An abstract approach to network models of uncertainty

Jason Morton

Penn State

October 3, 2015
AMS Sectional Meeting

Includes joint work with David Spivak (operadic normal forms)

Motivation (network modelers)

Goal: Abstractions and engine to quickly build performant, modular network models of uncertainty. It should be able to:

- Ingest qualitative domain knowlege expressed with graphs/flow charts/diagrams from non-programmers
- Attach multiple competing quantitative explanations (e.g. probabilistic, differential equation, discrete dynamical) and test them
- Port algorithms, using best solution in each component (e.g. tree decomposition)
- Scale to useful size
- Pay only a small abstraction cost in final assembly program.

Motivation

Goal: build a practical computer algebra system for computational (monoidal) category theory. It should be able to:

1. manipulate abstract categorical quantities such as morphism terms in a REPL.
2. compile/lower code expressed categorically to an efficient implementation in a particular category (e.g. numerical linear algebra, (probabilistic) databases, quantum simulation, belief networks).
3. scale to be useful for practical computational problems in modeling uncertainty in data analysis, statistics, physics, computer science.

Want something like REPLs for linear or **commutative algebra**

In a computer algebra systems such as Macaulay2, Singular, Maple, Mathematica, we:

- tell the computer the context, e.g. `polynomial ring` $R = \mathbb{Q}[x, y]$, and
- type in an expression such as $x^2 + 3xy + (x + y)^2$.
- The system performs some simplification according to the axioms of a free polynomial ring (or computes a normal form in a quotient ring $R = \mathbb{Q}[x, y]/I$ using a Gröbner basis), and
- displays something like $2x^2 + 5xy + y^2$, `element of` R.
- Even something this trivial requires some thought for computational category theory!

Want something like MATLAB, NumPy, R

But that:

- allows a higher level of abstraction in describing algorithms,
- can handle more types of "data" than matrices of floats, probability distributions, or indeterminates and
- treats morphism expressions as first class to enable rewriting, syntax tricks
  - an algebraic version of manipulating the graph in a graphical model

For a class of modelling problems, computational category theory could be a tool like numerical linear algebra or convex programming.

- Now: reducing an applied math problem to numerical linear algebra means you can solve it using BLAS, LAPACK primitives, matrix decompositions, etc.
- Future: reduce your uncertainty/information-processing applied math problem to (computational) category theory, solve it using generic engines and libraries with matching abstractions.

- Morphism Term: a human-readable **qualitative model**, captured by a labeled generalized graph; fixes the relationships, *suggests* qualitative rules and syntax of the model
- Doctrine: formal **categorical syntax** constraining the quantitative models of uncertainty that can be attached, rewrite rules, available constructions
- Value: a machine-processed **quantitative model** in which the graph is interpreted and the data summarized, e.g. probabilistically as a Bayesian network, in Hilbert space for a quantum circuit, or with rate constants in a chemical reaction network
- Representation: the **interpretation** assigning quantitative meaning to the qualitative description (generalizing the mathematical idea of a representation of a quiver or algebra)
- **Algorithms**, categorically expressed, for processing and analyzing data. Make quantitative predictions, choose the model which best explains a given system (often a variant of belief propagation).

Implementation

- Can be implemented in any programming language with suitable features.
  - Needed are typeclasses/traits/interfaces and, for performance, a means for zero or low cost abstraction
  - so prefer JIT and AOT languages with modern type systems
  - building in Julia, exploring Scala, Rust, maybe Python (what would you use?)
- Typeclasses/Traits represent *doctrines*: monoidal category, compact closed category, well-supported compact closed category, ... and describe the *common interface* available to manipulate terms in any particular category
- Types represent particular categories (e.g. sets and relations)
- Values are objects and morphisms in the particular category

Implementation

Everything the computer does is represented as one of five modular components:

- a doctrine typeclass (e.g. "compact closed category")
- an instance or implementation of a doctrine as a pair of types (e.g. matrices or relations as CCC):
  - a morphism term or word (e.g. $f \otimes (g \circ \delta_A \circ h)$) in a free language, or
  - a concrete value in an implementation (e.g. $\begin{pmatrix} 1 & 3 \\ 4 & 5 \end{pmatrix}$)
- a representation (an X-functor) between implementations (usu. free $\to$ concrete, e.g. a binding $f = \begin{pmatrix} 1 & 3 \\ 4 & 5 \end{pmatrix}$ for each symbol)
- algorithms are expressed in terms of the defining methods of the doctrine (e.g. $\otimes$, δ, μ, bind, return) or operadically

