Unfailing Network Rewriting

setting up the generic framework

Lars Hellström
Mälardalen University
lars.hellstrom@mdu.se

SNAG 2025, BTH, Karlskrona

Abstract

The simplification of algebraic expressions is a key step in many mathematical arguments; the systematic study of this is called rewriting, and the theory of Gröbner bases arises as an application to commutative algebra. For higher algebraic structures, things are not so straightforward; it may be quite difficult to distinguish one expression as simpler than another. Normally one requires the order which decides what is simpler (in Gröbner basis theory the "term order") to be compatible with placing monomials in contexts, but if such contexts may permute free variables then this prohibits the order from distinguishing expressions which are equal up to such permutation. In for example the theory of Lie algebras, that would render all terms of both the anti-commutativity and the Jacobi axioms incomparable, and thus useless for rewriting.

Unfailing Completion, introduced for term rewriting by Hsiang–Rusinowitch and Bachmair–Dershowitz–Plaisted in the 1980's, gets around this by abandoning compatibility, allowing the monomials to be ordered arbitrarily. Rewrite rules go every way, but are conditional: one may only apply a rule when doing so makes the expression smaller. This means one may need several resolutions of an ambiguity (critical pair), because what works in one context need not work in another. Enumeration of a sufficient set of cases is an interesting problem, with some similarities to that of enumerating all Gröbner bases of an ideal.

In this talk I will show how to integrate unfailing completion for networks into my multi-sorted "generic framework" for diamond lemmas.

A Diamond Lemma is a result relating

implicit descriptions of an algebraic structure, in terms of generators and relations, and

explicit descriptions of an algebraic structure, using a basis or other normal form.

Setting up the machinery lets you extract an explicit description from the implicit one, and gives you a discrete set of cases to check, by concrete calculations, for whether the two descriptions match.

Bergman's Diamond Lemma

The *Diamond Lemma for Ring Theory* applies to quotients of a free associative algebra $\mathcal{R}\langle X\rangle$, where $\mathcal{R}$ is a commutative ring with unit. For simplicity, I will assume $\mathcal{R}$ is a field here.

To prepare subsequent generalisation, denote:

$\mathcal{Y}$ is the set of monomials in $\mathcal{R}\langle X\rangle$.

$\mathcal{M}$ is all of $\mathcal{R}\langle X\rangle$.

Any relation to impose upon $\mathcal{R}\langle X\rangle$ can be given be form

$$\mu_s \equiv a_s \qquad \text{where} \quad s = (\mu_s, a_s) \in \mathcal{Y} \times \mathcal{M}.$$

Reductions

A simple reduction is a linear map $\mathcal{M} \longrightarrow \mathcal{M}$ which replaces *one* monomial by something equivalent to that monomial.

These can be parametrised by rules $s = (\mu_s, a_s) \in \mathcal{Y} \times \mathcal{M}$ and context $\lambda, \nu \in \mathcal{Y}$, to be defined as

$$t_{\lambda s \nu}(\mu) = \begin{cases} \lambda a_s \nu & \text{if } \mu = \lambda \mu_s \nu, \\ \mu & \text{otherwise} \end{cases} \qquad \text{for all } \mu \in \mathcal{Y}$$

and then extended to all of $\mathcal{M}$ by linearity.

General reductions are finite compositions of simple reductions.

Rewrite systems

A rewrite system S is a set of rewrite rules $s = (\mu_s, a_s) \in \mathcal{Y} \times \mathcal{M}$.
It defines a two-sided ideal

$$\mathcal{I}(S) = \langle \mu_s - a_s \rangle_{s \in S} = \left\{ \sum_{i=1}^{n} r_i \lambda_i (\mu_{s_i} - a_{s_i}) \nu_i \;\middle|\; \begin{array}{l} n \geqslant 0, \{\lambda_i\}_{i=1}^n, \{\nu_i\}_{i=1}^n \subset \mathcal{Y} \\ \{r_i\}_{i=1}^n \subseteq \mathcal{R}, \{s_i\}_{i=1}^n \subseteq S \end{array} \right\}$$

and a set of simple reductions by S

$$T_1(S) = \{t_{\lambda s \nu}\}_{\lambda, \nu \in \mathcal{Y}, s \in S}.$$

$T(S)$ is the set of finite compositions of such simple reductions.

