Computer Science
University of Manchester

The Algebraic Theory of Interaction Nets

R. Banach

Technical Report UMCS-95-7-2

The Algebraic Theory of Interaction Nets

R. Banach

Department of Computer Science
University of Manchester
Oxford Road, Manchester, U.K.
banach@cs.man.ac.uk

Recent technical reports issued by the Department of Computer Science, Manchester University, are available by anonymous ftp from ftp.cs.man.ac.uk in the directory pub/TR. The files are stored as PostScript, in compressed form, with the report number as filename. They can also be obtained on WWW via URL http://www.cs.man.ac.uk/csonly/cstechrep/index.html. Alternatively, all reports are available by post from The Computer Library, Department of Computer Science, The University, Oxford Road, Manchester M13 9PL, UK.

The Algebraic Theory of Interaction Nets

R. Banach
Computer Science Dept., Manchester University, Manchester, M13 9PL, U.K.
banach@cs.man.ac.uk

Abstract: The theory of interaction nets, invented by Lafont, is re-examined from the algebraic hypergraph rewriting perspective. Supersimple nets are defined and discussed, and some related classes of nets, the polysimple and monosimple classes, are defined and investigated. Their static properties are established, and the invariants that need to be preserved by rewriting are investigated in detail. It is shown that in the general case, context-specific information may be used to ensure that rules actually preserve the characteristics of the rewritten net. Subordinate agents, which like logical constants for falsity may be introduced only in non-void contexts, are presented, and the ramifications of the theory in their presence are investigated, relating it to the simple and semisimple classes of Lafont. Under suitable conditions, describable in purely combinatorial terms, net rewriting systems possess Church-Rosser and Strong Normalisation properties usually associated with rewriting systems derived from logic. This leads to the reformulation of the proof nets of multiplicative linear logic including constants in an essentially logic-free way.

Key Words: Interaction net, Graph grammar, DPO rewriting.

1 INTRODUCTION

In the late 80's, two seemingly unrelated research initiatives produced in the end, some rather similiar programming schemes. On the one hand, the connectionist paradigm (Hillis (1985)) gave rise to the connection graphs of Bawden (1986). On the other, linear logic (Girard (1987)) gave rise to the notion of multiplicative proof nets (Girard (1988), Danos and Regnier (1989)), and ultimately to the interaction nets of Lafont (1990, 1991). These two schemes are closely related since one can view interaction nets as special cases of connection graphs, possessing additional structure that enables powerful safety properties to be proved.

The purpose of this paper is basically twofold. The first objective is to present connection graphs (CGs) and interaction nets (INs), and particularly their safety properties, from a different perspective, that of hypergraph rewriting. It turns out that CG and IN rewriting is ideally suited to description by the algebraic "double pushout" techniques of Ehrig and others. Some of the material restated here might be regarded as folklore in the linear logic community, but the author hopes that the logic-free and relatively explicit version given in the present paper might make it more accessible to non-specialists in LL. In this sense, the earlier parts of the paper are partly pedagogical; moreover, the extra precision latent in the hypergraph treatment enables aspects of interaction nets that are technically intricate to be stated and investigated more crisply than would otherwise be possible. The second objective is to thoroughly investigate the theory of various classes of "simple" INs, and in particular how the requirement of closure under rewriting for such classes is played out in the relationship between a rule and the context in which it is applied. A fair amount of relatively simple graph theory has to be built up to accomplish this. Below we outline the remaining sections of this paper in a little more detail.

In section 2 we introduce agents, ports, connection graphs, and the all-important port invariant which states that each port connects precisely two agents. The port invariant enables a simple but efficacious static analysis for articulation ports of, (alternatively for potential cycles in) a CG system to be constructed. This analysis comes in handy in sections 7 and 8 for defining classes of IN rules whose safety is statically decidable.

In section 3 we present the rewriting model of CGs as an instance of the "double push-out" hypergraph rewriting (HR) methodology. The extra detail and universal algebra perspective of HR actually turn out to be convenient for dealing with many aspects of CGs and later INs.

In section 4 we introduce the basic concept of principality which most clearly differentiates INs from CGs, and characterise the safety of IN rewriting as deadlock avoidance. We show that the preservation of principal acyclicity by rewriting is the criterion to aim for in establishing safety. We discuss (an easy adaptation of) Lafont's elementary criterion for confluence and an equally elementary one for normalisation which is most clearly stated in terms of the new HR concepts. The combination of the two leads to unique normal forms, a phenomenon more often found in rewriting systems derived from logic, showing that a logical basis is by no means necessary to have such desirable properties.

In section 5 we describe threaded hypergraphs which constitute the basic mechanism upon which our treatment of the more subtle invariants of net theory relies. Section 6 describes multiplicative nets, in part reformulating the insights of Danos and Regnier in a logic-free environment.

Section 7 is the heart of the paper. A straightforward way of generating multiplicative nets, using switched clustering assignments for agents, immediately gives rise to supersimple nets, and the relatively trivial criteria that guarantee that supersimple net rewriting is safe. We present some theorems that precisely describe rewrite rules and systems whose safety is statically decidable.

Section 8 goes on to describe polysimple and monosimple nets as special cases of supersimple nets. The internal theory of polysimple nets turns out to be richer than that for supersimple nets because of the stronger invariants that hold, but by the same token, the preservation of safety by rewriting makes stronger demands for context-dependent data. As previously, we present theorems on rewriting for systems whose safety is statically decidable.

In section 9 we introduce subordinate agents as a simple generalisation of Lafont's "agents with an empty class in their partition", and describe the changes to the preceding theory that their use demands, in particular characterising precisely the combinatorial aspects of Lafont's simple and semisimple classes of nets. The more subtle theorems that characterise safe rewriting in the presence of subordinate agents are given.

A background theme in sections 7 – 9 is that of sequentialisation, which shows how the various classes of interest, which are defined by combinatorial invariants, relate to the sequential construction processes more familiar from logic.

Section 9 summarises and discusses the preceding. Also addressed are issues related to concrete implementation, especially garbage collection and synchronisation. The question of type disciplines for INs, largely ignored in the bulk of the paper because of the emphasis on graph theoretic techniques, receives some attention.

In the appendix we reconstitute the proof nets of the multiplicative fragment of LL as instances of normalising interaction nets for which the Church-Rosser property holds for reasons other than the ones given in section 4, and for which a type discipline different to that proposed by Lafont is appropriate. This reformulation brings the essentially combinatorial nature of (at least this fragment of) LL into stark relief. It has been noted that LL often has little to do with logic and much to do with "plugging" (see eg. Bellin and Scott (1994)). In a sense, this paper takes the "plugging" approach to an extreme, by banishing logic to the background.

