Jacob's Ladder

Arnaud Bailly - Christophe Thibaut

01/12/2011

Category Theory Crash course

- What is a category?

"Well-known" (or not so well-known) Categories

- Some examples of Categories: Set, monoid, poset…
- Hask: Category of Haskell types
- CAM: Categorical abstract machine, a practical compilation scheme for ML languages based on standard categorical constructs

Warmer: Deconstructing a function categorically

Rules: - Use only arrows and objects' names (ie. do not describe the internal structure of objects) - Assume we work in a well-behaved category - Refine a function till you drop

Constructions In Categories

Product: Tupling objects and functions

- Down the ladder: Define records and structures, apply functions on them

Sums: Choosing between alternatives

- Down the ladder: Error handling, making partial functions total
- Eg. Exceptions in Java/C#

Special objects: 0 and 1

- *0*: Initial object, one and only one arrow from 0 to any other object
- *1*: Terminal object, one and only one arrow from any other object to 1
- Using 1 as a *selector* for "elements" in objects (eg. members of a set)
- Uniqueness up-to isomorphisms

Exponential

- representing *function abstraction* in a category
- let A, B be objects, we can form BˆA the *exponential object* and a function eval : BˆA x A -> B s.t. for any f: C x A -> B, f factorizes uniquely through eval:
  $f = ev\ (f\times)$
- This also says that one can form f: C -> BˆA the function that that transforms a two-argument function (here C x A -> B) into a function from one argument to another function. We cannot however leave A in the equation is it provides the basis for the computation of the limit (??)

Categorical Abstract Machine

- Representing computation within a category : The Categorical Abstract Machine (Cousineau et al., 1987) allows compiling *strict* functional programs into an intermediate form suitable for compilation to low-level languages (eg. C, assembly…)
- Rests on the concept of a cartesian closed category: A category with all limits and exponentiation.

Case Study: A proxy HTTP server

- Define an HTTP Service for retrieving content of files from a uid stored in a repository

Constructions Over Categories

- Up the ladder: applying categorical concepts to categorical constructs
- Yes, the category of categories exists! (with some restrictions…)

Functors

- Mappings from objects to objects s.t. composition of functions is preserved f: a -> b => Ff : Fa -> Fb
- Down the ladder: Defining parametric data structures (eg. type constructors) like Lists, Sets, Streams…
- Exercise: The functor of functions with errors

Applicative Functors

- <$> : a -> b -> (f a -> f b) : this is standard "functor application"
- <*> : f (a -> b) -> (f a -> f b) : allows working on multiple arity functions, *push argument application inside functor*

Monads

- Article from Erik Meijer about LINQ: http://cacm.acm.org/magazines/2011/10/131398-the-world-according-to-linq/fulltext
- LINQ uses monads' *flatMap* (aka. *bind*) to construct "queries" over various datatypes, including relational structures
- the various keywords of the SQLish language (SELECT, FROM, WHERE) are just instances of (a -> m b) which are chained in the monad

Natural transformations

- Mappings *between functors*
- Down the ladder: Functions on lists that change the "shape" of the list without changing the elements in it (reverse, inits, tails…)
- Down the ladder:

- I/O on HTTP messages, REST interface of a service
- A transformation need to be an isomorphism (ie. invertible), it may "forget" some aspects of the functors it maps

Conclusion

Relating categorical thinking to agile design principles: - *Simple design*: only arrows and (abstract) objects, no patterns, no frameworks, no fancy coding tricks - *No BDUF*: Designing functions first: What will you do with your data/objects?, refrain from defining the implementation first, abstract from details to prevent data lock-in - *Removing duplication*: Generalizing to higher-levels allow DRY to apply not only on functions but also on control structures and "shape" of the system - Focusing on the *Domain*: First define core arrows, those that are part of the domain you are working on ; then apply functors and transformations to *lift* from core domain to a more complex behaviours (error handling, logging, interfacing with the outside world)

Caveats: - Do not talk of this conference to your local mathematician, its hair might go white earlier than expected! - True Cat.Th. is insanely complex, it has connections to all of mathematics (logic, topology, algebra, analysis) and is very abstract with lot of weird constructions

Reference

- Scalaz: Library in scala, provide a wealth of categorical constructions for scala programs
- Category in java: representing cat.th. in JAva
- Haskell: pioneer in applied cat.th.
- http://hseeberger.wordpress.com/2011/01/31/applicatives-are-generalized-functors/
- http://aabs.wordpress.com/2008/05/29/functional-programming-lessons-from-high-school-arithmetic/

Credits

- Bath's Jacob's Ladder
- Augsburg Barfüßerkirche