Critical Pairs in Network Rewriting

Lars Hellström

Division of Applied Mathematics, The School of Education, Culture and Communication, Mälardalen University, Box 883, 721 23 Västerås, Sweden; lars.hellstrom@residenset.net

Abstract

This extended abstract breifly introduces rewriting of networks (directed acyclic graphs with the extra structure needed to serve as expressions for PROducts and Permutations categories) and describes the critical pairs aspects of this theory. The author's interest in these comes from wanting to do equational reasoning in algebraic theories (such as Hopf algebras) that mix ordinary operations with co-operations; networks then serve as a formalism for expressions.

The main message is to point out two phenomena that arise in network rewriting. The first is that of non-convexity of rules, wherein the left hand side of a rule need not be syntactically similar to a symbol in any extension of the underlying signature. The second is one of critical pairs potentially arising where two redexes wrap around each other even when they do not intersect.

1 Introduction to networks

Words provide a natural model for expressions in an algebraic theory of all-unary operations; the corresponding abstract algebraic structure is the monoid, and the set of all words may be formalised as the free monoid. Generalising to operations of arbitrary arity, the natural model for expressions instead becomes that of terms. Adding the condition that the terms should be linear in the sense that each variable occurs exactly once, one arrives at a concept whose corresponding abstract algebraic structure is called an operad. Operads first became popular within topology, but have since become useful tools also in algebra in general, especially to study non-associative structures.

The generalisation from one to many is however not exhausted by operads or terms: if an n-ary operation would be implemented by a subroutine with 1 out-parameter and n in-parameters, then what kinds of expressions could be built from operations that syntactically are like subroutines with m out-parameters and n in-parameters? The corresponding abstract algebraic structure will be the PROP—or strict symmetric monoidal category (symocat) if one works with multiple atomic sorts—and the natural expression model will be that of networks [2].

A *network* is essentially like a term, expect that instead of having an underlying tree there is an underlying directed acyclic graph (DAG). Formally starting from a DAG, the extra data needed to turn it into a network are the following. (i) Each inner vertex is given a symbol from a doubly ranked alphabet. If the symbol $D(v)$ of vertex v has rank (m, n), then the in-degree of v must be n (the *arity*) and the out-degree of v must be m (the *coarity*). (ii) There is at each vertex a total ordering of the incoming edges, and a separate total ordering of the outgoing edges. (iii) There are two distinguished vertices 0 and 1 that represent the output and input respectively sides of the network; the arity of the network as a whole is the degree (all outgoing) of the input vertex 1, and the coarity of the network as a whole is the degree (all incoming) of the output vertex 0. In the special case that each symbol in the alphabet has coarity 1, the networks with coarity 1 are precisely the linear terms (networks of higher coarity would be to linear terms as forests are to trees). As expression models, networks are special cases of *share graphs* [1], but their built-in linearity—that each edge has exactly one tail and exactly

one head implies each intermediate result is generated once and used once—make them valid as expressions in a much wider range of contexts, such as quantum computing and multilinear algebra where a classical duplication of information would violate fundamental axioms.

Just like a monoid offers a natural setting for evaluating a word as an expression, the natural setting for evaluating a network is an algebraic structure called a PROP. A *PRO* [4, Ch. V] is a set of doubly-ranked elements together with two composition operations: the serial composition $\circ$ which corresponds to ordinary composition of functions, and the parallel composition $\otimes$ which corresponds to letting two functions act separately on disjoint parts of a composite argument; a trivial example of the latter is to make $(f \otimes g)(x, y) := \big(f(x), g(y)\big)$, whereas other basic examples make $\otimes$ the tensor product of two linear maps. As operations on networks, $\circ$ amounts to joining the outputs of the right operand to the inputs of the left operand, whereas $\otimes$ simply places the operands side-by-side, exposing each input and output of either operand as an input or output of the combined network. A *PROP* is a PRO equipped with actions of permutations on the elements, which for networks correspond to permutating outputs among themselves and/or inputs among themselves. The formal definition of how to evaluate a network is rather technical [2, Def. 5.2], although the fact that it can be done can be taken as an alternative definition of PROP [2, Th. 5.17].

The multiple atomic sorts counterpart of a PRO is a monoidal category, whereas in terms of networks it would correspond to adding a planarity constraint in the sense of 'no crossing edges'. PROs may seem more elementary if coming from the abstract algebra point of view, but from the formalised expression perspective they rather constitute a curious restriction, in that they call for items of data to be located to points within a geometric space.

2 Network rewriting

Naively, a rewriting step consists of replacing one subexpression equal to the left hand side of a rule with the right hand side of that rule. Since networks are graphs (with extra structure), network rewriting is visually intuitive: you cut some edges, remove the piece that thereby got separated and match it to the left hand side of a rewrite rule, replace the piece with a new one equal to the rewrite rule right hand side, and splice together the edges at the cuts. What turns out to be a nonobvious matter is however that of what kind of piece qualifies as a subexpression: different established formula formalisms lead to different answers.

