Turing machines and Coalgebras

Section 0 Preface

I'd like to start with a bit of personal background. I did my PhD here with Amnon, so it's a special pleasure to be here to help celebrate his birthday. Before I came to Canberra I was an undergraduate at UQ, and I was strongly influenced by the mathematical physicists there, particularly Mark Gould and Tony Bracken. I learned Quantum Field Theory from Tony with a friend of mine, Mark Dowling, who went on to do a PhD with Michael Nielsen in quantum information theory, while I came here to learn algebraic geometry from Amnon.

A few years ago I noticed some of the work that Mark did with Nielsen, along with Gu and Doherty, getting a lot of citations in the physics literature

- Nielsen, Dowling, Gu, Doherty "Quantum computation as geometry", Science (2006).

Their ideas are now influential in connection with the

- AdS/CFT correspondence e.g. volume (AdS) vs. complexity (CFT)
  (of processes creating states)

Unfortunately I do not understand AdS/CFT at all. But I found the hint of connections between logic and geometry intriguing. Now, while in general QFTs have no rigorous formulation, some special *topological* QFTs in low-dimensions have well-understood descriptions in terms of categories arising in algebraic geometry, and moreover this is an area I have been working in for years. So I became interested in the following question:

4

<u>Question</u> is there a good notion of "complexity" for algorithms constucting

morphisms / objects / functors

of triangulated (or DG, A_∞, ...) categories? These are (in CY cases)

states / boundary cond. / defects

of associated 2D TQFTs.

Some of the basic ideas necessary for studying this question exist, since the border between logic and category theory has been reasonably well explored. But complexity is not as well-studied in this context, nor is the linearity present in the above question, so some foundational work was required. This talk is a report on some of that work.

<u>Section 1</u> (The logic in linear algebra, see [M] in references)

- What is an algorithm? Say, for constucting <u>morphisms</u> in <u>Vect_k</u>.

- (<u>Vect_k</u>, $\otimes$, $\multimap$, $\mathbb{1}$) is closed symmetric monoidal, $V \multimap W := \mathrm{Hom}_k(V, W)$, e.g. it comes with the structure of <u>evaluation maps</u>

$$ev_{V,W} : V \otimes (V \multimap W) \longrightarrow W$$

Example $U, V, W \in \underline{\mathrm{Vect}_k}$

$$\begin{tikzcd}
\mathrm{Hom}_k(W,W) \arrow[d, "{\mathrm{Hom}(ev_{V,W},1)}"] & & 1_W \arrow[ll] \arrow[ddd, "\text{“meta-mathematical algorithm”}"] \\
\mathrm{Hom}_k(V\otimes(V\multimap W), W) \arrow[d, "{\mathrm{Hom}(ev_{U,V}\otimes 1, 1)}"] & & \\
\mathrm{Hom}_k(U\otimes(U\multimap V)\otimes(V\multimap W), W) \arrow[d, "\cong\ \text{adjunction}"] & & \\
\mathrm{Hom}_k((U\multimap V)\otimes(V\multimap W), U\multimap W) & & c_{U,V,W} \arrow[ll]
\end{tikzcd}$$

This diagram/construction/algorithm produces the <u>internal composition</u> c.
This algorithm is formalised in <u>linear logic</u> (the language of $\otimes, \multimap, \oplus, \ldots$) with <u>type variables</u> u, v, w standing for unknown vector spaces and <u>deduction rules</u> standing for available structural operations, as

$$\dfrac{\dfrac{}{u\vdash u}\,ax \qquad \dfrac{\dfrac{}{v\vdash v}\,ax \qquad \dfrac{}{w\vdash w}\,ax}{v,\ v\multimap w\vdash w}\,\multimap L}{\dfrac{u,\ u\multimap v,\ v\multimap w\vdash w}{u\multimap v,\ v\multimap w\vdash u\multimap w}\,\multimap R}\,\multimap L$$

"means" $1_w : w \to w$

"means" $ev_{v,w}$

"means" $c_{u,v,w}$

"formalised algorithm"

- This structure is called a <u>proof</u> (of $u\multimap v, v\multimap w \vdash u\multimap w$) or <u>algorithm</u>.
- Proofs modulo an equivalence relation give the <u>free</u> symmetric monoidal closed category on a set of objects $u, v, w, \ldots$ (Lambek & Scott).