2 AGENTS, PORTS AND CONNECTION GRAPHS

An agent is an entity which typically has a label and a (label dependent) set of ports eg. Fig. 1.

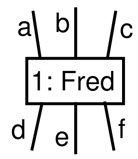


Fig. 1.

Here Fred is the label and a…f are the ports. The integer 1 is an agent identifier that serves to distinguish this particular Fred-labelled agent from other Fred-labelled agents; and so Fred and a…f are all strictly speaking attributes of the identifier 1. We will usually suppress agent identifiers (also port identifiers) when the context is sufficiently unambiguous. Agents can be linked together into nets by joining ports in such a way that the following holds:

Port Invariant: Each port is joined to precisely one other port.

Thus Fig. 2 is not a net while Fig. 3 is.

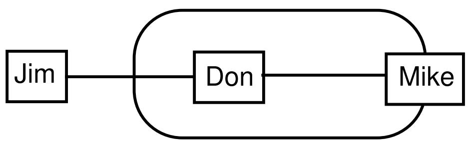


Fig. 2.

A net pattern (or just pattern) is obtained when we weaken the port invariant by replacing "precisely one" with "zero or one". Closed and open nets are also suitable names

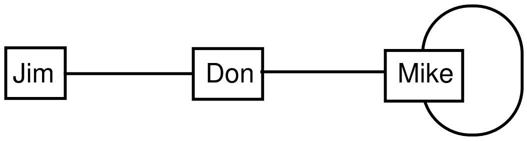


Fig. 3.

for what we have defined to be nets and net patterns respectively. Actually, it will prove useful in section 7 and later, when we study classes of patterns and nets, to use the name open net to refer to a subclass of patterns, having properties intermediate between nets and patterns proper, so ultimately there are three distinct notions. Mainly though, we will speak of nets and patterns. When we occasionally need to speak of subgraphs of nets or patterns for which agent are not regarded as having their full set of ports, we will use the general term subgraph.

The objective is to make nets into dynamic objects by allowing them to be rewritten according to rewrite rules. An example of a rule is illustrated in Fig. 4.

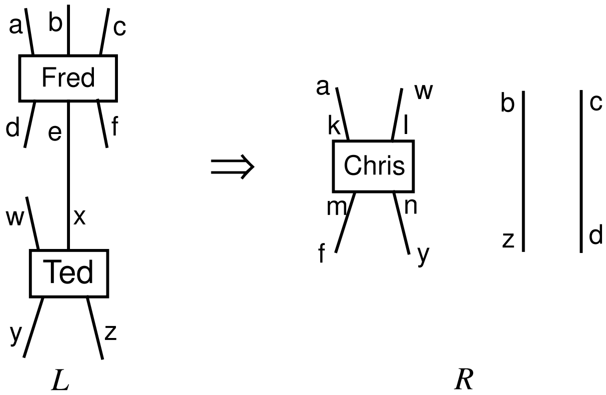


Fig. 4.

In the LHS L of the rule we have a pattern with precisely two agents, linked by a single pair of ports (all LHS's must be of this form). In the RHS R there is another pattern with a 1–1 correspondence between the free ports of the LHS and the free ports of the RHS, allowing for the fact that the port pairs (b, z) and (c, d) have been short circuited. The operational semantics of rules is rather obvious. First we find a homomorphic image of the LHS in the net we are rewriting i.e. we match the pattern. We insist that this matching is injective, which we can easily guarantee by having the two agents in the LHS bear different labels for example, but we don't insist on this labelling criterion.

Next we excise the image of the LHS, and replace it with a copy of the RHS, joining together the ports left dangling when they correspond to a short circuit. Since the matching is injective and we insist on a 1–1 correspondence between LHS ports and RHS ports, and only allow short circuiting of ports in pairs, the port invariant is maintained by rewriting. (Strictly speaking this is not true. If the images of a pair of LHS ports are joined in the rewritten net, and that pair is short circuited in the RHS, there is some ambiguity about the result. More complicated but similar cases can also arise. We will be more precise about these matters in the next section.)

Fig. 5 shows an example of rewriting using the above rule. Net G is the net being rewritten. Note the two ports of Fred connected together, and forming a configuration known as a breast (for rather obvious reasons — more on breasts in section 3). After removing the copy of L that G contains, a pattern D results, where the isolated blob is a placeholder for the two ports a and b of the copy of L that was removed from G. Splicing in R finally gives the net H. Note that the blob was in some sense necessary to make sure that the construction remembered to join the port of the D agent to port k of Chris.

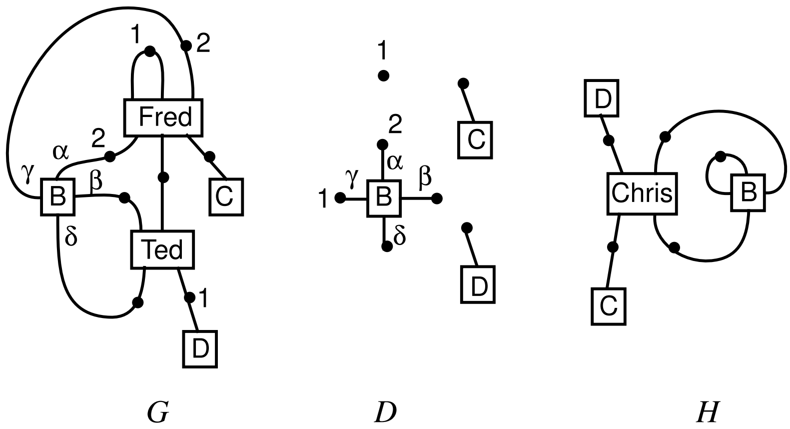


Fig. 5.

What we have so far are the connection graphs of Bawden modulo a change in the nomenclature, which is derived from interaction nets for uniformity with later usage.

The port invariant is very strong. It prevents sharing of subcomputations except in a highly explicit and controlled manner. One consequence of this is that cycles cannot arise “by accident” during rewriting; they must be explicitly introduced by rules, and this allows a dataflow analysis to be made for articulation ports, which is in practice more powerful than one might otherwise expect. By an articulation port we mean of course a port whose removal disconnects the connected component of the net or pattern in which it occurs. [N.B. Henceforth connected components will be refered to just as components.] Such an analysis will turn out to be very useful in subsequent sections on classes of various “simple” kinds of net. Since what it sets out to calculate is in principle uncomputable, we cannot but make an approximation, one of a set of such approximations generated by increasingly delicate static analyses. We briefly describe the analysis now.

