\ H$ possible future harm

Vulnerability is a measure of this possible future harm:

$m : F\ H \to V$ for some preorder V

$$v : S \to V$$
$$v = m \circ F\ h \circ p$$

In Haskell, using "user-friendly" names:

$$vulnerability = measure \circ fmap\ harm \circ possible$$

Formalization as DSL creation.

Monotonicity condition for vulnerability measures:

For all $f : H \to H$ such that f is increasing, for all $fh : F\ H$, we have

$$m\ fh \sqsubseteq m\ (F\ f\ fh)$$

or, if $\sqsubseteq$ is partial

$$\neg\ (m\ (F\ f\ fh) \sqsubset m\ fh) \quad \text{-- for suitably defined } \sqsubset$$

If every possible harm is increased, the total measure of possible harm should not be decreased.

$H = Nat, F = List$

1. $V = Nat, m = maximum$ works
2. $V = Real, m = average$ works
3. $V = Nat, m = most\ frequent\ value$ fails

$H = Nat, F = SP$

1. $V = Nat, m = maximum\ value$ works
2. $V = Real, m = expected\ value$ works
3. $V = Nat, m = likeliest\ value$ fails

Testing the monotonicity condition using QuickCheck:

$$testMonotonicity\ measure\ geninc\ fh = forAll\ geninc$$
$$(\lambda inc \rightarrow ((measure\ fh) \sqsubseteq (measure\ (fmap\ inc\ fh))))$$

Problems:

we need to create a custom generator for every harm type (and every preorder on the harm type)

pretty bad coverage for high dimensions

Testing the monotonicity condition:

can fail because of overflow or round-off errors

hard to distinguish between “conceptual” and “implementational” errors

Formulating the monotonicity condition in Agda is like writing the tests in QuickCheck.

$$
\begin{array}{l}
\mathit{record}\ \mathit{Preorder}\ (A : \mathit{Set}) : \mathit{Set}\ \mathbf{where} \\
\quad \mathit{field} \\
\quad\quad \mathit{le} : A \to A \to \mathit{Bool} \\
\quad \mathit{LE} : A \to A \to \mathit{Set} \\
\quad \mathit{LE}\ a_1\ a_2 = \mathit{lift}\ (\mathit{le}\ a_1\ a_2) \\
\quad \mathit{field} \\
\quad\quad \mathit{reflexive} : (a : A) \to \mathit{LE}\ a\ a \\
\quad\quad \mathit{transitive} : (a_1\ a_2\ a_3 : A) \to \\
\quad\quad\quad \mathit{LE}\ a_1\ a_2 \to \mathit{LE}\ a_2\ a_3 \to \mathit{LE}\ a_1\ a_3
\end{array}
$$

$$
\begin{array}{l}
\mathit{record}\ \mathit{Functor}\ (F : \mathit{Set} \to \mathit{Set}) : \mathit{Set1}\ \mathbf{where} \\
\quad \mathit{field} \\
\qquad \mathit{fmap} : \{A\ B : \mathit{Set}\} \to (A \to B) \to (F\ A \to F\ B) \\
\qquad \ldots. \\
\mathit{LE} : \{A : \mathit{Set}\} \to (P : \mathit{Preorder}\ A) \to (a_1\ a_2 : A) \to \mathit{Set} \\
\mathit{LE}\ P\ a_1\ a_2 = \mathit{lift}\ (\mathit{le}\ a_1\ a_2)\ \mathbf{where}\ \mathit{le} = \mathit{Preorder.le}\ P \\
\mathit{record}\ \mathit{Increasing}\ (A : \mathit{Set})\ (P : \mathit{Preorder}\ A)\ (f : A \to A) : \mathit{Set}\ \mathbf{where} \\
\quad \mathit{field} \\
\qquad \mathit{increasing} : (a : A) \to \mathit{LE}\ P\ a\ (f\ a)
\end{array}
$$

$$
\begin{array}{l}
\mathit{record}\ \mathit{VulnMeasure}\ (H\ V : \mathit{Set})\ (\mathit{PH} : \mathit{Preorder}\ H)\ (\mathit{PV} : \mathit{Preorder}\ V) \\
\quad (F : \mathit{Set} \to \mathit{Set})\ (\mathit{Func} : \mathit{Functor}\ F)\ (m : F\ H \to V) : \mathit{Set}\ \mathbf{where} \\
\quad \mathit{fmap} = \mathit{Functor.fmap}\ \mathit{Func} \\
\quad \mathit{field} \\
\qquad \mathit{mon} : (\mathit{inc} : H \to H) \to (\mathit{Increasing}\ H\ \mathit{PH}\ \mathit{inc}) \to (\mathit{fh} : F\ H) \to \\
\qquad\quad \mathit{LE}\ \mathit{PV}\ (m\ \mathit{fh})\ (m\ (\mathit{fmap}\ \mathit{inc}\ \mathit{fh}))
\end{array}
$$

It is easy to construct a term of type

$$VulnMeasure\ H\ H\ P\ P\ List\ ListFunc\ maximum$$

for any harm type H.

Similarly for $average$, $expected$, etc. (with appropriate assumptions on H).

In constructing these proofs, some properties will be *postulated* (e.g. associativity of addition on H) which do not hold, e.g. of $Float$.

Usually, different models for possible trajectories of the same system have the same state space S and the same trajectory space Trj, but differ in the choice of F, e.g. some give possible scenarios ($List\ Trj$), others stochastic information ($SP\ Trj$).

When are two vulnerability assessments with models of different types "compatible"?

The harm evaluation stays the same ($h : Trj \to H$), but the domain of the vulnerability measure $m : F\ H \to V$ has to change.

Intuitively speaking, $m_1 : F\ H \to V$ and $m_2 : F\ H \to V$ are compatible if they rank things in the same way:

$$\forall\ fh_1\ fh_2 : F\ H$$
$$m_1\ fh_1 \sqsubseteq m_1\ fh_2 \equiv m_2\ fh_1 \sqsubseteq m_2\ fh_2$$

i.e. if the induced preorders on $F\ H$ are order-isomorphic.

We can "reuse" this idea if one of the representations of "possible" can be embedded in the other, i.e. if there exists an *injective* natural transformation from one to the other.

Let F_1, F_2 be two functors, $\tau : F_1 \to F_2$ an injective natural transformation, H a set, $\sqsubseteq_1$ and $\sqsubseteq_2$ preorders on $F_1\ H$ and $F_2\ H$ respectively. Then $\sqsubseteq_1$ and $\sqsubseteq_2$ are compatible with respect to τ iff

$$\forall\ x_1\ x_2 : F_1\ H$$
$$x_1\ \sqsubseteq_1\ x_2 \equiv \tau\ x_1\ \sqsubseteq_2\ \tau\ x_2$$

The ability to easily formulate and prove high-level conditions is another advantage of using a (dependently typed) functional programming language for formalization of concepts.

Ideally, it would be as easy to prove these kind of properties as it is to implement QuickCheck tests.

But. . .

. . . we're not there yet:

1. We lack a good enough tutorial

2. Agda standard library is intimidating

3. Not enough experience with proof reuse