Open games in practice

Jules Hedges
(Max Planck Institute for Mathematics in the Sciences)

Philipp Zahn
(University of St. Gallen)

ACT 2020
This talk: category theory, compositionality, ??, profit!!!
Zanzi #BlackLivesMatter, Silence is Complicity @tangled_z... · 2m
You forgot the prelude slide of "If you're not familiar with catamorphic zygozorphisms already then I'm sorry I don't think I can convey the intuition in just a few slides, and I suggest getting a head start on the refreshments in the break room"

julesh
@_julesh_

Replying to @tangled_zans

This but unironically, anybody who missed the keynote on mixed optics will probably be in for a rough time
Mixed optics recap

\[
\text{Optic} \left( \begin{pmatrix} S \\ T \end{pmatrix}, \begin{pmatrix} A \\ B \end{pmatrix} \right) = \int_{M \in \mathcal{M}} \mathcal{C}(S, M \cdot A) \times \mathcal{D}(M \cdot B, T)
\]
Optics in this talk

\[ D = (\text{f.s.}) \text{ probability monad on } \textbf{Set} \]

forwards category = kleisli category of \( D \)
\[ \text{= category of (f.s.) Markov kernels} \]
acts by “lift”

backwards category = kleisli category of

\[ T(X) = \mathbb{R}^N \rightarrow D(\mathbb{R}^N \times X) \]

state monad transformer, state = payoff vector
applied to \( D \)

(nasty rather nice hack)
Open games recap

An open game \( \left( \begin{array}{c} S \\ T \end{array} \right) \rightarrow \left( \begin{array}{c} A \\ B \end{array} \right) \) consists of:

1. A set \( \Sigma \) of strategy profiles

2. A \( \Sigma \)-indexed family of optics \( \left( \begin{array}{c} S \\ T \end{array} \right) \rightarrow \left( \begin{array}{c} A \\ B \end{array} \right) \)

3. For each \( \Sigma \) and context, a “valuation” (list of deviations)

\[
\overline{\text{Optic}} \left( \left( \begin{array}{c} S \\ T \end{array} \right), \left( \begin{array}{c} A \\ B \end{array} \right) \right) = \int^{\Theta} D(\Theta \times S) \times (A \rightarrow T(B))
\]
Haskell library for monoidal category of open games

Implementation of string diagrams
(too much work)

DSL = ad-hoc text-based input method

equilibrium checking
(future work)

analytics

automatic solving
(impossible)
The pipeline

• Code in a domain specific language (DSL) + auxiliary Haskell code describes game

  DSL-to-Haskell compiler

• Haskell code importing open games backend library

  Load in interactive Haskell prompt (GHCi)

• Interactive model
Worked example

• 1 carbon credit = legal right to emit 1000 kg CO$_2$

• Total credits capped by emissions target

• How to allocate credits to producers?

• Mechanism design: We would like to design the rules to produce a “good” outcome (e.g. avoiding perverse incentives)

• Our goal: rapid prototyping of models
Random allocation

Fixed price sale

Auction

Grandfathering

Producers

Resale market

Credits consumed

= flow of credits
Baby steps

- Fix 2 firms,
- Fix 2 credits to be allocated
Random allocation

\[
\text{randomAllocation} = \ \backslash \text{game} \rightarrow \\
a_1, a_2 \leftarrow \text{nature allocation} \\
\text{returnG} \leftarrow a_1, a_2
\]

lift a distribution
Supplementary Haskell:

allocation :: Stochastic (Int, Int)
allocation = do a1 <- uniform [0..2]
              return (a1, 2 - a1)
Fixed price allocation

fixedAllocation = \game v1, v2 ->
    ask1 <- decision "player1" [0..2] <- v1 | -3*a1
    ask2 <- decision "player2" [0..2] <- v2 | -3*a2
    a1, a2 <- function allocatePermits <- ask1, ask2
    returnG <- a1, a2

Supplementary Haskell:

allocatePermits :: (Int, Int) -> (Int, Int)
allocatePermits (ask1, ask2)
  | (ask1 + ask2 <= 2) = (ask1, ask2)
  | (otherwise)        = (1, 1)
VCG auction

vcgAllocation = \game v1, v2 ->
  bid1 <- decision “player1” [0..5] <- v1 | -pays1
  bid2 <- decision “player2” [0..5] <- v2 | -pays2
  a1, pays1, a2, pays2 <- function auctioneer <- bid1, bid2
  returnG <- a1, a2

Supplementary Haskell (2nd price auction):

auctioneer :: (Int, Int) -> (Int, Double, Int, Double)
auctioneer (bid1, bid2)
  | (bid1 == bid2) = (1, bid2/2, 1, bid1/2)
  | (bid1 >  bid2) = (2, bid2, 0, 0)
  | (bid1 <  bid2) = (0, 0, 2, bid1)
Production game

Input: valuations + allocated credits

```
production = \game v1, v2, a1, a2 ->
  c1 <- decision "player1" [0..a1] <- v1, a1 | v1*c1
  c2 <- decision "player2" [0..a2] <- v2, a2 | v2*c2
returnG <- a1 - c1, a2 - c2
```
The story so far:

\[
\text{game1} = \text{\game} \rightarrow \\
\quad v1 \leftarrow \text{nature (uniform [1..5])} \\
\quad v2 \leftarrow \text{nature (uniform [1..5])} \\
\quad a1, a2 \leftarrow \text{vcgAllocation} \leftarrow v1, v2 \\
\quad _, _ \leftarrow \text{production} \leftarrow v1, v2, a1, a2
\]

(At this stage we can do equilibrium analysis)
Resale market

(skipped for time)
Hooking it together

game2 = \game ->
  v1 <- nature (uniform [1..5])
  v2 <- nature (uniform [1..5])
  a1, a2 <- vcgAllocation <- v1, v2
  b1, b2 <- production <- v1, v2, a1, a2
  c1, c2 <- resaleMarket <- v1, v2, b1, b2
  _, _ <- production <- v1, v2, c1, c2
Notes

• Equilibrium analysis skipped for time!

• ... but it was the purpose of the whole exercise

• Repository: https://github.com/jules-hedges/open-games-hs

• This example: https://github.com/jules-hedges/open-games-hs/blob/permitSale/src/OpenGames/Examples/PermitSale/PermitSale.hs