Want something like REPLs for commutative algebra

In a computer algebra systems such as Macaulay2, Singular, Maple, Mathematica, we:

- tell the computer the context, e.g. `polynomial ring` $R = \mathbb{Q}[x, y]$, and
- type in an expression such as $x^2 + 3xy + (x + y)^2$.
- The system performs some simplification according to the axioms of a free polynomial ring (or more generally, a Gröbner basis computation in a quotient ring $R = \mathbb{Q}[x, y]/I$), and
- displays something like $2x^2 + 5xy + y^2$, `element of` R.

REPL/Computer algebra system for computational category theory

In a computational cagtegory theory REPL, we

- tell it the context, e.g. the tensor signature $f : A \to A$, $g : A \to B$, with doctrine "compact closed category," and
- type in an expression such as $\mathrm{ev}_A \circ (\mathrm{id}_{A^*} \otimes f) \circ \mathrm{coev}_A$.
- The system performs some simplification according to the axioms of a free compact closed category, and
- diplays $\mathrm{tr}(f)$.
- Even easier: write $\mathrm{ev}_A \circ (\mathrm{id}_{A^*} \otimes f) \circ \mathrm{coev}_A == tr(f)$ and have the system return "True"
- This easier problem is complete for the complexity class Graph Isomorphism (GI). There is a partial solution that doesn't require solving graph isomorphism: pick a good representation (functor).

Monoidal languages

- Given object variables $O = \{A_1, \ldots, A_n\}$, get monoid $O^{\otimes}$ of words such as $(A_5 \otimes A_3) \otimes A_1$.
- A tensor signature $\mathscr{T}$ comprises finite sets $\mathrm{Ob}(\mathscr{T})$ of object variables, and $\mathrm{Mor}(\mathscr{T})$ of morphism variables, and functions $\mathrm{dom}, \mathrm{cod} : \mathrm{Mor}(\mathscr{T}) \to \mathrm{Ob}(\mathscr{T})^{\otimes}$.
- $\mathscr{T}$ defines a monoidal category $\mathcal{M}_X(\mathscr{T})$,
  - augmenting $\mathrm{Ob}(\mathscr{T})$ with a monoidal unit $I_{\mathscr{T}}$ and
  - $\mathrm{Mor}(\mathscr{T})$ with a finite set of parameterized structure morphisms $\mathrm{PSM}(\mathscr{T}, X)$ depending on the doctrine X.
  - Here $X =$ "compact closed category", PSM are e.g. ev_A for any A, etc.
- The language $\mathscr{T}^{\otimes,\circ}_{CCC}$ is all valid morphism words that can be formed from $\mathrm{Mor}(\mathscr{T}) \cup \mathrm{PSM}(\mathscr{T}, CCC)$, so generates the free compact closed category over $\mathscr{T}$.
- Q: When do two words define the same morphism? NF?

Constructively

Constructively, $\mathscr{T}_{CCC}^{\otimes,\circ}$ is as follows.

- Each $f \in \mathrm{Mor}(\mathscr{T}) \cup \mathrm{PSM}(\mathscr{T}, CCC)$ is a word.
- Given words u, u', $u \otimes u'$ is a word with domain $\mathrm{dom}(u) \otimes \mathrm{dom}(u')$ and codomain $\mathrm{cod}(u) \otimes \mathrm{cod}(u')$.
- Given words w, w' with $\mathrm{dom}(w') = \mathrm{cod}(w)$, $w \circ w'$ is a word.

Mod the relations for a compact closed category, and imposing strictness, this gives a presentation of the free strict compact closed category over the generating set of object variables and morphisms.

- Now let's discuss how to represent morphism terms, or more precisely morphism expressions, in the computer.

S-expression

The value of the expression

```
(foo  bar  baz)
```

is the result of applying function `foo` to arguments `bar` and `baz` (which may themselves be expressions).