<u>Def^n</u> <u>Programming</u> is the encoding of meta-mathematical algorithms into algorithms in a formal language (i.e. morphisms in a free category).

- Basic question: what can be programmed? (<u>expressiveness</u>)

- $\otimes, \multimap, \oplus$ is <u>not</u> very expressive, e.g. $\mathrm{End}_k(V) \to \mathrm{End}_k(V)$, $\alpha \mapsto \alpha^2$ cannot be programmed in it.

<u>Section 2</u> (Copying and coalgebras) [k char. 0, alg. closed]

- The reason we cannot program $\alpha \mapsto \alpha^2$ is that we have only allowed <u>linear</u> constructions in our basic language. To program <u>nonlinear</u> functions we need to <u>copy</u> vectors.

- All coalgebras are coassociative, counital, cocommutative.

$$\begin{tikzcd} \underline{\mathrm{Coalg}_k} \arrow[r, "\text{forget}"] \arrow[r, phantom, shift right=3, "\perp"] & \underline{\mathrm{Vect}_k} \arrow[l, bend left, "\text{cofree coalg } !(-)"] \end{tikzcd}$$

<u>Def^n</u> The <u>cofree coalgebra</u> over $V \in \underline{\mathrm{Vect}_k}$ is a pair $(!V, d)$ where $!V$ is a coalgebra, $d : !V \to V$ is linear and for any linear $\varphi : C \to V$ with C a coalgebra, there is a unique coalgebra morphism Φ making the following commute:

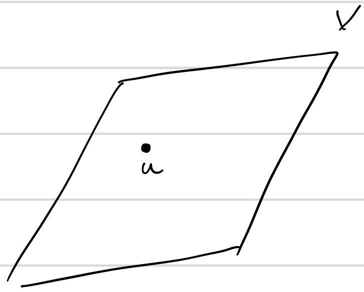
$$\begin{tikzcd} C \arrow[r, dashed, "\Phi"] \arrow[dr, bend right, "\varphi"'] & !V \arrow[d, "d"] \\ & V \end{tikzcd}$$

logana

5

Claim $!V$ is the universal way to copy vectors in V (Sweedler)

- **Group-likes**: $G(!V) := \{ c \in !V \mid \Delta(c) = c\otimes c \}$



$|\phi\rangle_u \longrightarrow G(!V) \cong \mathrm{Hom}_{\mathrm{Coalg}_k}(k, !V)$

vacuum at u

univ.

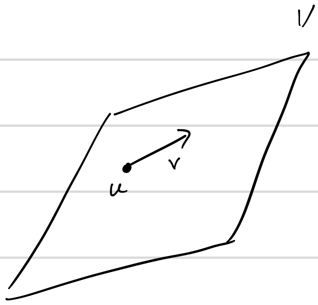
$d|\phi\rangle_u = u$ $\qquad \cong \mathrm{Hom}_k(k, V)$

$\cong V \longleftarrow u$

$\Delta|\phi\rangle_u = |\phi\rangle_u \otimes |\phi\rangle_u$

$|\phi\rangle_{u+v} \neq |\phi\rangle_u + |\phi\rangle_v$.

- **Primitive elements**: given $c \in G(!V)$, $P_c(!V) = \{ y \in !V \mid \Delta(y) = y\otimes c + c \otimes y \}$.
  and $P(!V) = \bigoplus_c P_c(!V)$.



$P(!V) \cong \mathrm{Hom}_{\mathrm{Coalg}_k}\left((k[x]/x^2)^*, !V \right)$

$|v\rangle_u$

$\rotatebox{90}{$\in$}$

$V \cong P_{|\phi\rangle_u}(!V)$

$\cong \mathrm{Hom}_k(k\cdot 1^* \oplus k\cdot x^*, V)$

$\cong V \times V \longleftarrow (u,v)$

Proposition $!V \cong$ space of finitely supported distributions on V $(k = \mathbb{R})$

$|\phi\rangle_u \longleftrightarrow$ Dirac distribution D_u at u

$|v\rangle_u \longleftrightarrow \sum_i v_i \frac{\partial}{\partial x_i} D_u$

Prop (Sweedler, M) $!V \cong \bigoplus_{u\in V} H^{\dim V}_{m_u}\left(\mathrm{Sym}(V^*), \Omega^{\dim V}_{\mathrm{Sym}(V^*)/k} \right) \cong \bigoplus_u \mathrm{Sym}_u(V)$

s.t. counit $!V \to k$ is $\sum_u \mathrm{Res}_u$.