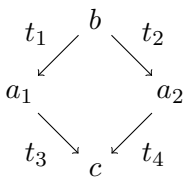
A general element $b \in \mathcal{M}$ is irreducible (a normal form) with respect to S if $t(b) = b$ for all $t \in T_1(S)$.
The set of all irreducible elements is denoted $\mathrm{Irr}(S)$.
This subspace is our candidate for *model* of the quotient $\mathcal{M}/\mathcal{I}(S)$.

Confluence and the Diamond Condition

Some $c \in \mathrm{Irr}(S)$ is called a normal form of $b \in \mathcal{M}$ if there is some $t \in T(S)$ such that $c = t(b)$.

It follows that $\mathcal{M} = \mathrm{Irr}(S) \oplus \mathcal{I}(S)$, and thus $\mathcal{M}/\mathcal{I}(S) \cong \mathrm{Irr}(S)$ (as vector spaces), *if* every $b \in \mathcal{M}$ has a unique normal from.
Such a rewrite system S is said to be confluent.

A necessary and sufficient condition for confluence (given lesser requirements) is the diamond condition (a.k.a. *local confluence*): for any $b, a_1, a_2 \in \mathcal{M}$ and $t_1, t_2 \in T_1(S)$ such that $a_1 = t_1(b)$ and $a_2 = t_2(b)$, there exist $t_3, t_4 \in T(S)$ such that $t_3(a_1) = t_4(a_2)$.

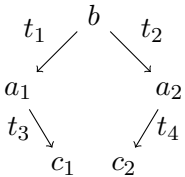


An aim is to find a small set of ambiguities (t_1, b, t_2) that are sufficient to check.
To begin with, because of linearity we need only consider $b \in \mathcal{Y}$.

Completion

When checking confluence, one may come across cases where the diamond fails to close.

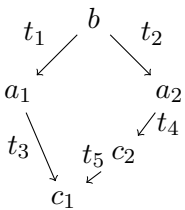
These are explicit counterexamples to confluence, and $c_1 - c_2 \in \mathrm{Irr}(S) \cap \mathcal{I}(S)$.



$c_1 \neq c_2$ both irreducible.

One can try to mend this by constructing a new rule s' out of $c_1 - c_2 \equiv 0$, and then seek to prove confluence of $S' := S \cup \{s'\}$.

Adding s' closes *this* diamond, but likely creates more ambiguities that need to be checked. The process need not terminate.



$t_5 \in T(S') \setminus T(S)$

Order

The Diamond Lemma is proved by *well-founded induction* on the monomials $\mathcal{Y}$.
This requires an order P on $\mathcal{Y}$, which is tailored to each application.

Reductions $t \in T(S)$ have to be compatible with the order, in the sense that

$$\text{if} \quad t(\mu) \neq \mu \quad \text{then} \quad t(\mu) \in \mathrm{Span}\big(\{\, \nu \in \mathcal{Y} \mid \nu < \mu \text{ in } P \,\}\big).$$

Provided $\nu < \mu$ in P implies $\kappa\nu\lambda < \kappa\mu\lambda$ in P for $\nu, \mu, \lambda, \kappa \in \mathcal{Y}$, this boils down to a condition that

> the left hand side μ_s of a rule must be strictly larger than every term in the right hand side a_s.

Hence, during completion, the order P determines which term is used for left hand side.
This will present some problems for us later.

