# Wrap ambiguities and how to enumerate them

Lars Hellström[1]

Division of Applied Mathematics and Physics, The School of Education, Culture and Communication, Mälardalen University, Box 883, 721 23 Västerås, Sweden
lars.hellstrom@mdh.se

**Abstract**

Network rewriting can be summarised as a generalisation of term rewriting to support that operations can have multiple out-parameters (coarity greater than 1) as well as the traditional multiple in-parameters (arity greater than 1). When fleshing out this idea, one is forced to make certain choices, which are discussed in this paper; network rewriting represent one way of making these choices: respect algebraic linearity, preserve acyclicity, and stay abstract (as opposed to imposing a geometric foundation).

In a 2014 IWC paper, it was reported that for network rewriting there emerges a third kind of ambiguity (critical pair) besides the classical overlap and inclusion ambiguities, namely wrap ambiguities where the way two redexes wrap around each other without overlapping can cause a rewrite of one to block the other. At that point it was not known how to enumerate these ambiguities, but here a generic method for this based on boolean matrices and SAT-solving is presented.

## 1 Discussion of models

Term rewriting operates on expressions as formalised in mathematical logic, where every combination of subexpressions to make a larger expression is by the use of an abstract function symbol taking zero or more arguments, and every expression is either a function application or a variable. However in modern algebra it is increasingly becoming necessary to deal with expressions that do not easily fit into this model; these theories comprise operations that vary not only in the number of in-parameters (arguments) they take, but also in the number of out-parameters they produce. To deal with these natively, one may relax the unspoken constraint that every expression has an underlying rooted (hence directed) tree structure, to allow the underlying structure to be that of a directed graph: operations are still vertices, there is an incoming edge for every in-parameter, and an outgoing edge for every out-parameter; an edge from one vertex to another means that an out-parameter of the first vertex is identified with an in-parameter of the second. Expressions are thus modelled as something like data-flow *networks*.

Whereas it may seem obvious that the optimal implementation of a certain computation may well be in terms of subroutines with multiple out-parameters—for example a single division operation that returns both a quotient and a remainder—it need not be immediately clear why said computation could not specified in terms of only single-result operations (such as separate quotient and remainder). A full explanation of this would have to explore the differences between cartesian and tensor products, but that is too long a digression to get into here; the interested reader may instead see [1]. The heart of the matter is however that many of the theories which make use of these operations with multiple results depend critically upon these being *entangled*, which means they have to be computed together.

In fact the interpretation of more general graphs as denoting expressions is not a trivial matter; the recursive interpretation for terms depends critically on them having a tree structure, which we just rejected. A data-driven evaluation—determine values on the outgoing edges from a vertex once values have been determined for all its incoming edges—is not out of the question, but would need to deal with entanglement explicitly, thus losing in generality. Instead the interpretation is mostly by decomposing the directed graphs into elementary pieces for which (function) values are given, and then a suitable algebraic structure (often a category) is used to recompose these elementary values into a value for the whole. The means of composition that this algebraic structure provides places restrictions on what a graph may look like if it is to be interpreted as an expression.

One axis of variation is what topological structure (if any) these graphs should be embedded into, which corresponds to the choice between symmetric, braided, or plain monoidal categories for governing the recomposition process. The symmetric case corresponds to abstract graphs, and should thus be the natural choice from a computer science or logic point of view (being less of an ontological commitment), but the plain and braided cases appear to be more popular in the category theory literature. A reason for that popularity is likely that topology has long been a prominent area for applications of category theory.

Another axis of variation is whether cycles should be allowed, and if so how an interpretation of those is concretely achieved; cycles in a data-flow network naively cause deadlock, when an operation vertex is waiting for input that (possibly in several steps) depend on an output of said vertex. Classically terms may be given something like a cycle in the underlying graph structure through some manner of fixed-point operator, but the rewriting of such is not entirely trivial. On the category side, the most direct way of supporting cycles is through the use of *traced* monoidal categories, where there is an operation trace/contraction/feedback that allows identifying an output with an input. A less direct way is by introducing 'cap' and 'cup' operations satisfying the zig-zag identities—this can be done as a matter of rewriting, but is often worked into the notation as 'raising/lowering indices' or 'bending edges' (allowing both endpoints to be heads or both tails)—and in particular the caps make heavy use of entanglement. However traces, caps, and cups can all be problematic when it comes to their interpretation in concrete applications; as a rule of thumb they are straightforward in finite-dimensional cases (the trace of matrix is trivial to evaluate) but may be impossible in infinite-dimensional cases (the trace of the identity operator becomes infinite). Hence it is for a general rewriting framework safest disallow cycles in expression, leaving it to users to add explicit caps and cups where appropriate.