"It should be noted that linear logic and differential linear logic were discovered by logicians in connection with logical questions, wholly separate from the categorical and algebraic considerations being used to motivate the def^{n}s here."

Def^n Linear logic (Girard '87) is the language of $\otimes, \multimap, \oplus, !$, i.e. we can use

$$!V \xrightarrow{d} V \qquad !V \longrightarrow k \qquad !V \xrightarrow{\Delta} !V \otimes !V$$

(dereliction) (weakening) (contraction)

$$\begin{tikzcd} !V \arrow[r, dashed, "\mathrm{prom}(\varphi)"] \arrow[dr, "\varphi"'] & !W \arrow[d, "d"] \\ & W \end{tikzcd}$$

(promotion)

logical terminology

In this language, squaring can be programmed:

Example For $\alpha \in \mathrm{End}_k(V)$

$$!\mathrm{End}_k(V) \xrightarrow{\Delta} !\mathrm{End}_k(V) \otimes !\mathrm{End}_k(V) \xrightarrow{d^{\otimes 2}} \mathrm{End}_k(V) \otimes \mathrm{End}_k(V) \xrightarrow{c} \mathrm{End}_k(V) \qquad (*)$$

$$|\phi\rangle_\alpha \longmapsto |\phi\rangle_\alpha \otimes |\phi\rangle_\alpha \longmapsto \alpha \otimes \alpha \longmapsto \alpha^2$$

This is the shadow of a proof in linear logic

$$\boxed{\text{Logic / syntax}} \xrightarrow{\text{semantics } [\![-]\!]} \boxed{\text{Algebra}}$$

$\underline{2}$

$$\dfrac{!(V \multimap V) \vdash V \multimap V}{\vdash !(V \multimap V) \multimap (V \multimap V)} \multimap R$$

"the squaring algorithm"

if $[\![V]\!] = V$ then $[\![V \multimap V]\!] = \mathrm{End}_k(V)$, $[\![!(V \multimap V)]\!] = !\mathrm{End}_k(V)$,

$[\![\underline{2}]\!] = (*)$ above.

In fact, the squaring (resp. cubing, ...) algorithm is how integers are encoded in linear logic:

<u>Def^n</u> $\underline{\mathrm{int}}_V := \,!(V \multimap V) \multimap (V \multimap V)$, $\underline{\mathrm{bint}}_V := \,!(V \multimap V) \multimap (\,!(V \multimap V) \multimap (V \multimap V))$.

<u>Lemma</u> There are functions

$$\mathbb{N} \longrightarrow \{\text{proofs of } \underline{\mathrm{int}}_V\}, \quad \{0,1\}^* \longrightarrow \{\text{proofs of } \underline{\mathrm{bint}}_V\}$$

<u>Example</u> $[\![\, \underline{011} \,]\!] \in [\![\, \underline{\mathrm{bint}}_V \,]\!] \cong \mathrm{Hom}_k(\,!\mathrm{End}_k(V) \otimes \,!\mathrm{End}_k(V), V \multimap V)$

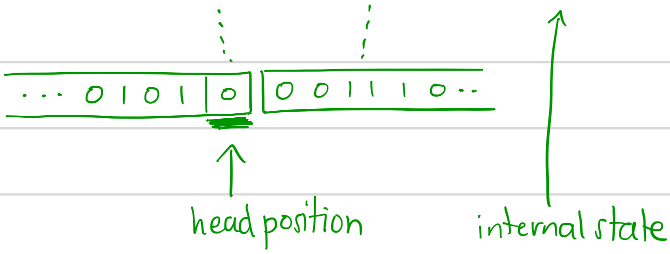
$$[\![\, \underline{011} \,]\!](\,|\phi\rangle_\alpha \otimes |\phi\rangle_\beta) = \overset{1}{\beta} \circ \overset{1}{\beta} \circ \overset{0}{\alpha}$$

So what can and cannot be programmed in linear logic (and thus realised in vector spaces using $\otimes, \multimap, \oplus$ and cofree coalgebras)? One way to approach this question is to try and encode <u>Turing machines</u>, which is our next topic.

