An approach to computational category theory

Jason Morton

Penn State

September 29, 2015
NIST Computational Category Theory Workshop

Includes joint work with David Spivak (operadic normal forms)

Outline

I'll discuss

- Overall design concept for a computer algebra and modelling system based on computational category theory
- Some necessary component algorithms:
  - Determining equality for morphism terms,
  - 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.
- Status of *Cateno*: pre-alpha software, looking for "customers" and collaborators

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.

Motivation (modelers)

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

- Ingest qualitative domain knowlege expressed with 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.

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 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.
- 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 or probability distributions, and
- treats morphism expressions as first class to enable rewriting, syntax tricks

Absurd Claim: Computational category theory is the numerical linear algebra of the 21st century.

- 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.

Design Principles

- Today: some **design principles** toward such a system.
- 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 JIT and AOT languages with modern type systems are preferred
  - 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

Design Principles

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

- a doctrine (e.g. "compact closed category")
- an instance or implementation of a doctrine (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

From the modeling perspective:

- 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).

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 interpretation 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.

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-complete (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.

Obvious approach: rewriting

- View a term as a grid of partial terms
- Look for tiles of any shape in the grid and rewrite them to something simpler
- Problem: dead ends, have to choose how to associate
- No known confluent terminating rewriting system
- Will need to solve parts of this eventually

S-expression

The value of the expression

(`foo` `bar` `baz`)

is the result of applying function `f` 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):

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

  here `ev`, `coev`, `id`, are unary functions, $\otimes$, and $\circ$ are 2-ary and $\texttt{(}\otimes\ \texttt{g f)}$ 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. $(f_2 \circ f_1) \otimes (g_2 \circ g_1)$).

Philosophy

- Fix a doctrine X.
- A morphism term (parenthesis and all) in an X-category is a program for obtaining a value.
- Don't rewrite!
- Instead, find good bindings for the symbols in another X-category
- such that when the program is executed, a normal form is obtained as that value.
- (or something from which a normal form can be extracted)

Operad approach

- Now we have not two $(\otimes, \circ)$ but one notion of composition: function composition.
- There are different ops or functions, taking different numbers of arguments, and we impose some associativity constraints.
- This matches the setting of operads.
- We represent each function in the expression (such as $\otimes$), and each primitive (e.g. f, ev_A) as an op in an operad. For the CCC doctrine, the relevant operad is 1-cobordisms.
- 1-D composition in that operad is associative. So different terms, and different trees with the same morphism give the same answer. (up to GI issue).

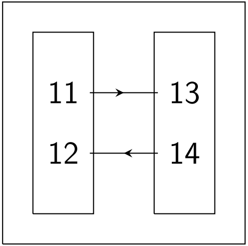
k-ary ops

The ops include

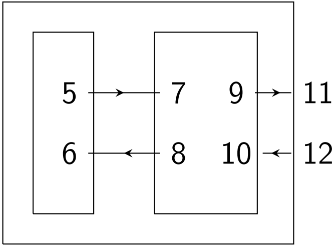
- 2-ary ops: $\otimes, \circ$
- 1-ary ops: (partial) traces, (partial) transposes, and
- 0-ary ops: ev_A, coev_A, id_A, $\sigma_{A,B}$.

Note that 0-ary ops and 2-ary ops again form a monoidal category equivalent to the original. Each op has a representation in 1-Cob.

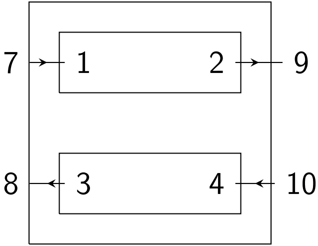
This depends on the type information, e.g. if $f : A \to B$ and $g : A \otimes A \to I$, the $\otimes$ in $f \otimes f$ and $g \otimes g$ technically represent two different ops (although they are implemented by the same function).



ϕ_3



ϕ_2



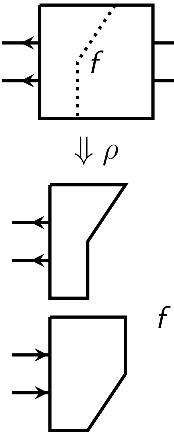
ϕ_1

The ops ϕ_3 and ϕ_2 correspond to compositions, while ϕ_1 corresponds to a tensor product.

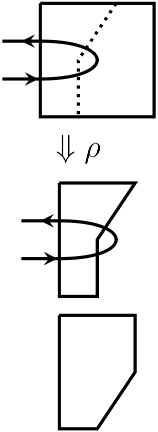
Representing in / Code lowering to 1-Cob

- Even when we have many objects and their duals, we can typecheck this at the top level (when the morphism term is formed).
- Represent object variables as wires, morphism variables as labeled identity ops, and structure morphisms ev,coev, etc. as special 0-ary ops.
- Lowered morphism term is just a 1-Cob
- Representing in 1-Cob, then evaluating, gives a (typed) morphism of 1-Cob.
- This typed morphism gets rid of all relations for CCC, except those requiring the solution of a graph isomorphism problem.