For simplicity in future, we assume that computations always start with a fixed initial net with no free ports, containing two specific agents joined by a single port connection, and that there is a rule containing exactly this net as its LHS to enable the computation to get started. (The initial net is therefore obviously acyclic.) Now some definitions.

Definitions 2.1 Let L be an agent label and p one of L's ports. We call the pair (L, p) a germ. Nets and patterns do or do not contain instances of a given germ if there is or is not a suitably labelled agent in the net or pattern. Let $[a_1{:}w_1, a_1{:}w_1', a_2{:}w_2, a_2{:}w_2', \ldots, a_n{:}w_n']$ be a path through a net or pattern from port $a_1{:}w_1$, to port $a_n{:}w_n'$. Here the a_i are agent identifiers and the w_i and w_i' are ports of the corresponding agents. We assume that the port pairs $(a_1{:}w_1', a_2{:}w_2)$ etc. are actually joined in the net. A path is minimal iff no port instance occurs more than once in it and it never visits the same agent more than once. In particular, a minimal path never backtracks on itself. A minimal cycle is a minimal path in which the first port is joined to the last port (this being regarded as a single visit). [N.B. In the rest of this paper we will have little use for paths or cycles that are not minimal. Henceforth all paths should be assumed minimal even when it is not stated that they are. However, we will from time to time explicitly mention the minimality of paths just for emphasis.]

A sequence of germs $[(L_1, p_1), (L_1, p_1'), \ldots, (L_n, p_n')]$ covers a path as above iff it is of the same length as the path, and the following hold:

> a_1 is labelled by L_1, $w_1 = p_1$, $w_1' = p_1'$,
> a_2 is labelled by L_2, $w_2 = p_2$, $w_2' = p_2'$, …, etc.

The cover is said to be a C-cover iff all the germs in the cover are members of a set of germs C. Let Q be a rule and let $[a_1{:}w_1,\ldots, a_n{:}p_n]$ be a (minimal) path between the free ports $a_1{:}w_1$ and $a_n{:}p_n$ in the LHS of the rule. Then iff $b_1{:}x_1$ is the RHS free port that corresponds to $a_1{:}w_1$ in the LHS-RHS free port correspondence, and $b_m{:}x_m$ is the free port corresponding to $a_n{:}p_n$, and $b_1{:}x_1$ and $b_m{:}x_m$ are not short circuited in the RHS of Q, and there is a (minimal) path between $b_1{:}x_1$ and $b_m{:}x_m$ in the RHS of Q, then such a path in the RHS is called a corresponding minimal path for $[a_1{:}w_1,\ldots, a_n{:}p_n]$.

Theorem 2.2 Suppose a fixed system given. Let there be a set of germs C such that:

(1) If (L, p) occurs in a cover of a minimal cycle of the RHS of a rule of the system, then $(L, p) \in C$.

(2) If (M, x) occurs in a cover of a minimal RHS path corresponding to a minimal path having a C-cover between free ports in the LHS of a rule of the system, then $(M, x) \in C$.

Then every minimal cycle in any net generated by the system has a C-cover.

Proof. By induction on the structure of execution sequences. The initial net has no cycle thus giving the base case. Now suppose the theorem holds for some prefix $[G_0,\ldots, G_i]$ of an execution. The construction of the next net in the execution G_{i+1}, proceeds by excising the redex to make D_i, and then filling in the RHS and performing any necessary short circuits. Suppose then that G_{i+1} has a (minimal) cycle. If none of its ports were present in D_i, then it must have been introduced entirely within the RHS of the rule, whence (1) guarantees it has a C-cover. If some but not all of its ports were present in D_i, then there was a cycle having a C-cover which intersected the redex non-trivially in G_i, (because the LHS of the rule used is connected). By (2) the corresponding path that replaced the portion within the redex has a C-cover, so the G_{i+1} cycle has a C-cover. If all the cycle's ports were present in D_i, then either the cycle is a copy of a cycle of G_i, so has a C-cover by hypothesis, or it results from short circuiting a cycle of G_i which intersected the redex non-trivially, whence it also has a C-cover by hypothesis. ☺

Corollary 2.3 Let a set C good for theorem 2.2 be given. Then any instance of a germ not in C in a net generated by the system, is an articulation port.

Theorem 2.2 can be viewed as implicitly giving a specification that a set of germs C has to satisfy if it is to serve as a safe estimate of the germs occurring in coverings of all cycles that might be generated by the system. We can turn it into an algorithm for calculating such a C fairly readily as follows.

We introduce a two valued domain $\mathbf{2} = \{\mathbf{A}(\text{rtic}) < \mathbf{C}(\text{ycle})\}$ to represent (**A**) certainty that all instances of a given germ are not part of a cycle i.e. are articulation ports, and (**C**) possibility that they might be, respectively. Our analysis computes a function from germs to **2** by monotonically updating an array X indexed by germs with values in **2**, until a least fixed point is reached. We will use functional notation for X where convenient, eg. the new and old C's are related by $C = X^{-1}(\mathbf{C})$.

Algorithm 2.4

```
Initialisation:    For all germs (L, p) Do X(L, p) := A oD
                   Pend := the set of all rules (of the system)

While   Pend ≠ ∅
Do      Remove a rule Q from Pend
        For all germs (L, p)
        Do      If (L, p) occurs in a cover of a minimal cycle of the RHS of Q
                Then    If X(L, p) = A
                        Then    For all rules Q′ containing an instance
                                        of (L, p) in the LHS of Q′
                                Do Put Q′ in Pend oD
                                X(L, p) := C
                        fI
                fI
        oD

        For all minimal paths [a1:p1,…, an:pn] between free ports in the LHS of Q
                having X^-1(C)-covers {note that all such paths are of length 2 or 4
                due to the structure of the LHS's of rules}
        Do      If there is a corresponding minimal path for [a1:p1,…, an:pn]
                        in the RHS of Q
                Then    For all germs (M, x) occurring in covers of such
                                corresponding minimal paths
                        Do      If X(M, x) = A
                                Then    For all rules Q′ containing an instance
                                                of (M, x) in the LHS of Q′
                                        Do Put Q′ in Pend oD
                                        X(M, x) := C
                                fI
                        oD
                fI
```

oD

oD

Theorem 2.5 Algorithm 2.4 calculates an X such that $X^{-1}(\mathbf{C})$ is the smallest set of germs that satisfies the hypotheses of theorem 2.2.

The proof is relatively straightforward and is omitted.