Section 3 (Turing machines, see [C])

The state of a Turing machine can be encoded as a proof of (V any formula)

$$\underline{\mathrm{Tur}}_V = !\,\underline{\mathrm{bint}}_V \otimes !\,\underline{\mathrm{bint}}_V \otimes !_n\,\underline{\mathrm{bool}}_V \qquad \ulcorner {}_n\underline{\mathrm{bool}}_V = V^{\oplus n} \multimap V \lrcorner$$



Theorem (Clift-M, '17 based on Girard '95) There is an encoding of any Turing machine as a proof in (intuitionistic, first-order) linear logic

$$\begin{array}{c} \underline{\mathrm{step}}_V \\ \vdots \\ \underline{\mathrm{Tur}}_{V^{\oplus 3}} \vdash \underline{\mathrm{Tur}}_V . \end{array}$$

This proof simulates one time step, in the sense that it sends the encoding of the state of M after t steps to the encoding of the state after t+1 steps.

Remark
- Technical improvement on encoding of Girard in 2^{nd}-order LL, (Girard '95) as it is more "idiomatic" and has the advantage of being compatible with differential linear logic (Girard's encoding is not).

- The algorithm which iterates step cannot be programmed in (first-order) linear logic as we have presented it, but this can be programmed by going to second-order (which is very expressive, therefore).

So the first surprise here is: you can do quite a lot of nontrivial programming with just cofree coalgebras! (on top of $\otimes, \multimap, \oplus$). One might say this is a case of <u>logicians surprising algebraists</u>.

The second surprise is that some of the "geometric" structure of the cofree coalgebra can be reflected back into logic, with interesting results. This is a case of <u>algebraists surprising logicians</u>, and I want to focus on two examples: <u>derivatives</u>, and <u>nondeterminism</u>.

<u>Example</u> Consider the linear map $[1\,3\,1]$ and its promotion

$$
\begin{tikzcd}
!\operatorname{End}_k(V) \arrow[r, "{[1\,3\,1]}"] \arrow[dr, dashed, bend right, "\Phi"'] & \operatorname{End}_k(V) \\
& !\operatorname{End}_k(V) \arrow[u, "d"']
\end{tikzcd}
\qquad \Phi|\phi\rangle_\alpha = |\phi\rangle_{\alpha^2}
$$

With $k = \mathbb{R}$, view $M = \operatorname{End}_k(V)$ as a smooth manifold. Then for $\alpha \in M$,

primitive elements

$$
\begin{tikzcd}
P_\alpha(\,!\operatorname{End}_k(V)) \arrow[rr, "\Phi"] \arrow[d, phantom, sloped, "\cong"] & & P_{\alpha^2}(\,!\operatorname{End}_k(V)) \arrow[d, phantom, sloped, "\cong"] \\
\operatorname{End}_k(V) \arrow[d, phantom, sloped, "\cong"] & & \operatorname{End}_k(V) \arrow[d, phantom, sloped, "\cong"] \\
T_\alpha M \arrow[rr, "T_\alpha(x^2)"] & & T_{\alpha^2} M
\end{tikzcd}
$$

commutes, so in this case the induced map on <u>primitive elements</u> matches the map induced on tangent vectors by $x^2 : M \longrightarrow M$.