Beyond associative algebras

Back in 2004, Jean-Louis Loday asked me whether this could be generalised to algebraic structures exhibiting features such as:

- Multiple multiplication-like operations.
- Multiplication-like operations of arity greater than 2.

My conclusion — after a day of thought — was that this was indeed possible, but would require rebuilding the supporting machinery.

It took until 2007 before my *Generic Framework for Diamond Lemmas* was ready for upload to arXiv.

- Lars Hellström. *A Generic Framework for Diamond Lemmas* (2007), 74 pages. arXiv:0712.1142v1 [math.RA].

Operads

An operad $\mathcal{Q}$ is a collection $\{\mathcal{Q}(n)\}_{n\in\mathbb{N}}$ of linear spaces.

The elements of component $\mathcal{Q}(n)$ are like multilinear expressions with n free variables.
In particular, the operad has to be closed under composition of elements, and that composition must behave like substitution for free variables.
(Warning: the formal details get messy.)

In an operad $\mathcal{Q}$, one might find:

$\mathcal{Q}(0)$	(nonassociative) algebra elements
$\mathcal{Q}(1)$	operators; *associative* algebra elements Example: the hom α of a hom-algebra
$\mathcal{Q}(2)$	binary operations; quadratic identities multiplication, (Lie) bracket; (anti-)commutativity
$\mathcal{Q}(3)$	ternary operations, cubic identities associativity, Jacobi identity, Leibniz identity
$\vdots$	$\vdots$

Multi-sorted framework

An operad is a multi-sorted algebraic structure — elements are *syntactically* different, since *how many* other elements an element would be composed with depends on to which component it belongs.

This calls for the rewriting framework to also be multi-sorted. There is a set of sorts I, and for each sort $i \in I$ there is:

- a space $\mathcal{M}(i)$ of *general elements* of sort i,
- a set $\mathcal{Y}(i) \subset \mathcal{M}(i)$ of *monomials* of sort i,
- a well-founded *order* $P(i)$ on $\mathcal{Y}(i)$,
- a set $T_1(S)(i)$ of *simple reductions* $\mathcal{M}(i) \longrightarrow \mathcal{M}(i)$,
- derived sets $\mathrm{Irr}(S)(i), \mathcal{I}(S)(i) \subseteq \mathcal{M}(i)$ and $T(S)(i)$.

The induction for the diamond lemma is done separately in each sort.

Unifying structures

The rewrite system S is *not* per sort, but common for all sorts.
A rule s may be formalised as a triple (i_s, μ_s, a_s) where $\mu_s \in \mathcal{Y}(i_s)$ and $a_s \in \mathcal{M}(i_s)$.

The multiplicative structure is what ties the components together.
Because there is not just one 'multiplication' operation, this is encoded into context maps v that maps monomials to (larger) monomials, just like

$$v(b) = \lambda b \nu \qquad \text{for all } b \in \mathcal{M}.$$

(This nicely abstracts away a lot of messy technicalities.)

Formally, the context maps are collected in a category $\mathcal{V}$, with the *sorts* as objects, and context maps as morphisms:

$$v \in \mathcal{V}(i,j) \quad \text{has} \quad v\colon \mathcal{M}(j) \longrightarrow \mathcal{M}(i) \quad \text{with } v\big(\mathcal{Y}(j)\big) \subseteq \mathcal{Y}(i) \cup \{0\}$$

For any $\mu \in \mathcal{Y}(j)$, $t \in T(S)(j)$, and $v \in \mathcal{V}(i,j)$ there must be some $t' \in T(S)(i)$ such that $t'\big(v(\mu)\big) = v\big(t(\mu)\big)$.