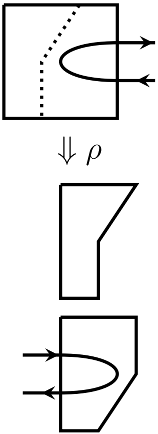
0-ary ops from morphisms: morphism variable f



0-ary ops from morphisms: coev : $I \to A \otimes A^*$



0-ary ops from morphisms: ev : $A^* \otimes A$



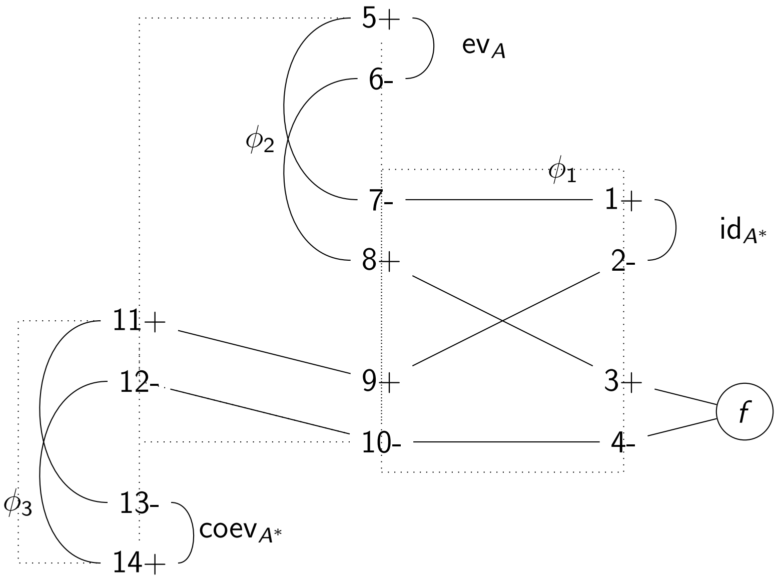
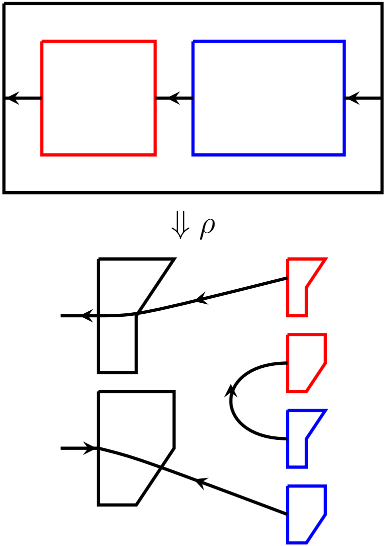


Figure : Operad tree and abstract syntax tree for $(\circ(\circ, \mathrm{ev}_A, (\otimes, \mathrm{id}_{A^*}, f)), \mathrm{coev}_{A^*})$ with cobordisms.

2-ary ops from composition: $\circ$



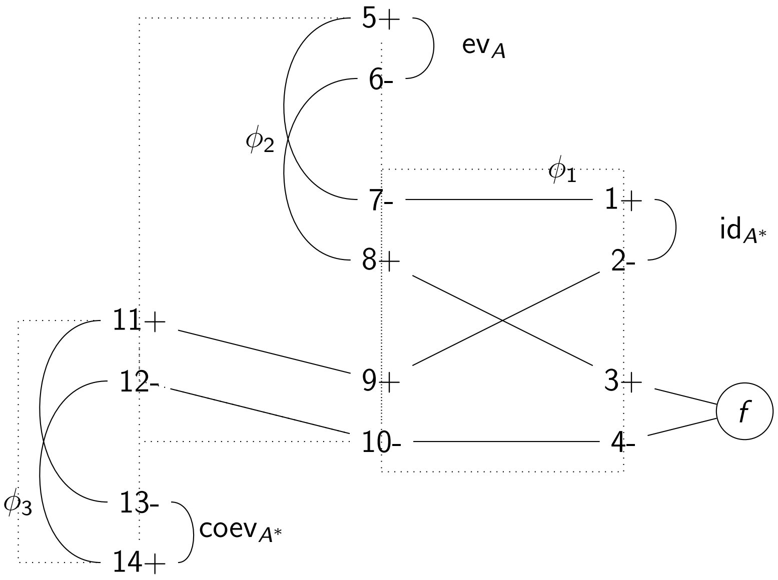
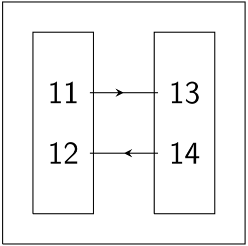
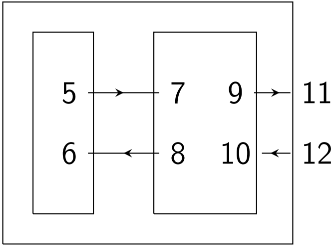