The monoidal category perspective suggests that a rule $l \to r$ can be applied to those expressions that are obtained by padding l using the two compositions $\circ$ and $\otimes$, i.e., that any rewrite step done using $l \to r$ can be written as

$$B \circ C_1 \otimes l \otimes C_2 \circ D \to B \circ C_1 \otimes r \otimes C_2 \circ D$$

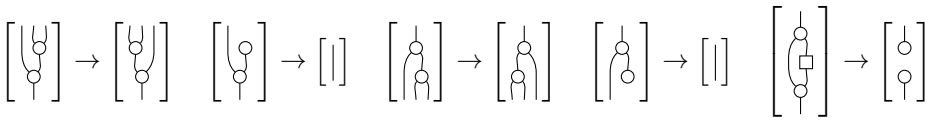
for some B, C_1, C_2, and D, where $\otimes$ is taken to have higher priority than $\circ$. (Having permutations allows combining C_1 and C_2, but that is beside the point here.) This is also the subexpression concept one gets from a straightforward double pushout graph rewriting formalism where l and r are both being produced as images (under separate morphisms) of a marker symbol x, and the context graph network has the form $B \circ C_1 \otimes x \otimes C_2 \circ D$. It is however not the most general subexpression concept.

An alternative formula formalism for writing expressions in PROPs and symocats is the Abstract Index Notation [7], which is an abstract reinterpretation of the Einstein summation convention for tensors. Here, an expression is written as a formal product of factors which each carry zero or more sub- and superscripts, e.g. $\mu^a_{bc} S^b_d \Delta^{dc}_e$. The Einstein summation convention

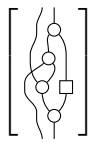
says that there is an implicit summation over any index letter appearing once as superscript and once as subscript in a product, whereas letters with one appearance in total are externally visible indices of the composed tensor, and two or more appearances of the same kind (super or sub) is forbidden. In terms of networks, the abstract index reinterpretation is that each factor is a vertex, the base letter of a factor is the operation symbol associated with that vertex, and the index letters name the edges that are incident with the vertex: superscripts are outgoing, subscripts incoming. Here, two appearances of an index letter means it is an internal edge, whereas just one means it is an external edge with the other end at the implicit output vertex 0 or input vertex 1 as appropriate; two or more appearances of the same kind are impossible for a directed edge. In abstract index notation, any subset of the factors would constitute a valid subexpression, so a rewrite step can be any $lA \to rA$, where l, r, and A are products of labelled factors such that the composite products lA and rA satisfy all syntactic constraints.

This is different from the previous subexpression concept in that it allows subexpressions to be *nonconvex*: a path may begin in a vertex/factor of l, pass through some vertex in the context A, and then return to the subexpression l, even if it must then be at a different vertex of l than that at which it started. This is not possible in a formalism that considers a subexpression to be something that is similar to a vertex, since any path leaving a vertex through one edge and then returning to it via another would constitute a cycle. Are nonconvex subexpressions useful for rewriting, though? The system for Hopf algebras considered in [3] demonstrate that they are very useful indeed.

The axioms for a Hopf algebra can be naturally stated as a network rewrite system, five of the rules in that being



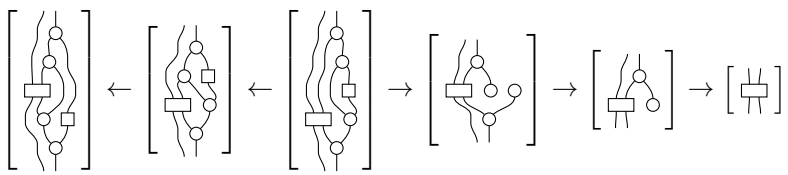
The brackets here serve as frames around a network when it appears as part of a formula, to clarify its graphical extent. For the interpretation of the various vertex types in the case of Hopf algebras, see [3]. Completing the rewrite system consisting of just the Hopf axioms does however produce several nonconvex rules. One of these derived rules (which follows directly from those five above) is:



where the left input may have a depenence on the left output;

this 'may have a dependence on' phrase is stating that the left hand side may be matched against a nonconvex subexpression during rewriting, and it says how that nonconvexity may be realised.

The reason this rule should be nonconvex is that there is no step in its derivation which contradicts a dependence of left input on left output; any rewrite step made where this rule is matched against a nonconvex subexpression can alternatively be carried out as a sequence of forward and backward steps using the five axiom rules above, at each step only replacing convex subexpressions. One example of this would be



(1)

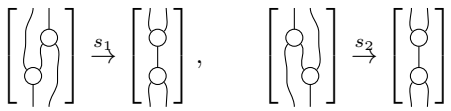
where the right input to left output edge of the $r = [\times]$ network is part of the top right edge of the $\left[\text{⫫}\right]$ network, whereas the left input to right output edge of the r network is part of the bottom right edge. It can therefore be argued that nonconvex subexpressions are natural from a rewriting perspective: since a derived rule is somewhat like a prepackaged sequence of rewrite steps, and since there in a two-dimensional setting is no reason for the union of a number of polygons (say) to be convex (even if each component is convex and the union is connected), there is no reason to expect that derived rules will all be convex even if the rules they are derived from happen to be. Indeed, some experience with completing network rewriting systems suggest that derived rules will typically grow nonconvex fairly soon after they have become complicated enough to exhibit such features.