Reduction definition

The obvious definition of simple reductions in this framework is

$$t_{v,s}(\mu) = \begin{cases} v(a_s) & \text{if } \mu = v(\mu_s), \\ \mu & \text{otherwise} \end{cases} \qquad \text{for } \mu \in \mathcal{Y}(i), \text{ extend by linearity}$$

$$\text{where } s = (i_s, \mu_s, a_s) \in S \text{ and } v \in \mathcal{V}(i, i_s).$$

That $\mathcal{V}$ is closed under composition lets you transport rewrites carried out in some $\mathcal{M}(j)$ to $\mathcal{M}(i)$ across a context map $v \in \mathcal{V}(i, j)$.

If for $\lambda, \mu \in \mathcal{Y}(j)$ and $v \in \mathcal{V}(i, j)$

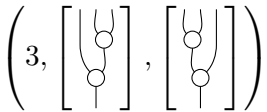
$$\lambda < \mu \text{ in } P(j) \qquad \Longrightarrow \qquad v(\lambda) < v(\mu) \text{ in } P(i)$$

then correct orientation of a rule implies all simple reductions generated by that rule are compatible with the ordering.

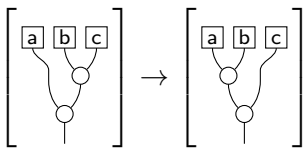
If, in the associative algebra setting, one instead restricts $\mathcal{V}$ to maps multiplying on only one side, the framework cranks out a theory for quotients by one-sided ideals.

Rewrite example

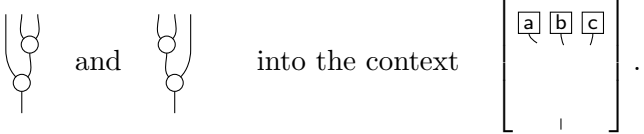
Associativity can expressed using the arity 3 rule



This provides for rewrite steps such as

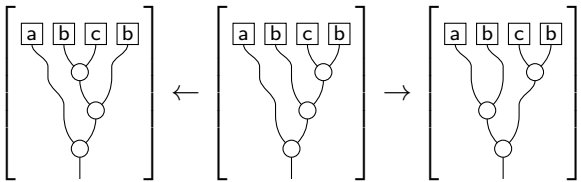


by inserting



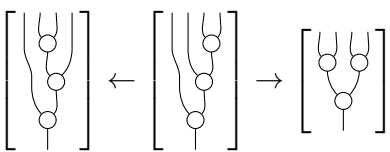
Ambiguity example

The same rule gives rise to the ambiguity

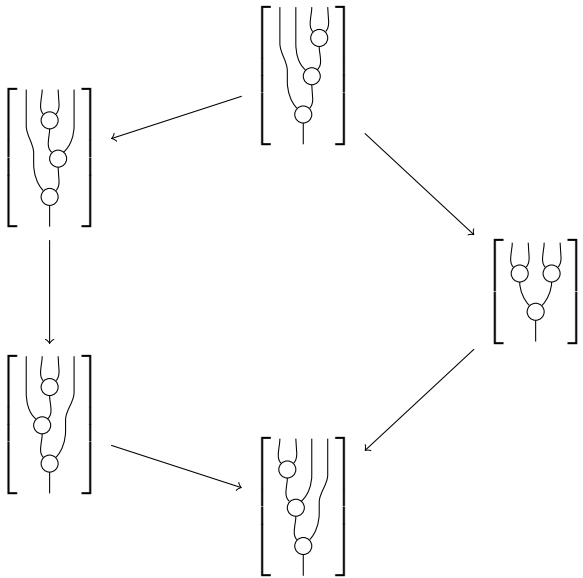


among with many others, in arity 0.

These are all mere shadows of the *critical* ambiguity of this rule with itself, which is found in arity 4:



Once you strip away the irrelevant context, you are left with those few cases you actually need to check.



The straightforward resolution of this particular ambiguity is sometimes celebrated as "the associativity pentagon".

When applying the generic framework for a new $\mathcal{Y}$ and $\mathcal{V}$, you need to analyse how ambiguities can be stripped down for these. Usually it boils down to enumerating the ways in which two monomials can overlap.

