What are Bit Strings? The View from Process

Nick Rossiter

Northumbria University

Outline

- Process as a Monad/Comonad
- Underpinning by Cartesian Closed Category
  - Adjointness
  - Composition
  - Product/Exponentiation
  - Finite products
- Generation of Strings
  - Kleisli Category
  - Free Monoids

Current State of Play

- Process
  - Viewed as monad/comonad
  - Three cycles in each direction:
    - One reflective – monad
    - Other anticipatory – comonad
  - Handles transaction concept
    - In databases (ACID)
    - In universe

Example of Adjointness

$$\begin{tikzcd} L \arrow[r, shift left=2, "F"] & \mathbf{R} \arrow[l, shift left=2, "G"] \end{tikzcd}$$

If conditions hold, then we can write F ⊣ *G*
The adjunction is represented by a 4-tuple:
<F,G,η, ε>
η and ε are unit and counit respectively

Endofunctor T = GF

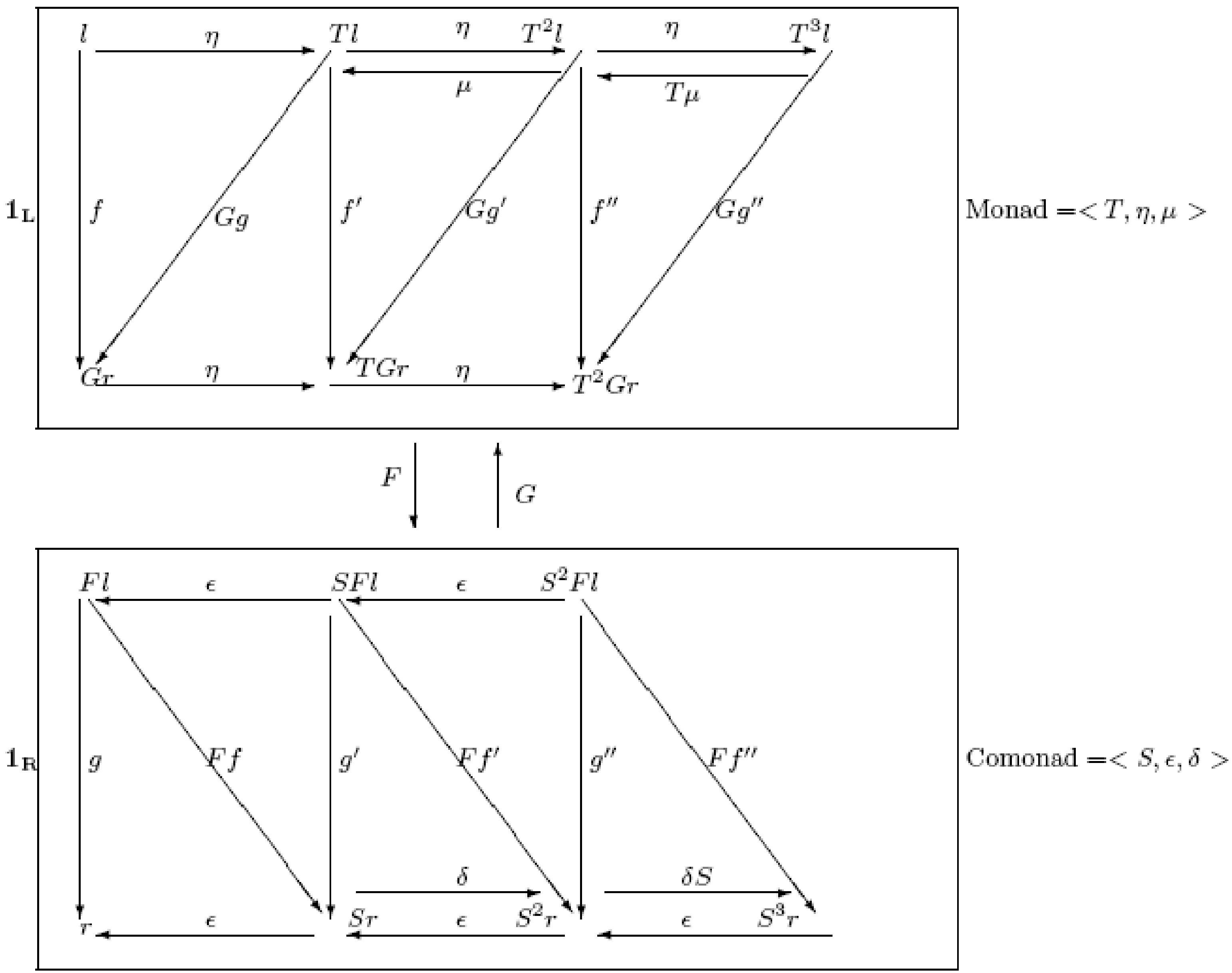

Fig. 2. After three cycles $GFGFGF$ from left-hand category and three cycles $FGFGFG$ from right-hand category: η and δ map onto other than $\perp$, $\top$ maps onto other than ϵ and μ