Allowing rewrite rules to match against nonconvex subexpressions does however raise the problem of how to avoid creating cycles. (Many technicalities become much easier if one allows cycles, including that of defining evaluation of a network—cf. the 'normal form expressions' in [6, Sec. 1.4]—but far from all interpretations of networks support cycles. In a computational context, a cycle could correspond to sending information backwards in time, or at best to some kind of fixpoint operation. For multilinear interpretations, cycles tend to be problematic as soon as one considers infinite-dimensional spaces.) The framework of [2] uses a filtration (indexed by boolean matrices) of the PROP of networks to keep track of which dependencies of inputs on outputs of a network are consistent with it appearing as a subnetwork of another network. Each rule has an associated *transferrence type*, and rules may only apply in contexts consistent with that type. Likewise, each critical pair has an associated transferrence type, and if it gives rise to a derived rule, then that will also be the transferrence type of that derived rule. In the author's opinion, nonconvex rules are well understood and cared for by the framework of [2].

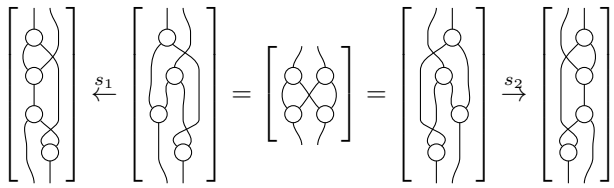
3 Critical pairs

The nice thing about the ability to handle nonconvex subexpressions in network rewriting is that the sites of critical ambiguities (i.e., critical pairs) need never include vertices that do not correspond to a vertex in the left hand side of at least one of the rules [2, proof of L. 10.13]; without this, (1) would give rise to a separate critical pair for every way of replacing the wide ⫫ vertex with something else. In [5], Mimram seeks to achieve the same end of eliminating irrelevant vertices by formally bending edges; unlike the nonconvex subexpression concept, this cannot cope with an irrelevant vertex completely surrounded by vertices acted upon by a rewrite rule of the critical pair, but on the other hand it preserves the plane embedding which is a concern in that paper. As mentioned above, the network rewriting formalism of [2] does not consider data items to have a location in a geometric space, so concerns about planarity are meaningless.

An issue that at present is not fully understood is however that of critical pairs of 'wrap' type, which are due to two redexes wrapping around each other in such a way that reducing one will block reducing the other, even though they do not overlap. The simplest example of this is probably that the two rules



give rise to the critical pair



(2)

where a naive attempt at applying the other rewrite rule at either side would violate acyclicity. The left hand side of one rule can still be found as a subexpression after the other rule has been applied, but it is no longer a redex since the subexpression does no longer satisfy the transferrence constraints associated with the rule. This is different from at least the elementary sense of rewriting residual (wherein a redex is moved by but still remains a redex after the application of a separate rule; the network rewriting framework handle ordinary residuals under the name of 'montage ambiguities' [2, Def. 10.14]).

It is notable that this example of a wrap ambiguity does not rely on having nonconvex rules or subexpressions, so including them in the framework did not cause this problem. Indeed, it is rather the impression of the author that wrap ambiguities is a problem that nonconvex rules do not quite manage to solve, even though that they are in the vicinity of doing so; if networks are viewed as merely having a particular arity and coarity, then wrap ambiguities cannot be ruled out, but if they are instead viewed as having a particular transferrence type then there is a practical condition ('sharpness' [2, Def. 10.3]) that will guarantee that a rewrite system is free of wrap ambiguities. Perhaps a further refinement of the network rewriting framework can help eliminate them altogether.

On the other hand, the nontrivial derivation in (2) suggests that there really is something here that an automated completion procedure would need to explore. It is an open problem in the theory of network rewriting to enumerate all critical pairs where wrap ambiguities remain a possiblity.

References

[1] Masahito Hasegawa. *Models of sharing graphs*. Diss. PhD thesis, University of Edinburgh, Department of Computer Science, 1997.

[2] Lars Hellström. *Network Rewriting I: The Foundation*, 2012. arXiv:1204.2421v1 [math.RA].

[3] Lars Hellström. *Network Rewriting II: Bi- and Hopf Algebras*. Manuscript, 2014, 15 pp. urn:nbn:se:mdh:diva-25102

[4] Saunders MacLane. Categorical Algebra. *Bull. Amer. Math. Soc.* **71** (1965), 40–106.

[5] Samuel Mimram. *Computing Critical Pairs in 2-Dimensional Rewriting Systems*. arXiv:1004.3135v1 [cs.FL].

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

[7] Wikipedia. *Abstract index notation — Wikipedia, The Free Encyclopedia*. `http://en.wikipedia.org/w/index.php?title=Abstract_index_notation` [Online; accessed 4-June-2014]