Total versus partial

In the Gröbner basis tradition, the ordering has to be total because they need to single out the *leading* term of arbitrary ideal basis elements when using them as rules.

In Bergman's tradition, orderings can be partial because the separation of left and right hand sides in rules is explicit.

One might expect total orderings to be more powerful, but it turns out there are things that a partial order can do which a total order cannot. Examples in:

- Lars Hellström and Sergei D. Silvestrov. Ergodipotent maps and commutativity of elements in noncommutative rings and algebras with twisted intertwining. *J. Algebra* **314** (1) (2007), 17–41. Doi: `10.1016/j.jalgebra.2007.03.031`
- Lars Hellström. Network rewriting II: bi- and Hopf algebras, pp. 194–208 in: *26th International Conference on Rewriting Techniques and Applications*, LIPIcs. Leibniz Int. Proc. Inform. **36** (2015). Doi: `10.4230/LIPIcs.RTA.2015.194`
- Lars Hellström. Valued custom skew fields with generalised PBW property from power series construction. Pp. 33–55 in: *Engineering mathematics. II*, Springer Proc. Math. Stat. Vol. **179** (2016). Doi: `10.1007/978-3-319-42105-6_3`

The permutation problem

Among the context modifications that you can do in an operad is to permute the inputs, written as a right action by a permutation: $b\sigma$ for $b \in \mathcal{Q}(n)$ and $\sigma \in \Sigma_n$.

Consider $v(b) = b\sigma$. If $v(\mu) \neq \mu$ then a total $P(n)$ would have

$$\mu < v(\mu) < v^{\circ 2}(\mu) < \cdots < v^{\circ k}(\mu) \text{ in } P(n)$$

or

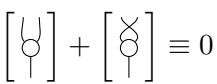
$$\mu > v(\mu) > v^{\circ 2}(\mu) > \cdots > v^{\circ k}(\mu) \text{ in } P(n).$$

However, for some $k > 0$ we get $\sigma^k = \mathrm{id}$, and thus $v^{\circ k}(\mu) = \mu$, contradicting transitivity.

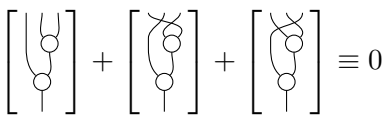
The conclusion is that if your algebraic structure allows contexts that are purely permutations, then monomials which are permutations of each other have to be unrelated, so the ordering cannot be total.

Lie axioms are troublesome

In the standard anticommutativity

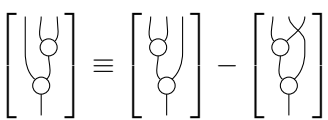


and Jacobi



axioms of a Lie algebra, all terms are permutations of each other.
That means you *can't make any rules* out of these!

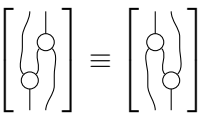
You can be clever and derive the Leibniz identity



where you can order the left hand side as larger than the right hand side terms, but that's an undesirable amount of preprocessing.
Also, Leibniz algebras are a wider class than Lie algebras.

It's more than just the permutations

When going beyond operads, to general networks as monomials, you encounter more comparisons than merely permutations that are troublesome.
In a Frobenius algebra, there is an identity



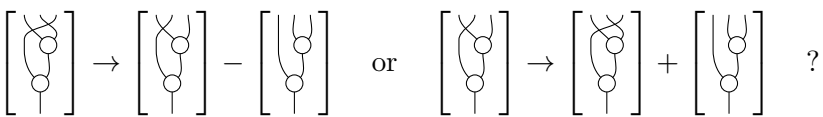
which is difficult to orient into a rule, because inputs and outputs here have different "dependencies" on each other: output-1 on input-2 in LHS, but not in RHS, whereas output-2 depends on input-1 in RHS, but not in LHS.