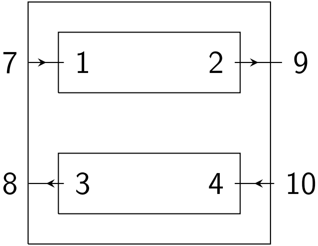
Figure : Operad tree and abstract syntax tree for $(\circ(\circ, \mathrm{ev}_A, (\otimes, \mathrm{id}_{A^*}, f)), \mathrm{coev}_{A^*})$ with cobordisms.



ϕ_3



ϕ_2



ϕ_1

The ops ϕ_3 and ϕ_2 correspond to compositions, while ϕ_1 corresponds to a tensor product.

Graph isomorphism issue

A hierarchy

- A morphism expression is what is stored in the machine
  - A strict morphism word is an equivalence class of expressions
- Evaluating a morphism expression in terms of 1-Cob as we've described yields a labelled 1-Cob.
  - If there is only one morphism variable, labelled 1-Cobs are equal iff the morphism terms are equal
  - If there are > 1 morphism variables, a labelled 1-Cob may have nontrivial automorphisms under permutation of the morphism variables
  - A labelled 1-Cob is an equivalence class of words (and so of morphism expressions)
- A morphism of the category is a (GI)-equivalence class of labelled 1-Cobs, up to these automorphisms.

Obtaining a normal form from a labeled 1-Cob

- Any stable procedure for obtaining a morphism expression from a labeled 1-Cob yields a normal form.
- Probably the most interesting are efficient ones wrt some measure of the cost of each 2-ary op.
- This is where the NP-hardness appears, in finding an optimal tree decomposition
- There are many deeply developed algorithms for this problem, and heuristics work quite well

Algorithms authored axiomatically

- When modeling, algorithms should be expressed in terms of categorical primitives.
- The right primitive for many algorithms is the well-supported compact closed category [RSW05].
- Belief propagation (and its many variants) can be expressed this way [Morton 2014].

(Limited) practical algorithms, even though the problems are hard in general

- Many practical questions are instances of one of these problems
  - quantum programming and logic
  - probabilistic graphical models,
  - tensor network state approach to quantum condensed matter,
  - computational complexity theory: circuits, CSP, #CSP
  - even databases
- So many tractable special cases, approximate algorithms, and heuristics exist
- Let's **turn these into categorical algorithms**, and formalize analogies among procedures.

Example: belief propagation

- A message-passing algorithm (Pearl 1982), for contraction, marginalization, and optimization problems
- Many extensions, analogs (survey propagation, turbo coding)
- These should be the same abstract categorical algorithm, varying the category (e.g. prob. graphical models vs. sets and relations).

To make this precise, first recall the set-up

Monoidal languages

- Given object variables $O = \{A_1, \dots, A_n\}$, 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?

I-valued points: the messages

- Important word problem for Belief Propagation: equality of morphisms of type $\mathrm{Mor}(I, A)$ for objects A (I-valued points).
- Why?
  - Want to generalize algorithms (e.g. belief propagation in the category of vector spaces and linear transformations)
  - Can't assume objects A are sets with points (such as probability distributions in the classical belief propagation algorithm).
- But, messages are still morphisms of type $\mathrm{Mor}(I, A)$ for each object A; equate these for belief propagation equations
  - Deciding if two vectors are equal up to numerical tolerance becomes deciding a **word problem** in $\mathrm{Mor}(I, A)$.
  - These messages must also be stored somehow.

Word problems in monoidal languages

- Coherent graphical languages for some types of monoidal categories means those word problems can be reduced to e.g. graph isomorphism, and produces normal forms by word$\mapsto$graph$\mapsto$word.
- Hence the word problem for the free closed category and free compact closed category over a finite tensor scheme are in LOGSPACE and P [Luk82] respectively.
- Adding adjectives (X-monoidal categories) and relations, or fixing values by applying a functor F, so that the category is no longer free may make it easier or harder.

Proposition

The word problem and implementability problem in a monoidal category over a finite tensor scheme are undecidable.