A third axis concerns whether multiple edges may attach to the same "port" of a vertex, or equivalently, whether (internal) edges should have exactly two endpoints. Term graphs certainly suggest that using a single output as input in multiple places has its uses, and the share graphs of Hasegawa [2] aim to support this; likewise Ştefănescu [5] cover a number of variations in this regard, and also give examples of where such things may be appropriate. However in higher algebra there are strong reasons not to allow such things—changing the multiplicity of an intermediate result completely destroys multilinearity. In practice it is straightforward to introduce explicit operations for duplicating (coproduct) or destroying (counit) data, so a 1-to-1 restriction on the graphs is not a significant expressive loss, and several interesting algebraic theories even arise as a reformulation of classical theories to satisfy this linearity constraint; Hopf algebras arise from groups in that way. An notable consequence for rewriting is that unification disappears as a separate problem; it rather happens implicitly as part of overlaps involving rules for the coproduct and counit.

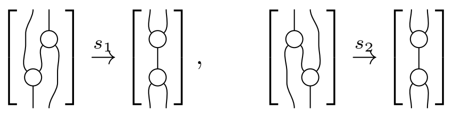
Finally it may be remarked that the 'monoid' in 'monoidal category' refers to the fact that the set of types supported constitute a monoid under concatenation/tensor product. In practice only free monoids seem to be used, which corresponds to having a set of atomic types given by the user, that do not interact except in operation vertices. A graph model for this should then make sure to label every edge with one of these atomic types—the type of data that may be carried along that edge—but for rewriting that is mostly rendundant since the type of an edge can be inferred from the vertices it is incident with, and overlapping existing type-consistent graphs will only generate new type-consistent graphs. Hence it is perfectly possible (an notationally easier) to set up the rewriting framework as being untyped, in which case symocats simplify to what MacLane called a PROP. It turns out *networks*—directed acyclic open graphs where vertices are with respect to in- and out-degree consistently decorated with symbols from a doubly ranked alphabet and each edge attach to a separate port of a vertex it is incident with—are modulo network isomorphism exactly the elements of the free PROP generated by that doubly ranked alphabet [3, Sec. 5].

## 2 Formal feedbacks

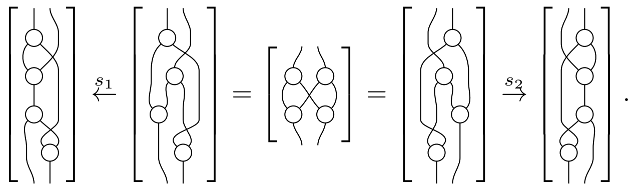
Even if cycles can be problematic for the interpretation of networks as expressions, they are quite useful when it comes to analysing the structure of networks, since they permit expressing any whole as an $A$ part beside a $B$ part, having some outputs of that $A \otimes B$ combinations connected back to select inputs of it, without getting into details of whether $A$ comes before $B$ dependency-wise, $B$ comes before $A$, or in fact it might be both. Interestingly enough, this is possible even in the free PROP, since it supports *formal feedbacks* [3, Sec. 9].