There might be a way around that if one discovers a new construction for ordering networks — I don't have a *proof* that it's not possible, as in the case of the permutations — but progress has not been promising.
It's *hard* to distinguish networks using an ordering, if that ordering is also to be preserved under placing things in contexts.

Ordinary completion may FAIL

For applying the diamond lemma, you *might* have succeeded with orienting all defining relations into rules, in which case a partial order suffices.

But if the aim is to run completion on a rewrite system, then you may encounter derived relations with multiple leading terms (multiple monomials are maximal), and then you cannot proceed.



There's no structural reason for picking one over the other!

Such a situation is described by saying the completion procedure fails.

Unfailing completion

Hsiang–Rusinowitch (1987) and Bachmair–Dershowitz–Plaisted (1989) suggested schemes of *unfailing completion* in term rewriting to overcome similar problems, mostly caused by relations that permute free variables.

The ideas are:

- Make sure you use a total order, if necessary by extending it arbitrarily. This *loses preservation of order* when changing the context.
- Make rewriting conditional: only apply a rewrite rule in those contexts where it makes things smaller.
  This restores induction, but may cause *resolutions* to work in some contexts but not others.
- Orient every identity every way.
  There is one rule which rewrites in one direction, and another rule which rewrites in the other direction. Which of those two rules you may apply depends on the context.

Unfailing in the generic framework

Since reductions are explicit entities in the generic framework, conditional rewriting is simply a matter of changing how the reductions are generated.

For a rule $s = (j, \mu, a)$ where $a = \sum_{k=1}^{n} r_k \kappa_k$, the modified construction of the simple reduction $t_{v,s}$ with $v \in \mathcal{V}(i, j)$ becomes

$$t_{v,s}(\lambda) = \begin{cases} v(a) & \text{if } \lambda = v(\mu) \text{ and } v(\kappa_k) < \lambda \text{ in } P(i) \text{ for } k = 1, \ldots, n, \\ \lambda & \text{otherwise} \end{cases}$$

for $\lambda \in \mathcal{Y}(i)$, extend by linearity.

The problem with this is: it need no longer be possible to transport resolutions across context maps, because some of the steps in a resolution might not exist in the target! (The rewrite step you wanted to take may violate the order condition.)

Order multiple ways

A freedom that exists in the generic framework is the system of sorts. Syntactic differences imposes a minimal separation of elements into different sorts, but there is no upper bound on how *fine-grained* the sort system can be.

In particular, you can have sorts $i_1 \neq i_2$ with the same expressions and the same monomials, but different orders:

$$\mathcal{M}(i_1) = \mathcal{M}(i_2), \qquad \mathcal{Y}(i_1) = \mathcal{Y}(i_2), \qquad P(i_1) \neq P(i_2):$$
$$\mu < \nu \text{ in } P(i_1) \quad \text{but} \quad \mu > \nu \text{ in } P(i_2) \qquad \text{for some } \mu, \nu \in \mathcal{Y}(i_1) = \mathcal{Y}(i_2).$$

This way, encountering essentially the same ambiguity in contexts that are so different that no single resolution works everywhere is no problem: they may be regarded as shadows of *different versions* of the same critical ambiguity, that are cast from different sorts.
As long as both versions have resolutions, there are resolutions of both shadows; in one context the ambiguity is resolved one way, in another context it is resolved another way.

Unfailing set-up

You start with a basic set of sorts I_0, and usual framework data for this:

- Per-sort general expressions $\{\mathcal{M}(i)\}_{i\in I_0}$, monomials $\{\mathcal{Y}(i)\}_{i\in I_0}$, and well-orders $\{P_0(i)\}_{i\in I_0}$.
- Category of context maps $\mathcal{V}_0$, with set of objects I_0.
- System of rules $s = (i, \mu, a)$ where $i \in I_0$, $\mu \in \mathcal{Y}(i)$, and $a \in \mathcal{M}(i)$.