As final comment on this matter we remark that the germ set C may be characterised as the solution of a set of dataflow equations parametrised by the rules in the system as follows:

$$C = \mathrm{NewCycles}_Q \cup \mathrm{PropCycles}_Q(C)$$

Here $\mathrm{NewCycles}_Q$ returns the set of germs covering all the RHS cycles of Q, while $\mathrm{PropCycles}_Q$ accumulates into C all germs in covers of corresponding minimal paths of C-covered LHS minimal paths of Q. Of course rules may destroy cycles as well as create them, but including this in the dataflow equations, or other versions of the present fairly crude analysis would destroy monotonicity.

3 HYPERGRAPH REWRITING

In the previous section nets were undirected graphs whose nodes were agents and where edges were given (somewhat implicity) by an equivalence relation on agents' ports in which the cardinality of each equivalence class was exactly two. This conforms to an intuition of ports as "bits of wire" that we can solder together to connect agents, and certainly reflects the way we draw pictures of nets. However, as noted parenthetically in the previous section, such a notion can be a little imprecise; and while it is possible to formalise the "wire" intuition, doing so needs more detailed machinery than the job seems to deserve. We will take a simpler approach to formalising nets, by making the equivalence classes of ports into nodes in their own right, and thus nets become bipartite graphs whose nodes are either agent nodes as before, or new port nodes, and whose edges (port edges) connect an agent node to a port node. In future a port connection will consist of the two port edges having a port node in common, and the unqualified word "port" will refer to a port edge. We will draw port nodes using black blobs to reflect their increased prominence. Thus our previous net is reincarnated in Fig. 6.

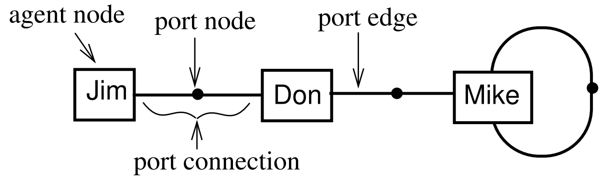


Fig. 6.

There is a well known correspondence between bipartite graphs and hypergraphs and we will now exploit it to reformulate the net rewriting model more precisely as an instance of the classical "double pushout" algebraic graph rewriting formalism of Ehrig (1979, 1986), as extended to "structures" and "algebraic specification grammars" of which hypergraphs are instances, in Ehrig et al. (1990). (Alternatively, hypergraphs are instances of the general algebraic theory for which a "single pushout" rewriting model is described in Lowe (1993).) There are a number of ways to express the net paradigm in this general framework but we will fix on the following.

We will work in a universal algebra framework. Assume given once and for all an alphabet of agent labels AL, and for each $L \in AL$, an alphabet PL_L of port labels, and consider the following signature.

Sorts: $\mathrm{SortAgentNode}_L$ ($L \in AL$), SortPortNode

Operations: $\mathrm{OperPortEdge}_p$ ($p \in PL_L$) : $\mathrm{SortAgentNode}_L \to \mathrm{SortPortNode}$

Let us agree that carrier sets and functions, of algebras of this signature are named by deleting the prefixes "Sort" and "Oper" from the signature names above. Then such algebras are hypergraphs whose nodes are port nodes and whose hyperedges are agents. Thus each hyperedge is connected to a set of nodes as determined by the ports belonging to its agent label. Since the $\mathrm{PortEdge}_p$ operations are total functions, each agent hyperedge automatically comes equipped with its full complement of ports, precisely one of each, giving one desirable invariant that is naturally generated by this formalism, and explains why we chose it. The other desirable invariant that we need, the port invariant, is not given to us automatically so we restate it as follows (we retain our previous notation for port occurences):

Port Invariant: For each port node x, $|\{\, a{:}p \mid \exists\, L \bullet a \in \mathrm{AgentNode}_L$ and $p \in PL_L$ and $\mathrm{PortEdge}_p(a) = x\}| \in \{2, 0\}$

This gives us the invariant for nets. Note that we have admitted isolated port nodes into nets. Nets containing such nodes will be regarded as being pathological, and will be called thus, since they violate our natural feeling for what a port connection should be; nets without such nodes will be called normal nets. Pathological nets can arise as a result of rewriting normal nets as we will see below, but note that once created, no isolated port node can ever play a part in a rewrite on account of the form of the LHSs of rules.

For patterns we replace "equal to 2 or 0" by "less than or equal to 2" in the invariant. The latter allows for free port edges as well as isolated port nodes. In patterns, the isolated port nodes are actually vital for a proper treatment of the rewriting mechanism, particularly as regards short circuits which were somewhat glibly presented in the previous section. In fact the blob we remarked upon in the rewriting example presented there was a case where isolated port nodes are unavoidable if a mathematically sound description is to be achieved.

Hypergraphs admit a notion of homomorphism in the sense of universal algebra. Such homomorphisms are given by collections of functions

$$f_{\mathrm{AgentNode}_L} : \mathrm{AgentNode}_L \to \mathrm{AgentNode}'_L$$
$$f_{\mathrm{PortNode}} : \mathrm{PortNode} \to \mathrm{PortNode}'$$

such that the collection of diagrams in Fig. 7 commutes.

$$\begin{tikzcd}
\mathrm{AgentNode}_L \arrow[r, "f_{\mathrm{AgentNode}_L}"] \arrow[d, "\mathrm{PortEdge}_p"'] & \mathrm{AgentNode}'_L \arrow[d, "\mathrm{PortEdge}'_p"] \\
\mathrm{PortNode} \arrow[r, "f_{\mathrm{PortNode}}"'] & \mathrm{PortNode}'
\end{tikzcd}$$

Fig. 7.

Given the preceding, patterns form the objects of a category $\mathcal{P}$ when as arrows, we take hypergraph homomorphisms which are injective on the port edges. (As a consequence of this, the arrows are also injective on the agent nodes, though not necessarily on the port nodes.)

Nets can be characterised as $\mathcal{N}$, the full subcategory of $\mathcal{P}$ such that none of the objects has any free ports.

Double pushout rewriting works according to Fig. 8.

$$\begin{tikzcd}
L \arrow[d, "g"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "d"] & R \arrow[d, "h"] \\
G & D \arrow[l, "l*"] \arrow[r, "r*"'] & H
\end{tikzcd}$$

Fig. 8.

A rule is given by the top line of the diagram, i.e. a pair of arrows from the interface hypergraph K. For us, $l : K \to L$ will always be the injection of the set of port nodes "round the edge" of the LHS hypergraph L into L, and $r : K \to R$ will always be similarly injective apart from short circuits of pairs of nodes of K. Rewriting completes the rest of the diagram as follows.