The idea is that one may to any network $G$ associate a boolean matrix $\mathrm{Trf}(G)$ called the *transferrence* of $G$: this has a 1 in position $(i, j)$ iff there is a directed path in $G$ from input $j$ to output $i$; this Trf may also be interpreted as a PROP homomorphism into the PROP of boolean matrices. If $G$ and $H$ are networks whose transferrences have block matrix decompositions $\mathrm{Trf}(G) = \left[\begin{smallmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{smallmatrix}\right]$ and $\mathrm{Trf}(H) = \left[\begin{smallmatrix} b_{22} & b_{23} \\ b_{32} & b_{33} \end{smallmatrix}\right]$ where $a_{22}$ is $q \times r$ and $b_{22}$ is $r \times q$, then the matrix $a_{22}b_{22}$ is nilpotent iff the *symmetric join* $G \bowtie^q_r H$ is acyclic, that one obtains by identifying the $q$ last outputs of $G$ with the $q$ first inputs of $H$ and likewise the $r$ first outputs of $H$ with the $r$ last inputs of $G$. This carries over to the free PROP, which canonically comes with a filtration $\mathcal{F}$ indexed by boolean matrices, such that $\mathcal{F}_a$ is the set of all elements $\mu$ of the free PROP that have a transferrence $\leq a$ in the standard matrix order. $\mathcal{F}$ being a PROP filtration, it follows from $\mu \in \mathcal{F}_a$ and $\nu \in \mathcal{F}_b$ that $\mu \circ \nu \in \mathcal{F}_{a \circ b}$ (if $a \circ b$ is defined) and $\mu \otimes \nu \in \mathcal{F}_{a \otimes b}$, but more importantly each $\bowtie^q_r$ may, provided $a$ and $b$ satisfy the above nilpotency condition, be regarded as an operation $\mathcal{F}_a \times \mathcal{F}_b \longrightarrow \mathcal{F}_c$ for $c = \left[\begin{smallmatrix} a_{11}+a_{12}b_{22}(a_{22}b_{22})^*a_{21} & a_{12}(b_{22}a_{22})^*b_{23} \\ b_{32}(a_{22}b_{22})^*a_{21} & b_{33}+b_{32}a_{22}(b_{22}a_{22})^*b_{23} \end{smallmatrix}\right]$, where $p^* = \sum_{k=0}^{\infty} p^k$ denotes the *Kleene star* of the boolean matrix $p$. Joining with a width $q$ identify is also known as the width $q$ *(formal) feedback* $\uparrow^q$. A symmetric join on all inputs and outputs of the right factor is for simplicity denoted $\rtimes$.

In [3, Sec. 10] this was used to construct a rewriting theory for networks, where a rewrite rule $\mu \to \mu'$ with $\mu, \mu' \in \mathcal{F}_b$ could be placed into a context defined by some $\nu \in \mathcal{F}_a$ and used to do $\nu \rtimes \mu \to \nu \rtimes \mu'$ whenever that symmetric join is defined; the machinery of formal feedbacks make it feasible to ensure that this always respects the ordering with which these rules are compatible. It was however in the detailed analysis of the resulting ambiguities (critical pairs) observed that one could not claim that resolving only overlap and inclusion ambiguities would suffice unless assuming that $b = \mathrm{Trf}(\mu_1)$ (rewrite rule is "sharp"), and in [3, Ex. 10.28] an example was given of a *wrap ambiguity* where two redexes would block each other despite being disjoint, by virtue of each having a input that depends on output from the other. The two rules



give rise to the critical pair

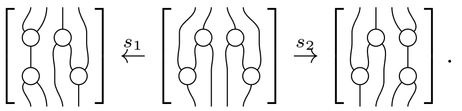


(1)

Characterising those situations where this happened were then left as an open problem. A compounding factor is that this mainly seemed to arise where the transferrence of a network would change without decreasing (rather increase or be outright incomparable), which is troublesome when one seeks to find a compatible ordering: the rules involved in this are often also not (easily) orientable. That is however a different story.

## 3 Networks with obligations

Eliding the interconnections between the two redexes, which would go from output 3 to input 1 and from output 2 to input 4, the ambiguity looks like



In either branch, applying the other rule to its redex is blocked because doing so would create a cycle due to the change introduced by the first rule, whereas in the initial configuration both rules can be applied. Since nilpotency of transferrence matrices allows us to test for acyclicity, we can formulate a condition for the transferrence $\left[\begin{smallmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{smallmatrix}\right]$ of a context $\nu$ that would cause this: the block $p_{22}$ connecting outputs of the ambiguity back to inputs thereof must have

$$\begin{pmatrix} 1&1&0&0\\0&1&0&0\\0&0&1&0\\0&0&1&1 \end{pmatrix} p_{22} \text{ nilpotent,} \qquad \text{but } \begin{pmatrix} 1&1&0&0\\1&1&0&0\\0&0&1&0\\0&0&1&1 \end{pmatrix} p_{22}, \begin{pmatrix} 1&1&0&0\\0&1&0&0\\0&0&1&1\\0&0&1&1 \end{pmatrix} p_{22} \text{ non-nilpotent.} \tag{2}$$

The only solution to this is that $p_{22} = \begin{pmatrix} 0&0&1&0\\0&0&0&0\\0&0&0&0\\0&1&0&0 \end{pmatrix}$, which indeed places us in the situation depicted in (1).