I-valued points: the messages

- For an efficient algorithm, need representation and word problem for I-valued points to be efficient.
- Classical belief propagation: have a monoid homomorphism, $\text{size}\colon \text{Ob}(\mathscr{T})^{\otimes} \to \mathbb{N}$, from the free monoid generated by the objects of our tensor scheme to the natural numbers.
- Monoidal product $\mapsto$ multiplication of vector space dimensions
- Then words in $\text{Mor}(I, A)$ can be stored and compared in $O(\text{size}(A))$.

Now look at type of category BP will work in. Need something like variables: objects in a well-supported compact closed category accomplish this.

Sum-product and belief propagation for contraction

- The sum product algorithm [KFL01]: if a term is a tree, can perform contraction according to the tree.
- If not, use a tree decomposition [Hal76] to force it to be a tree, then run sum-product.
  - Tree decompositions can be computed at the symbolic level with a cost function (e.g. dimension of each vector space); best is NP-hard but many good approximation strategies exist.
  - The result is the *junction tree algorithm* [LS88], also extended to the quantum case [MS08].
- Can improve on the abstract sum-product algorithm by using an optimized message-passing version, which among other benefits permits parallelization.

this is belief propagation

Belief propagation in factor graphs

- The algorithm operates on a *factor graph*, a bipartite graph with
  - one part discrete random variables $v \in V$ and
  - one part factors $u \in U$.
- Each factor (potential) assigns a real number to each combination of states of the variables it is connected to.
- Multiplying factors and normalizing if needed gives a joint probability distribution.
- Belief propagation is a message passing algorithm.
  - Each message is a probability distribution over the states one variable v can take: a vector in the associated vector space V_v.
- Each factor f_U at node u is a tensor in $\bigotimes_{v \in \mathrm{nbhd}(u)} V_v$, defines valence$(u)$ reshaped linear maps

$$f_{u,v} : \bigotimes_{i \in \mathrm{nbhd}(u) \setminus v} V_i \to V_v,$$

one for each $v \in \mathrm{nbhd}(u)$.

Messages at variables.

- Compute the pointwise (Hadamard) product of the incoming messages, and output it as the outgoing message along e.
- In a probabilistic category, Hadamard product rescales so the out message is a probability distribution.
- If there are no incoming messages, output the uniform message.

Messages at factors.

- Compute the tensor product of the incoming messages,
- apply reshaped $f_{u,v} : \bigotimes_{i \in \mathrm{nbhd}(u) \setminus v} V_i \to V_v$, and output the result as the outgoing message along the edge to v.

Resulting algorithm.

- *BP equations* describe fixed points of the update rules.
- Initial messages can be uniform distributions.
- Tree factor graph: done in two "passes," leaves to root then root to leaves, updating messages only as they change.
- Belief propagation is exact on trees

Messages at spiders.

- Apply the reshaped spider to incoming messages, and output the result as the outgoing message.
- If there are no incoming messages, treat the spider as a Frobenius unit.

Messages at "factor" morphisms.

- Compute the monoidal product of the incoming messages,
- apply the reshaped f,
- output the result as the outgoing message.

Resulting algorithm.

- System of BP equations are equalities of I-valued points describing the fixed points of the update rules.
- Initial messages can be chosen to be units at the spiders.
- Nice behavior on trees preserved

A spider is just a special kind of morphism. To get the **general bipartite** version, replace the message procedure at spiders with another copy of the factor message procedure.

To solve a problem, just reduce it to computational category theory

- Goal: general tools that work for any category with suitable properties
  - specialize automatically by giving a monoidal category interface
- Rapidly expanding universe of applied problems given categorical representations
  - a problem-solving abstraction with the potential to be as useful as convex programming or numerical linear algebra.

- Bob Coecke and Ross Duncan, *Interacting quantum observables: categorical algebra and diagrammatics*, New Journal of Physics **13** (2011), no. 4, 043016.
- Rudolf Halin, *S-functions for graphs*, Journal of Geometry **8** (1976), no. 1-2, 171–186.
- Frank R Kschischang, Brendan J Frey, and H-A Loeliger, *Factor graphs and the sum-product algorithm*, Information Theory, IEEE Transactions on **47** (2001), no. 2, 498–519.
- Steffen L Lauritzen and David J Spiegelhalter, *Local computations with probabilities on graphical structures and their application to expert systems*, Journal of the Royal Statistical Society. Series B (Methodological) (1988), 157–224.
- Eugene M Luks, *Isomorphism of graphs of bounded valence can be tested in polynomial time*, Journal of Computer and System Sciences **25** (1982), no. 1, 42–65.

Igor L Markov and Yaoyun Shi, *Simulating quantum computation by contracting tensor networks*, SIAM Journal on Computing **38** (2008), no. 3, 963–981.

Robert Rosebrugh, Nicoletta Sabadini, and RFC Walters, *Generic commutative separable algebras and cospans of graphs*, Theory and applications of categories **15** (2005), no. 6, 164–177.