Def^n Differential Linear Logic (DLL, Ehrhard-Regnier) is the language of LL (i.e. $\otimes, \multimap, \oplus, !$ as coalg. + univ. property) plus

- the bialgebra structure on $!V$

$$\begin{array}{cc} !V \otimes !V \longrightarrow !V, & k \longrightarrow !V, \\ \text{(cocontraction)} & \text{(coweakening)} \\ |\phi\rangle_u \otimes |\phi\rangle_v \longmapsto |\phi\rangle_{u+v} & 1 \longmapsto |\phi\rangle_0 \end{array}$$

- tangent vectors / primitive elements

$$\begin{array}{cc} V \otimes !V \xrightarrow{D} !V & \left(\begin{array}{c} \text{equiv.} \\ V \longrightarrow !V \\ \text{(codereliction)} \end{array} \right) \\ \text{(deriving transform)} & \\ v \otimes |\phi\rangle_u \longmapsto |v\rangle_u & \end{array}$$

In this language, all algorithms can be differentiated! Including Turing machines. (see Clift's forthcoming master's thesis for examples of the latter).

Using the bialgebra structure allows for the programming of nondeterminism.

Theorem (Clift-M) There is an encoding of nondeterministic TMs as proofs in DLL of the sequent $!\,\underline{\mathrm{Tur}}_{V \oplus 3} \vdash !\,\underline{\mathrm{Tur}}_V$ (using cocontraction).

- There is an intrinsic characterisation of the complexity class P within 2^{nd}-order linear logic, due to Girard. We hope the above will give a similar characterisation of NP in 2^{nd}-order DLL.

Section 4 (Algorithms constructing objects)

$\mathcal{C}$ – k-linear category, $\mathcal{C} \multimap \mathcal{C}' := \mathrm{Cat}_k(\mathcal{C}, \mathcal{C}')$ – k-linear functors

Def^n $!\mathcal{C}$ is the category enriched over Coalg_k with $(!\mathcal{C})(a,b) = !\,\mathcal{C}(a,b)$.
There are canonical functors

$$!\mathcal{C} \xrightarrow{d} \mathcal{C}, \quad !\mathcal{C} \xrightarrow{\Delta} !\mathcal{C} \otimes_k !\mathcal{C}, \quad !\mathcal{C} \xrightarrow{\omega} k$$

making $!\mathcal{C}$ a "coalgebra" object in $\mathcal{C}at_k$.

One can find a shadow of linear logic in $(\mathcal{C}at_k, \otimes_k, \multimap, !)$, e.g.

Example The functor

$$!(\mathcal{C} \multimap \mathcal{C}) \xrightarrow{\Delta} !(\mathcal{C} \multimap \mathcal{C}) \otimes_k !(\mathcal{C} \multimap \mathcal{C}) \xrightarrow{d^{\otimes 2}} (\mathcal{C} \multimap \mathcal{C}) \otimes_k (\mathcal{C} \multimap \mathcal{C}') \xrightarrow{- \circ -} \mathcal{C} \multimap \mathcal{C}$$

sends a k-linear functor $F : \mathcal{C} \to \mathcal{C}$ to its square $F \circ F$.

Theorem (M, '17) There is a fully faithful functor

$$\bigoplus_{d \geq 0} \underbrace{\mathrm{rep}\, T^d_k}_{\substack{\text{strict polynomial} \\ \text{functors on } \underline{\mathrm{vect}_k} \text{ of} \\ \text{degree } d.}} \hookrightarrow \mathcal{C}at_k(!\underline{\mathrm{vect}_k}, \underline{\mathrm{vect}_k})$$

f.d. vector spaces

Conclusion

- Question : how to define algorithms constructing morphisms (or objects, ...) in triangulated, DG, A_∞, ... categories and study their complexity ?

- Today I discussed some initial steps, including Turing machines for constructing new morphisms from old ones, in any closed symmetric monoidal category with cofree coalgebras, and in particular in $\underline{\mathrm{Vect}}_k$.

- The theory of coalgebras naturally leads one to study derivatives of algorithms (inc. TMs) and nondeterminism.

- What are cofree coalgebras in RMod, DG-RMod ?

References

[G] J.-Y. Girard, "Light linear logic", Information and Computation 14, 1995.

[CM] J. Clift and D. Murfet, "Cofree coalgebras and differential linear logic", arXiv: 1701.01285.

[M] D. Murfet, "Logic and linear algebra : an introduction", arXiv: 1407.2650.

[C] J. Clift, master's thesis at the University of Melbourne, 2017.