How could one automatically solve such problems, nilpotency being a rather nonlinear constraint? It turns out that they can quite conveniently be formulated as boolean satisfiability problems, with the elements of $p_{22}$ as individual boolean variables. Nilpotency as such may seem difficult to encode, but the nilpotency of a boolean $n \times n$ matrix $A$ is equivalent to the claim that $A^n = 0$, and repeated boolean multiplications are straightforward to encode if one introduces helper variables for the elements of the intermediate products; exponentiation by squaring helps to further reduce the number of multiplications that need to be encoded. Non-nilpotency of a boolean matrix $A$ is conversely equivalent to the claim that its Kleene plus $A^+ = AA^* = \sum_{k=1}^{n} A^k$ does not have a zero diagonal, which again is thus possible to encode in terms of repeated boolean multiplications.

In hindsight, a problem with the [3] rewriting theory is that it assigns a transferrence to each rule, when what it in fact needs is to know what restrictions may be imposed by the context in which the rule is to act. For the derived rule a completion would produce from (2) the least possible transferrence is $\left(\begin{smallmatrix}1&1&0&0\\1&1&0&0\\0&0&1&1\\0&0&1&1\end{smallmatrix}\right)$, but that is also too large to allow that both dependencies expressed by the $p_{22}$ matrix above—this derived rule would not be able to resolve the very ambiguity from which it was derived! That is clearly not satisfactory, but such is sometimes the lure of the algebra; the theory of the PROP filtration $\mathcal{F}$ simply looked too good for it to not be the right basis for the rewrite theory. It still has its uses, but it is not what should characterise the rewrite rules.

A better approach is instead to let each rewrite rule come with an *obligation* of supporting a certain amount of feedback imposed by the context in which it is to operate—essentially that $p_{22}$ matrix derived above. The basic sets $\mathcal{Y}(r)$ of objects being rewritten are indexed by boolean matrices $r$, and consist of all $\mu$ in the free PROP such that $\mathrm{Trf}(\mu)r$ is (defined, square, and) nilpotent; ordinary algebraic expressions have obligation $r = 0$, but higher obligations arise when resolving ambiguities. A rule $\mu \to \mu'$ supporting obligations $r$ can be applied to make the rewrite step $\nu \rtimes \mu \to \nu \rtimes \mu'$ while respecting obligations $q$ iff the transferrence $\mathrm{Trf}(\nu) = \left[\begin{smallmatrix}p_{11}&p_{12}\\p_{21}&p_{22}\end{smallmatrix}\right]$ also satisfies (i) that $qp_{11}$ is nilpotent and (ii) that $p_{21}q(p_{11}q)^*p_{12} + p_{22} \leqslant r$; in other words the context does not by ifself violate the target obligations $q$, and combining the context with those obligations does not create effective obligations exceeding those that this rule supports.

For enumerating wrap ambiguities, this leads to a slightly more complicated set of constraints that just the (2) combination of nilpotency and non-nilpotency, but it is all possible to handle with the same set encoding tricks, of which the theoretically foremost is that one need only consider matrix powers up to a known bound.

## References

[1] John C. Baez and Mike Stay. Physics, Topology, Logic and Computation: A Rosetta Stone. Pp. 95–174 in *New Structures for Physics* (ed. Bob Coecke), Lecture Notes in Physics vol. **813**, Springer, Berlin, 2011. arXiv:0903.0340v3.

[2] Masahito Hasegawa. *Models of sharing graphs.* CPHC/BCS Distinguished Dissertations. Springer-Verlag London, Ltd., London, 1999. A categorical semantics of let and letrec, Dissertation, University of Edinburgh, Edinburgh. doi:10.1007/978-1-4471-0865-8.

[3] Lars Hellström. Network Rewriting I: The Foundation, April 2012. arXiv:1204.2421 [math.RA]. arXiv:1204.2421.

[4] André Joyal and Ross Street. The Geometry of Tensor Calculus, I. *Adv. Math.* **88** (1991), 55–112.

[5] Gheorghe Ştefănescu. *Network Algebra.* Springer, 2000. ISBN 1-85233-195-X.