Logical Dependence
via
Functional Dependence

Paulo Oliva
Queen Mary University of London
Main point:

If we want to capture logical dependence via functional dependence in a modular fashion we must work with higher-types.
logic  function
modular  higher-types
- From logic to functionals
- No-counterexample interpretation and the Dialectica interpretation
- Selection functions
- Higher-order game theory
- Pirates and coins
The axiom of choice expresses this translation from logical dependence to functional dependence.

\[ \forall x \exists y A(x,y) \rightarrow \exists f \forall x A(x,f(x)) \]

Truth/proof of premise implies the existence of a function witnessing the conclusion.
\[ \forall x \exists y A(x, y) \rightarrow \exists f \forall x A(x, f(x)) \]

- Is \( A(x, y) \) decidable? Not necessarily
- Classical or constructive logic?
- What are the types of \( x \) and \( y \)? Any
- Existence of \( f \) in which universe/model? Primitive recursive functionals
Consider the infinite pigeon-hole principle:
If you colour the natural numbers with finitely many colours then one colour must be used infinitely often.

∀ n, c \mathbb{N} \rightarrow [n] ∃ i < n ∃ k > j (c(k) = i)

proof by induction and classical logic

recursive? no!

\forall j \exists k > j (c(k) = i)

colour i is used infinitely often
∀ n, c \in \mathbb{N} 
\exists i < n \forall j \exists k > j (c(k) = i)

∀ n, c \in \mathbb{N} \rightarrow [n] 
\exists i < n \exists k > g(i) (c(k) = i)

recursive?

yes! (even primitive recursive)

no counter-example interpretation

counterexample function

recursive?
no counter-example interpretation

- Kreisel (1951)

- Not modular! (Kohlenbach 1999)

Dialectica interpretation

- Gödel (1958) - ideas from 1930s

- Modular

- Interprets Markov principle, but not full CL

Dialectica interpretation + negative translation
∀ \(n, c\) \(\in \mathbb{N} \rightarrow [n]\)

∃ \(i < n\)

∀ \(j\) ∃ \(k > j\) \((c(k) = i)\)

negative translation

∀ \(n, c\) \(\in \mathbb{N} \rightarrow [n]\)

¬ ∀ \(i < n\) ¬ ∀ \(j\) ¬ ∀ \(k > j\) ¬ \((c(k) = i)\)
∀ \( n, c \in \mathbb{N} \rightarrow [n] \)

\[ \neg \forall i < n \neg \exists i < n \exists p \in \mathbb{N} \rightarrow \mathbb{N} \]

\[ p(\varepsilon_i p) > \varepsilon_i p \land c(p(\varepsilon_i p)) = i \]

\text{dialectica interpretation}
∀ n, c \in \mathbb{N}^{\mathbb{N} \to [n]} \exists \varepsilon \in \mathbb{N}^{\mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}} \exists i < n \exists p \in \mathbb{N}^\mathbb{N}
\begin{align*}
p(\varepsilon_i p) &> \varepsilon_i p \\
\land c(p(\varepsilon_i p)) &= i
\end{align*}

- selection functions = players
- game context
- move of i-th player in context p
- game outcome of given move
Higher-order infinite pigeon-hole principle:

Given $n$ players $\varepsilon_i$, $0 < i < n$, and an assignment of players to numbers, $c : \mathbb{N} \rightarrow [n]$, there exists a player $i$ and a game context $p$ such that the outcome bounds the player's move, and the player is assigned to the outcome index.
“In the field of analysis, it is common to make a distinction between “hard”, “quantitative”, or “finitary” analysis on one hand, and “soft”, “qualitative”, or “infinitary” analysis on the other....The finitary version of an infinitary statement can be significantly more verbose and ugly-looking than the infinitary original, and the arrangement of quantifiers becomes crucial.”

–Terence Tao
(https://terrytao.wordpress.com/2007/05/23/)
Higher-order Games

\[ \text{outcome} = q(x_1, x_2, x_3, x_4) \]
Higher-order Games

context = mapping from move to outcome
player = mapping from context to best moves
context = mapping from move to outcome
player = mapping from context to best moves

move = $X$
outcome = $R$
context = $X \rightarrow R$
player = $(X \rightarrow R) \rightarrow X$

$J_R X$
selection functions \( J_R X \)

return :: \( X \rightarrow J_R X \)

bind :: \( J_R X \rightarrow (X \rightarrow J_R Y) \rightarrow J_R Y \)

sequence :: \([J_R X] \rightarrow J_R [X]\)

Surprising fact: sequence is total even on infinite lists! (for discrete types \( R \))
Pirate Example
Pirates and Treasures

A group of 7 pirates has 100 gold coins. They have to decide amongst themselves how to divide the treasure, but must abide by pirate rules:

1. The most senior pirate proposes the division.
2. All of the pirates (including the most senior) vote on the division.
   - If half or more vote for the division, it stands.
   - If less than half vote for it, they throw the most senior pirate overboard and start again.
3. The pirates are perfectly logical, and entirely ruthless (only caring about maximising their own share of the gold).