Morphism expressions

- A morphism term such as $\mathrm{ev}_A \circ (\mathrm{id}_{A^*} \otimes f) \circ \mathrm{coev}_A$ in a monoidal category can be represented as an expression tree (AST):

$$(\circ\ (\mathtt{ev}\ \mathtt{A})\ (\circ\ (\otimes\ (\mathtt{id}\ \mathtt{A})\ \mathtt{f})\ (\mathtt{coev}\ \mathtt{A})))$$

  here `ev`, `coev`, `id`, are unary functions, $\otimes$, and $\circ$ are 2-ary and `(⊗ g f )` means "apply function $\otimes$ to arguments g and f."
- Note: more than one expression tree can represent the same term (e.g. associate from left or right), and
- more than one morphism term can represent the same morphism in the free compact closed category (e.g. $(f_2 \otimes g_2) \circ (f_1 \otimes g_1)$ vs. $(f_2 \circ f_1) \otimes (g_2 \circ g_1)$ equal mod "deeper" relations).
- Can have other relations as well.

Complexity

- The easy part, Solving word problems such as $\mathrm{ev}_A \circ (\mathrm{id}_{A^*} \otimes f) \circ \mathrm{coev}_A \stackrel{?}{=} tr(A)$ is GI-complete.
- For some reasonable choices of normal form, finding the normal form (fixing tensor signature and doctrine = compact closed category):
  - given $\mathrm{ev}_A \circ (\mathrm{id}_{A^*} \otimes f) \circ \mathrm{coev}_A$, output $\mathrm{tr}(f)$

  is NP-hard (allows optimal contractions).
- This is not such bad news. The normal form for polynomials in a quotient ring, using Gröbner bases and Buchburger's algorithm, is worst case doubly exponential. Yet they are still extremely useful and practical for many computations.
- If we are willing to accept an imperfect but pretty good normal form, or can control the term complexity in various ways, we can get a good, fast normal form giving near-optimal contractions.

Computational category theory problems

- *term*: equivalence class of monoidal words over a finite tensor scheme, usually with certain additional properties X.
- *representation:* an X-monoidal functor "assigning values"

Questions of a term represented in a particular quantitative category:

1. compute a (possibly partial) contraction,
2. solve the word problem (are two terms equivalent, i.e. do they have the same representation) or compute a normal form for a term,
3. solve the implementability problem (construct a word equivalent to a target using a library of allowed morphisms), and
4. choose morphisms in a term to best approximate a more general term (possibly allowing the approximating term itself to vary).

Computational category theory is hard

- *term*: equivalence class of monoidal words over a finite tensor scheme, usually with certain additional properties X.
- *representation:* an X-monoidal functor "assigning values"

Questions of a term represented in a particular category:

1. compute a (possibly partial) contraction, (#P-hard)
2. solve the word problem (are two terms equivalent, i.e. do they have the same representation) or compute a normal form for a term, (undecidable)
3. solve the implementability problem (construct a word equivalent to a target using a library of allowed morphisms) (undecidable)
4. choose morphisms in a term to best approximate a more general term (possibly allowing the approximating term itself to vary). (NP-hard)

Recap

- Every doctrine has a free language that can be used to express morphism terms.
- This is how the machine represents domain-expert diagrams (e.g. gene interactions, high-school science test).
- A morphism term can be rewritten and simplified efficiently indpendently of interpretation/value.
- The normal form problem for morphism term subsumes query planning, finding an efficient way to contract a network, etc.
- Part of this problem can be solved "internally" by applying a suitable functor to a value category that removes distinctions between equal morphism terms.
- Sometimes it makes sense to think in terms of operad algebras rather than monoidal functors to exploit this.

Core algorithms

Some necessary component algorithms:

- - Determining equality for morphism terms [M- Spivak 2015, An Operadic approach to normal forms in compact closed categories],
  - Related aspects of normal forms and rewriting (such as finding tree decompositions), and
  - Categorically-formulated belief propagation, a common generalization of algorithms used for many types of models [M-2104, Belief propagation in monoidal categories].
- Status of *Cateno*: pre-alpha software, looking for "customers" and collaborators
- Now, a demo.