Cartesian Closed Category (CCC)

- Underpins applied category theory
- Basis of many fundamental structures in applications
  - Partial order
  - Boolean/Heyting algebras
  - Pullbacks/ Pushouts
  - Scott domains
- Also emerges in lambda calculus
- In computing functions become first-class data
  - Functional programming languages
  - Database design (normalisation)

Definition of CCC paraphrased

- CCC-1 There is a terminal object 1
- CCC-2 Each pair of objects has a product with projections
- CCC-3 There is only one path between the product and the related objects.

In more detail: CCC-1

- For every object A in the category, there is exactly one arrow $A \rightarrow T$
  - T is the terminal object
- Category is closed on top T

CCC-2

- Each pair of objects *A* and *B* of the category has a product *A X B* with projections

  π_l: *A X B* → *A*

  π_r: *A X B* → *B*

- Category has products and projections
  - Giving route to relationships

CCC-3a

- Notion of currying: change function on two variables to a function on one variable
- For function $f: C \times A \rightarrow B$
- Let $[A \rightarrow B]$ be set of functions from A to B
- Then there is a function:

  $\lambda f: C \rightarrow [A \rightarrow B]$

  where $\lambda f(c)$ is the function whose value at an element $a \in A$ is $f(c,a)$

- Equivalent to typed lambda calculus
- Examples:

  f : multiply(_,2) → R curries to λf: double(_) → R

  f : exponentiate(_,2) → R curries to λf : square(_) → R

CCC-3b

- For every pair of objects *A* and *B*, there is an object $[A \rightarrow B]$
  and an arrow eval: $[A \rightarrow B] \, X \, A \rightarrow B$
  with the property that for any arrow
  $f: C \, X \, A \rightarrow B$ (C is product object)
  there is a **unique** arrow
  $\lambda f: C \rightarrow [A \rightarrow B]$
  such that the following diagram commutes:

CCC-3c

$$
\begin{tikzcd}
C \times A \arrow[rr, "\lambda f \times A"] \arrow[d, "f"'] & & {[A \rightarrow B] \times A} \arrow[dll, "eval"] \\
B & &
\end{tikzcd}
$$

One path from product

$[A \rightarrow B]$ is termed B^A : all arrows from A to B, A is the exponent of B

Uniqueness

- The category is CCC if (other conditions satisfied) and :
  - *λf* is unique (one path)
- Notes
  - eval is also a function
  - eval: $[A \rightarrow B] \rightarrow B$

  refers to one *A* object and its associated *B* object

  -- eval: $[A \rightarrow B] \times A \rightarrow B$

  refers to all *A* objects and their associated *B* objects

Finite Products

- CCC is not restricted to binary products
- Can have finite products
- For any objects $A_1, \ldots, A_n$ and A of a CCC and any $i=1,\ldots,n$, there is an object $[A_i \rightarrow A]$ and an arrow:

  eval $: [A_i \rightarrow A] \times A_i \rightarrow A$
- such that for any $f: \Pi A_k \rightarrow A$, there is a unique arrow:

  $\lambda_i f : \Pi A_k \rightarrow [A_i \rightarrow A] \qquad (k > 1)$

Finite products give construction of n-tuples which can represent strings.

Note: this notation may offend Gödel's theorems!