What division should the most senior pirate suggest to the other six?

1 http://www.ox.ac.uk/admissions/undergraduate/applying-to-oxford/interviews/sample-interview-questions
modelling the problem

outcome of the game? partition of n coins amongst pirates

moves of players?

when most senior: partition of n coins amongst k remaining pirates

when less senior: agree with proposed partition or not?
-- Share contains the distribution of coins for each pirate type
  type Pirate = Int

-- Share contains the distribution of coins for each pirate type
  type Share = [Int]

-- Whether a pirate agrees with share
  type Vote = Bool

-- Collection of all votes
  type Poll = [Vote]

-- Number of coins
  nc = 5

-- Number of pirates
  np = 5
helpful functions...

-- All possible ways to divide n coins amongst i pirates

divide :: Int -> Int -> [Share]
divide n 1 = [[n]]
divide n i = [ k:xs | k <- [0..n], xs <- divide (n-k) (i-1) ]

-- Generic argmax
argmax :: (Ord b) => [a] -> (a -> b) -> [a]
argmax d p = [ x | x <- d, p x == maximum [ p x | x <- d ] ]

-- Generic argmax with parallel map
argmax :: (NFData a, NFData b, Ord a, Ord b) => [a] -> (a -> b) -> [a]
argmax d p = [ x | (v,x) <- graph, v == (fst.last $ graph) ]
  where graph = sort $ parMap rdeepseq (\x -> (p x, x)) d
Basic player 1: The voter

Input

- Pirate index $i$
- Continuation $p :: \text{Bool} \to \text{Share}$

Choose boolean that maximises his share

\[
\begin{align*}
\text{v} &:: \text{Pirate} \to (\text{Bool} \to \text{Share}) \to \text{Bool} \\
v \ i \ p &= \text{head} \space \text{argmax} \ [\text{True},\text{False}] \ (((!!i).p) \\
\text{sv} &:: \text{Pirate} \to J \text{ Share} \text{ Bool} \\
\text{sv} \ i &= J(v \ i)
\end{align*}
\]
Basic player 2: The sharer

Input

- Pirate index $i$
- Continuation $p :: \text{Share} \to \text{Share}$

Choose global share that maximises his share

```haskell
s :: Pirate -> (Share -> Share) -> Share
s i p = head $ argmax dom (!!i).p
  where shares = divide nc (np - i)
    dom = map ((replicate i 0)++) shares

ss :: Int -> J Share Share
ss i = J (s i)
```
Composing players...

**Poll player** = sequencing of voters

\[
\text{sp} :: \text{Pirate} \rightarrow \text{J Share Poll} \\
\text{sp} \ i = \text{sequence} \ (\text{map sv} \ [(i+1)..(n-1)])
\]

**Round player** = Product of share and poll players

\[
\text{e} :: \text{Int} \rightarrow \text{J Share} \ (\text{Share}, \ \text{Poll}) \\
\text{e} \ i = \text{prod} \ (\text{ss} \ i, \ \text{sp} \ i)
\]

**Global player** = Sequence of round players

\[
\text{g} :: \text{J Share} \ [(\text{Share}, \ \text{Poll})] \\
\text{g} = \text{sequence} \ (\text{map e} \ [0..(n-1)])
\]
Outcome function

Determines outcome of the game

Given list of [(Share, Poll)] I return first share that was agreed in poll

If none, take last share

\[
q :: [(\text{Share, Poll})] \rightarrow \text{Share}
q [(s,v)] = s
q ((s,v):xs) = \text{if } 1 + \text{pro} \geq \text{con} \text{ then } s \text{ else } q \text{ xs}
\text{where } [\text{con,pro}] = (\text{map length}).\text{group}\cdot\text{sort } \{ \text{True:False:v}\}
\]
Demo

- Number of pirates = 5
- Number of coins = 5

MBP{paulo}: ghc -O2 -threaded --make pirates.hs
[1 of 1] Compiling Main                                     ( pirates.hs, pirates.o )
Linking pirates ...

MBP{paulo}: time ./pirates +RTS -N
[3,0,1,0,1]
real 19m5.150s
user 138m18.802s
sys  4m19.996s

Optimal division proposed by most senior pirate
Modularity...

Solution built of individual/independent parts

More pirates / coins
(change values of np / nc)

Changes in voting rule
e.g. 1/3 enough
(change outcome function q)

Changes in "goal" of pirates
e.g. happy with 1 coin
(change selection functions)
Some references...