Then introduce a "bundle" I over I_0, with projection $\pi\colon I \longrightarrow I_0$, and turn I into the set of sorts in a more fine grained system.

- Within each fibre of I, all the $\mathcal{M}$ and $\mathcal{Y}$ are the *same* as in the corresponding base sort.
- The order P varies between sorts in the same fibre. Some sort i in each fibre has $P(i) = P_0\big(\pi(i)\big)$.
- The morphisms of $\mathcal{V}$ (context maps) are likewise duplicated, but not maximally. Instead we only keep those lifts where the domains and codomains have matching ordering for this context: some $v \in \mathcal{V}_0\big(\pi(i), \pi(j)\big)$ is only in $\mathcal{V}(i, j)$ if

$$\lambda < \mu \text{ in } P(j) \quad \Longrightarrow \quad v(\lambda) < v(\mu) \text{ in } P(i) \qquad \text{for all } \lambda, \mu \in \mathcal{Y}(j).$$

Brute force fine structure

A brute force choice of bundle over I_0 is to use *all morphisms of* $\mathcal{V}_0$:

- $v \in \mathcal{V}_0(i,j)$ for $i,j \in I_0$ becomes a sort in the fibre over j (the domain).
- Starting from some family $\{P_0(i)\}_{i\in I_0}$ of "base orders", define

$$\lambda \leqslant \mu \text{ in } P(v) \quad \Longleftrightarrow \quad v(\lambda) \leqslant v(\mu) \text{ in } P_0(i) \qquad \text{for } v \in \mathcal{V}_0(i,j),\ i,j \in I_0.$$

Then the orders become compatible with the morphisms of $\mathcal{V}$ by definition.

In practice, $\mathcal{V}$ is a much too complicated set to be manageable as set of sorts.

What one wants is for each fibre $\pi^{-1}(j_0)$ of I to be given by some parametrisation of the orders $P(j)$ on that fibre, such that the family P is closed under applying contexts: for any $i \in I$, $j_0 \in I_0$, and $v \in \mathcal{V}_0\big(\pi(i), j_0\big)$ there exists $j \in \pi^{-1}(j_0)$ such that

$$\lambda \leqslant \mu \text{ in } P(j) \quad \Longleftrightarrow \quad v(\lambda) \leqslant v(\mu) \text{ in } P(i)$$

Hence ambiguities in sort i can be shadows of ambiguities in sort j.

Finding sufficient resolutions

With an *unfailing* set-up for the generic framework, enumerating ambiguities is the same as usual.

But an ambiguity might require several resolutions, so that in every context there's one which works.

These are always resolutions of the usual critical ambiguity, so they are all in the same fibre, but that fibre can be very big — for a base sort of arity n operad elements, the fibre might be parametrised by $n \times n$ matrices (over $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$).

You can't check every sort in the fibre explicitly, but you don't need to — the set of *possible resolutions* of an ambiguity is much smaller than the parent fibre.
(This is a bit like finding all the reduced Gröbner bases of an ideal in some $\mathbb{C}[x_1, \ldots, x_n]$ — there can be lots of these, but not infinitely many, because any reduced Gröbner basis has to be a selection of monic polynomials actually in the ideal.)

I have a recent paper on how to turn this enumeration problem into an optimisation problem. For operads, you solve linear programmes.

Thank you for listening!

On finding a covering set of resolutions:

- Lars Hellström. *Bilinear Feasibility and Unfailing Network Completion* (2025). In press.

Classical (fallible) network rewriting:

- Lars Hellström. *Network Rewriting I: The Foundation* (2012). `https://arxiv.org/abs/1204.2421` [math.RA].

- Lars Hellström. Network rewriting utility description. In: S. Silvestrov, A. Malyarenko (eds.) *Non-commutative and Non-associative Algebra and Analysis Structures*, pp. 429–476. Springer International Publishing, Cham (2023). Doi: 10.1007/978-3-031-32009-5_17