The first step in rewriting is to locate an image of the LHS hypergraph L in the hypergraph G being rewritten, the arrow $g : L \to G$. The injectivity of g on port edges (though not on port nodes) prevents an agent from spontaneously self-interacting via a reflexive rule (see the next section), an eventuality we regard as undesirable. The second step is to remove everything in $g(L)$ which is not in $g(l(K))$, which amounts to the two agent nodes and their associated port edges, along with the port node they share in common. This is unproblematic because of the port invariant, since the only port node of G that gets removed is (g of) the shared port node of L, for which both incident port edges also get removed, leaving nothing "dangling". This creates the context hypergraph D. The third step glues into D, copies of everything in R which is not in $r(K)$, and performs any short circuits required, by joining together pairs of ports which are free or isolated in D, corresponding to pairs of nodes of K on which r is not injextive.

Under the circumstances, the two squares in the diagram turn out to be pushouts in $\mathcal{P}$ in the sense that the first two steps construct the "pushout complement" D by building the left pushout square in an unconventional way, and the third constructs the honest pushout of $r : K \to R$ and $d : K \to D$ to give the result H. As we have hinted, the construction of the object D is in general problematic, but thanks to the shape of our L's, the fact that $l : K \to L$ is always the injection of the border into L, and the port invariant, a unique smallest D clearly always exists. Since honest pushouts exist in hypergraph algebras, we have a valid construction of H. (For a more detailed treatment of the double pushout construction see the cited references.) Note that we will use the above notation for a rewrite throughout the rest of the paper.

Lemma 3.1 Let $L \leftarrow K \to R$ be a rule, $g : L \to G$ a redex, and H the result of double pushout rewriting as above. Let the relations $\sim_g$ and $\sim_r$ on K be given by

$$a \sim_g b \iff g(l(a)) = g(l(b)),$$
$$a \sim_r b \iff r(a) = r(b).$$

Then $\sim^*$, the reflexive extension of the transitive closure on K of $(\sim_g \cup \sim_r)$, has equivalence classes that are either

(1) singletons, or

(2) non-singleton chains of alternating $\sim_g$ and $\sim_r$ edges, or

(3) non-singleton cycles of alternating $\sim_g$ and $\sim_r$ edges,

when $a \sim_g b$ (resp. $a \sim_r b$) is represented by a $\sim_g$ (resp. $\sim_r$) edge between nodes a and b.

Proof. Any node of K can be $\sim_g$-related to at most one other, by the port invariant for G. Similarly any node of K can be $\sim_r$-related to at most one other because short circuits are only permitted in pairs. This easily gives the conclusion. ☺

Theorem 3.2 Let $L \leftarrow K \to R$ be a rule, $g : L \to G$ a redex, and H the result of double pushout rewriting as above. Then we have that

(1) H is in $\mathcal{P}$,

(2) H is in $\mathcal{N}$ iff G is in $\mathcal{N}$,

(3) For a a node of K, $h(r(a))$ is an isolated port node of H iff its $\sim^*$ equivalence class is a cycle, or a chain for which the two outermost edges are $\sim_g$ edges, or a chain for which the two end nodes are mapped by $g \circ l$ onto free ports of G.

Proof. Consider a $\sim^*$ equivalence class. Suppose it is a singleton. Then the corresponding port node is free in G iff it is free in H, which is consistent with conclusions (1) – (3). Suppose the class is a cycle. Then the cycle corresponds to a series of breasts in G. These are transformed into an isolated port node in H, consistent with (1) – (3).

Now suppose the class is a chain. If the two outermost edges of the chain are $\sim_g$ edges, then the chain again corresponds to a series of breasts in G, which are thus transformed into an isolated port node in H, (the absence of one $\sim_r$ edge compared to the previous case making no difference.)

If both outermost edges of the chain are $\sim_r$ edges, then the chain corresponds in G to a series of breasts flanked on both sides by a pair of non-breast port edges. In this case, if both images in G of the flanking port nodes are free, then an isolated port node is created in H once again. If one image is free but the other not, then a free port is created in H. If neither image is free, then the free port nodes in D corresponding to the two ends of the chain are joined together.

Finally if one outermost edge of the chain is a $\sim_r$ edge and the other a $\sim_g$ edge, we have in G a series of breasts and one non-breast port edge. Then a free port edge is created in H iff the non-breast port edge was free.

All of the above is consistent with (1) – (3), and furthermore exhausts the possibilities, so we are done. ☺

Fig. 9 gives our rule from the last section in double pushout form. The morphisms $l : K \to L$ and $r : K \to R$ are given by the numbering on the port nodes, with the port nodes of R labelled by pairs of integers being the short circuits.

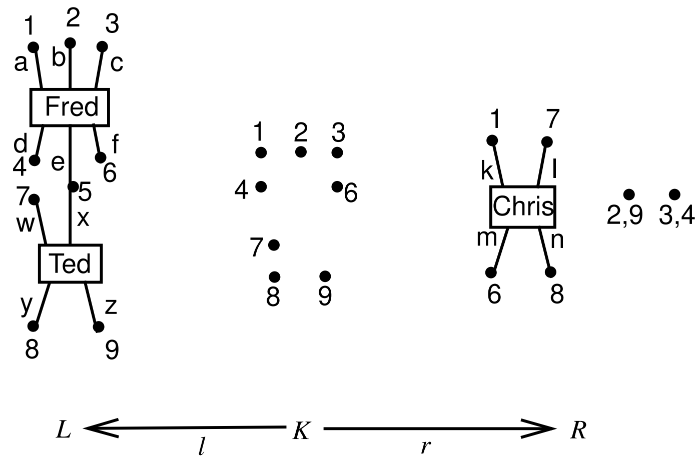


Fig. 9.

The bottom line of the double pushout rewriting diagram, for the application of the rule to the net G mentioned in the previous section, has already been given in the diagram there. We note that the choice of a blob as placeholder in D was made with malice aforethought.

Despite the technical usefulness of the hypergraph framework, we will continue to use the bipartite graph picture in the rest of this paper. It will be more convenient to speak of agent nodes and port nodes and of port edges between them, rather than of port operators from agent hyperedges to port nodes. In this representation, a pattern or net will be given by a septuple $(AN, PN, E, AL, \{PL_L\}_{L \in AL}, al, pl)$ where AN is the set of agent nodes, PN is the set of port nodes, E is a set of port edges given by ordered pairs (a, x) with a an agent node and x a port node, AL and $\{PL_L\}_{L \in AL}$ are the alphabets spoken of before, and al and pl are appropriate labelling functions. An appropriate suite of invariants must hold so that this notation is equivalent to the former. (Readers can easily see that the collection of rather ad-hoc invariants that we need via this approach is considerably larger that when we define things using hypergraphs — for this reason we regard the hypergraph definition as fundamental.) Also, despite the (a, x) notation for port edges, we will continue to use the $a{:}p$ notation when convenient. Regardless of which notation is used, fully labelling diagrams with all the attributes relevant to their definition becomes excruciatingly heavy on notation. Even including all the blobs can get to be tedious. For humanitarian reasons therefore, we will suppress as much of the notation as will still render diagrams meaningful in the sequel, rather as we have already been doing above.