Abstract View of CCC

- An adjoint relationship
  - $F \dashv G$
  - Free functor F creates binary products
  - Underlying functor G checks for exponentials (one path)

Adjoints

- Left adjoint -- free functor on category **C**:

  $\_ \ X\ A$: **C** → **C**

  For fixed object *A* and an object *B*, this gives binary product *B X A* and an arrow:

  - $f \times id_A : B\ X\ A \rightarrow C\ X\ A$

- Right adjoint – underlying functor G on value of object *C* on right-hand side:
  - Unique arrow $\lambda f : B \rightarrow G(C)$ such that $eval \circ (\lambda f\ X\ A) = g$

- Adjointness requires both left and right adjoints to exist

Composition for there to be a Right Adjoint

$$
\begin{tikzcd}
B \times A \arrow[rr, "\lambda f \times A"] \arrow[d, "g"'] & & \mathit{G(C)} \times A \arrow[dll, "\mathrm{eval}"] \\
C & &
\end{tikzcd}
$$

One path from product

Compositions for Adjointness with unit/counit

$$
\begin{tikzcd}
A \arrow[r, "\eta"] \arrow[dr, "f"'] & G(\_XA(A)) \arrow[d, "Gg"] \\
 & GC
\end{tikzcd}
$$

$$
\begin{tikzcd}
 & \_XA(G(C)) \arrow[d, "\varepsilon"] \\
\_XA(B) \arrow[ur, "\_XA(f)"] \arrow[r, "g"'] & C
\end{tikzcd}
$$

Unit of adjunction η

Counit of adjunction ε

ε is eval
_XA(G(C)) is GCXA
_XA(B) is BXA
_XA(f) is Fλf

Locally CCC

- Satisfied when:
  - The category **C** has pullbacks and either:
    - The pullback functor has a right adjoint OR
    - For every object *A* in **C**, the slice category **C**/*A* is cartesian closed
- Pullbacks express relationships over objects in a particular context
- Locally CCC provide more expressiveness in capturing the real world

Product vs Pullback

$$
\begin{tikzcd}
 & A \times B \arrow[dl, "\pi_l"'] \arrow[dr, "\pi_r"] & \\
A & & B
\end{tikzcd}
$$

Product and
projections

$$
\begin{tikzcd}
 & A \times_C B \arrow[dl, "\pi_l"'] \arrow[dr, "\pi_r"] & \\
A \arrow[dr] & & B \arrow[dl] \\
 & C &
\end{tikzcd}
$$

Pullback of A and B
in the context of C

Kleisli Category

- Free algebra
- Based on monad earlier **T** = <*T*, *η*, *μ*>
  - where *T* is endofunctor *GF* for adjoint functors $F \dashv G$
  - *η* is unit of adjunction $\eta : 1_A \rightarrow GFA$
  - *μ* is multiplication $\mu : GFGF \rightarrow GF$

    compares results of 2nd and 1st cycles
  - **T** is a category
  - *A* is an object in left-hand category

Kleisli Category 2

- In Kleisli category
  - **T** = <T, η, μ>
  - The arrows are substitutions
  - μ can be thought of as carrying out a computation
- For arrow $f : A \rightarrow B$
  - then $A \rightarrow TB$
  - where T defines the substitutions as functions

Kleisli example

- For $f : A \rightarrow TB$

  $A = \{g,h\}$ and $B = \{i,j,k\}$

  f(g) = cddc, $f(h)$ = ec
- $Tf: TA \rightarrow TTB$

  TA is for example string ‘ghhg’

  TTB is (cddc), (ec), (ec), (cddc) (concatenations)

  $\mu : TT \rightarrow T$ is ‘cddcececcddc’ → ‘ghhg’
- In the comonad:

  $\delta : T \rightarrow TT$ is ‘ghhg’ → ‘cddcececcddc’
- So we have string generation through substitution

Kleene Closure

- Given a set A:
  - The Kleene closure A^* of a set A is defined as
    - the set of strings of finite length of elements of A
- In adjointness terms:

  $F : A \rightarrow A^*$

  $G : A^* \rightarrow A$
- The closure is then GFA
- F is the free functor, adding structure
- G is the underlying functor, removing structure

Example

- $A = \{a, b, c, d, …., z\}$ (alphabet)
- $F(A) = A^*$ = all finite strings constructed from A by F
- $G(A^*)$ returns the alphabet
- The closure relies on adjointness
  - *F* can be free and open (all possibilities)
  - *G* can check for language rules

Example 2

- The adjoint (if it exists) is $\langle F, G, \eta, \varepsilon \rangle$
  - *F* constructs all possibilities
  - *G* applies the language rules
  - η defines the change from $A \rightarrow GFA$ in the alphabet
  - ε defines the change from $FGA^* \rightarrow A^*$ in the language

Summary

- Category Theory provides a number of routes for generating strings:
  - n-tuples through cartesian closed categories
  - String expansion through substitution as in Kleisli categories
  - String generation through free functors as in the Kleene closure