4 PRINCIPALITY AND INTERACTION NETS, CONFLUENCE AND NORMALISATION

Previously we have discussed connection graphs using net nomenclature and a hypergraph formalism. Now we introduce the concept of principality which characterises interaction nets proper (at least for this author).

Definition 4.1 A principality assignment maps each agent label to a subset of its ports called principal ports.

We will indicate principal ports in conformity with Lafont by drawing them using heavy arrows.

Definitions 4.2 A port connection in a net where two principal ports meet is called a cut, and the corresponding port node is called a cut node. If neither port is principal, the port connection is called a link and the port node is called a link node. If just one port is principal, the port connection is called a principal arc, directed in the obvious way, with the port node called an arc node.

Principality restricts rewriting by enforcing the following rule invariant.

Rule Invariant: In every rule, the two port edges which connect the two agents on the LHS are both principal.

Thus net rewriting is (intentionally) an analogue of cut elimination. Our previous rule must therefore look like Fig. 10 say, in order to be a valid rule.

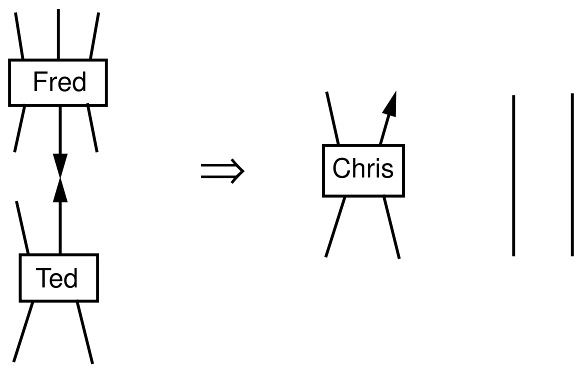


Fig. 10.

Definition 4.3 A reflexive rule is one such that in the LHS, both agents have the same agent label, and both port edges in the cut have the same port label.

A reflexive rule clearly possesses an inversion of its LHS which exchanges all similarly labelled port pairs on the periphery of the LHS.

Definition 4.4 A reflexive rule is symmetric iff the RHS is invariant under the inversion that exchanges all RHS port pairs corresponding to similarly labelled port pairs on the periphery of the LHS.

Here are a couple of easy propositions. The first one is a simple adaptation of a proposition of Lafont's.

Proposition 4.5 If each agent has at most one principal port (so that each pair of agents can form the LHS of a rule in at most one way), each pair of agents forms the LHS of at most one rule, and any reflexive rule is symmetric, then net rewriting is Church-Rosser.

Proof. We note that in a net or pattern G, two redexes with distinct images $g_1(L_1)$ and $g_2(L_2)$, can overlap on no more than some part of the images $g_1(l_1(K_1))$ and $g_2(l_2(K_2))$, of the two interface patterns K_1 and K_2 of any rules for which $g_1(L_1)$ and $g_2(L_2)$ are redex images. A standard result on double pushout rewriting gives a one step CR property for such a situation (see the cited references). Noting further that with the given restrictions, each cut in G can cause the pair of agents that it joins to be rewritten in at most one possible way, gives the remainder. ☺

Proposition 4.6 If in the bipartite hypergraph formulation, the RHS of each rule has fewer nodes (agents + ports) than the LHS, then net rewriting is strongly normalising.

The combination of these gives unique normal forms. It is almost, but not quite true, that the normalisation properties of cut elimination in the proof nets of (classical) multiplicative linear logic are an instance of this. See the appendix. The *bette noir* of net rewriting is the deadlock.

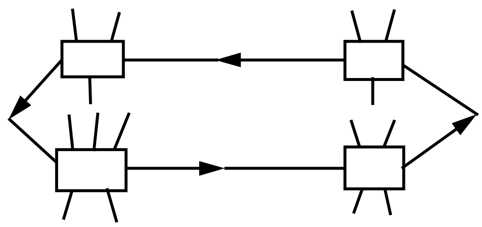


Fig. 11.

Assuming each agent in Fig. 11 has only one principal port, none of them will ever rewrite. Similar situations arise when agents are permitted to have more than one principal port, if every principal path out of an agent (directed and consisting entirely of principal arcs) returns to it. Principal acyclicity is thus a good thing for a net, assuring absence of deadlocks regardless of how many principal ports agents happen to have. Most of the rest of this paper is concerned with establishing and nurturing principal acyclicity on the basis that it is an essential ingredient for correctness.

5 THREADED HYPERGRAPHS

Definition 5.1 A hyperforest is an undirected acyclic bipartite graph containing two sorts of vertices: agent vertices and port vertices, and edges which connect an agent vertex with a port vertex. We insist that the pattern version of the port invariant applies, i.e. each port vertex is connected to at most two agent vertices. More formally it is given by a triple (AV, PV, H) with AV and PV the agent and port vertex sets and H the set of (ordered pair) edges, plus the invariant. Fig. 12 is an example.

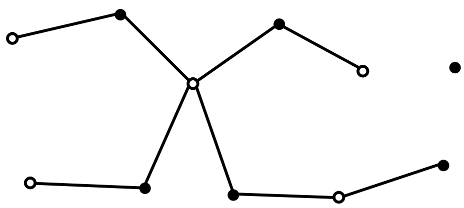


Fig. 12.

As before agents are drawn open while ports are drawn filled in. Hyperforests are unlabelled. A hypertree is a connected hyperforest.

Definition 5.2 A hyperforest (AV, PV, H) threads a pattern (AN, PN, E, AL, $\{PL_L\}_{L \in AL}$, al, pl) iff there are surjective maps

$$f_A : AV \to AN$$
$$f_P : PV \to PN$$
$$f_H : H \to E$$

such that f_H is also injective, and such that

$$(a, x) \in H \text{ iff } (f_A(a), f_P(x)) \in E,$$

which we can summarise by writing $t : T \to B$, where t denotes the three maps, T denotes the hyperforest, and B denotes the pattern.

Definition 5.3 A threading $t : T \to B$ is special iff the cardinality of the forest T (in terms of the number of trees in it) equals the number of components of B.

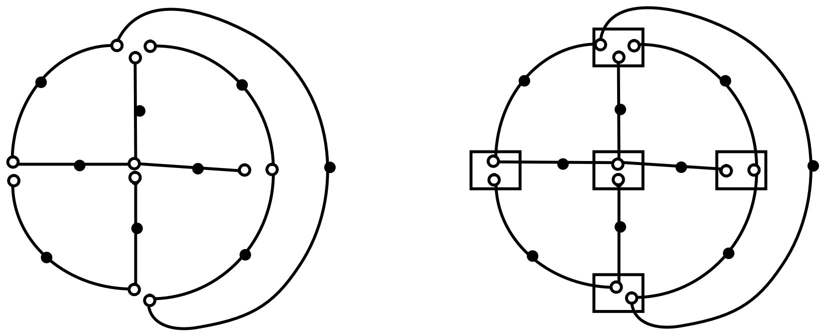


Fig. 13.

Fig. 13 provides an example of a threading. We see that a threading induces a partition of the set of ports of each agent node according to the agent vertex that they belong to; but not vice versa, because of the topological constraint of forestness.

A threading of a net by a hyperforest is a threading of the net regarded as a pattern (whose set of free ports is empty, as in the example above). Thus Fig. 13 is a threading of a pattern and a threading of a net, but not a special threading.

The option of having the specialness constraint and its repercussions are the main reasons that we make such a fuss about distinguishing nets and patterns. It turns out that we will mainly need the concept of special threadings of nets rather than patterns, except in discussions of sequentialisation (see below).

Consider a net with one component specially threaded by some hypertree. Starting at any point of the hypertree, the image of a pre-and-postorder traversal of the hypertree in the net yields a Girard longtrip round the ports of the net if we write the preorder occurences as $a{:}w^{\vee}$ and the postorder occurrences as $a{:}w^{\wedge}$. Thus the threading in Fig. 14 starting at the point ? yields the longtrip $a{:}t^{\vee}$, $a{:}t^{\wedge}$, $b{:}x^{\vee}$, $b{:}y^{\wedge}$, $b{:}z^{\vee}$, $b{:}z^{\wedge}$, $b{:}y^{\vee}$, $b{:}x^{\wedge}$. The correspondence between such longtrips and families of nonintersecting chords of a circle has been pointed out by Danos and Regnier (1989).

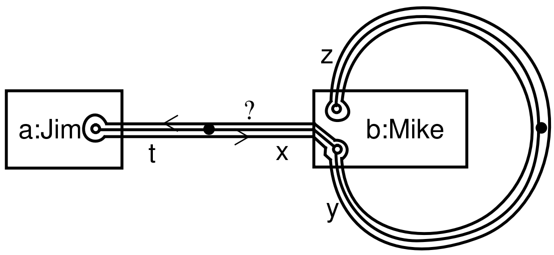


Fig. 14.

Definition 5.4 A threaded pattern or net (B, F) is a pattern or net B equipped with a set of threadings F (by possibly different hyperforests). Likewise for a specially threaded net, bearing in mind the additional constraint on cardinalities.

Definition 5.5 Threaded patterns form a category $\mathcal{TP}$. The arrows of this category are pairs

$$(h, \Phi) : (B, F) \to (C, G)$$

where $h : B \to C$ is a $\mathcal{P}$ morphism and $\Phi(F, G)$ is a relation on threadings $t : T \to B$ and $u : U \to C$ such that for each $t \in F$ there is a u, and for each $u \in G$ there is a t, such that $\Phi(t, u)$ holds; and whenever $\Phi(t, u)$, there is a hyperforest homomorphism $\phi(t, u) : T \to U$ such that Fig. 15 commutes.

The condition on ϕ almost forces h to be injective but not quite. For example, if B_1 is a connected component of B and $h(B_1)$ is within C_1 (a connected component of C), then every threading of B_1 extends to at least one threading of C_1, and every threading of C_1 arises as such an extension. The fact that there may be many different extensions of a threading of B_1, explains the relational form of the definition. Short circuits via h are allowed too under suitable circumstances, since two free port nodes in B may share the

$$\begin{tikzcd} T \arrow[r, "t"] \arrow[d, "\phi(t, u)"'] & B \arrow[d, "h"] \\ U \arrow[r, "u"'] & C \end{tikzcd}$$

Fig. 15.

same image in C without necessarily damaging the port invariant or the conditions on threadings.

Definition 5.6 The category $\mathcal{TN}$ of threaded nets arises as the full subcategory of $\mathcal{TP}$ given by the objects whose hypergraphs have no free ports, and the category $\mathcal{STN}$ of special threaded nets is the subcategory of $\mathcal{TN}$ where hyperforests obey the cardinality restriction.

Following Girard, pushouts in $\mathcal{TP}$ (especially when the result is in $\mathcal{TN}$ or $\mathcal{STN}$) may be viewed as combining two subsystems along an interface, with the threading representing some deeper structure suitably merged into a consistent whole. However our interest in pushouts is more to do with rewriting. Consider the double pushout rewriting diagram once more. All of the objects and arrows are now to be understood as pertaining to $\mathcal{TP}$, and so come embellished with data pertaining to threadings, see Fig. 16. In addition we may require the objects G and H to be members of $\mathcal{TN}$ or perhaps $\mathcal{STN}$.

$$\begin{tikzcd} (L, \Lambda) \arrow[d, "(g, \gamma)"'] & (K, 1) \arrow[l, "(l, \lambda)"'] \arrow[r, "(r, \rho)"] \arrow[d, "(d, \delta)"] & (R, \mathrm{P}) \arrow[d, "(h, \eta)"] \\ (G, \Gamma) & (D, \Delta) \arrow[l, "(l^*, \lambda^*)"] \arrow[r, "(r^*, \rho^*)"'] & (H, \Omega) \end{tikzcd}$$

Fig. 16.

The construction of the object D is mercifully no more problematic than previously given that K consists entirely of isolated port nodes, and so has only one possible (trivial) threading, (the set consisting of which is) denoted by 1. However the honest pushout of $(r, \rho) : (K, 1) \to (R, \mathrm{P})$ and $(d, \delta) : (K, 1) \to (D, \Delta)$ may fail to extend to a pushout in $\mathcal{TP}$ because threadings of R from P and of D from Δ may glue together to form a cyclic structure.

We remark that the above formalism threatens to become even more heavy than in the last section. Mercifully it can be avoided most of the time in favour of geometric intuition. But at the end of the day, to settle pathological and unexpected questions, one needs a precise framework to fall back on, which is why we have taken some space above to do so.

There is an embedding from $\mathcal{P}$ into $\mathcal{TP}$ that maps each object of $\mathcal{P}$ to the same object equipped with a single threading, wherein each port connection, free port, or isolated port node has its own separate tree in the threading hyperforest. This embedding extends to the double pushout rewriting construction, but is rather uninteresting. Given arbitrary threadings therefore, fewer “interesting” rewrites will exist than previously. In fact one might ligitimately wonder whether such a seemingly tortuous formalism was of any use whatsoever. It is no small tribute to Girard and co-workers that convincing computational models can be constructed on this basis. The following sections will bring this out and furthermore will answer the question of what all this has to do with principality.

6 MULTIPLICATIVE NETS AND PATTERNS

In the description above, different instances of L-labelled agents in a threaded net could support entirely unrelated partitions of their ports, for any agent label L. In this and the remaining sections, they can still support different partitions, but these will no longer be unrelated.

Definition 6.1 A partition assignment maps each agent label to a set of partitions of its ports.

This is the spur for us to impose another invariant.

Threading invariant: Given a threaded pattern, the partition on the port edges of an agent node induced by any threading is always a member of the set assigned to its label. Further, if (A, F) is a threaded pattern, then any choice of partitions for agents from among those assigned to their labels joins together to form a threading, and F is exactly the set of threadings given by all possible such choices.

This clearly restricts the possible threaded nets quite severely.

Definition 6.2 A switched partition assignment s is a partition assignment in which for each agent label L, for each partition of L's ports in $s(L)$, one set of ports in the partition is distinguished. We call partitions possessing such distinguished members, switched partitions.

Definition 6.3 Given a switched partition assignment, a threaded pattern (A, F) is a multiplicative pattern iff for every threading in F, discarding all the non-distinguished port edges from the pattern, results in a spanning forest (SF) of the pattern.

A spanning forest is obviously just a subgraph of the pattern in its bipartite graph representation, that contains all the nodes and is acyclic. For multiplicative nets we have the same constructions except that there are no free or isolated port nodes in the original pattern (although there may be some after deleting non-distinguished port edges), and for special multiplicative nets we demand further that the number of connected components in the SF equals the number of connected components in the original net (and so the discarding of non-distinguished port edges prunes the peripheries of the original threading hypertrees but does not disconnect them).

Much of what will concern us in the following two sections will entail proving that certain nets and patterns do in fact have SFs. The mechanisms for generating the partitions and threadings will be evident from context, so the proofs reduce to proving the acyclicity of the things that thread them, (or to deducing some contradiction from an assumption of their cyclicity). We will abuse language rather shamelessly by refering to these “things” as SFs anyway, this being a convenient shorthand for what would otherwise require a messy alternative nomenclature. Since we will always succeed in our objectives, this practice will cause no real confusion.

Let us look at an example of partitions and their threadings. Consider the two agents Jim and Mike and give them the switched partition assignments illustrated in Fig. 17 (where distinguished partitions are drawn heavy and others light).

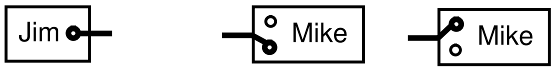


Fig. 17.

Consider the longtrip example net from before. Fig. 18 shows that this is a special multiplicative net.

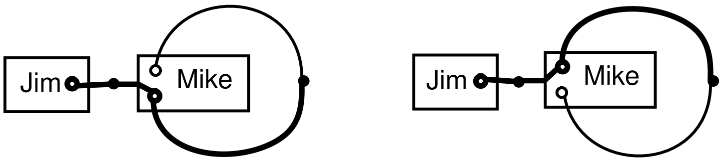


Fig. 18.

We see that every choice of a switched partition from Jim and Mike's assigned sets results in a threading of the net; and furthermore that deleting the non-distinguished ports from Mike, in each case results in a spanning tree, drawn heavy. In this example there are no isolated port nodes to discard. A choice for each agent node of a pattern or net, of one of the switched partitions assigned to its label, is called a switching of the net or pattern. In general there are $\Pi_{L \in A_L} n(L)^{|s(L)|}$ possible switchings of a net or pattern where $n(L)$ is the number of agents with label L, $|s(L)|$ is the number of partitions assigned to L, and the product is over all L.

We have seen that multiplicative nets can be cyclic. In effect the family of SFs they come equipped with serve to regulate the cyclicity and stop it getting out of hand, rather like a collection of whips keeping a number of spinning tops under control. None of this would be of any use however were it not for the fact that we can reliably propagate the required invariants via rewriting. For this we need to insist that when we perform a pushout in $\mathcal{TP}$ of multiplicative patterns, the sets of SFs of each component neatly glue together along the interface to give the set of SFs of the result.

Consider the pushout of $a : X \to A$, $b : X \to B$ where we have suppressed the threading data, and for convenience, X consists solely of isolated port nodes. Suppose the result Y is a special net with one component rather than just some pattern. Then lining up X in a row, the interface region of Y might look rather like Fig. 19.

Here we have shown the distinguished partitions and ports for some switching using heavy lines, and the rest thin as usual. Whatever switching we choose for the patterns either side of X, they have to fit together so that there is a series of pairs of ports (numbered in the diagram), such that each port belongs to a distinguished partition, and each pair forms a bridge of the agent spanning tree of Y between two components of the SFs of A and B. Since the switchings of A and B may be chosen independently, both the

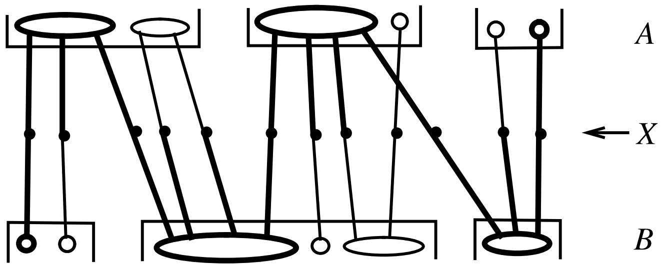


Fig. 19.

number of such pairs of ports, and the number of distinguished partitions of *A* and *B* adjacent to *X* in the result, are intrinsic properties of the interface.

Further study of the possible ways one can construct such interfaces rapidly degenerates into a vivisection of the theory of partitions, Stirling numbers of the second kind, and much that approachable programming systems should best avoid. We eschew these matters in favour of simple subsets of the theory which we describe in the following sections.

We refer the reader to Danos and Regnier (1989) for some more on the general theory, presented in a rather different way and embedded in a discussion of logic, but to aid the comparison, we make the following remarks.

Danos and Regnier discuss multiplicative connectives. Such a connective is a generalisation of a logical connective in sequent calculus, with a number of premises (each consisting of one or more formulae), and one conclusion (consisting of one formula). This may be viewed as an agent having a number of inputs (partitioned according to membership of the same premise), and one output. The inputs and output are typed by their corresponding formulae, the relationship between input and output types being a property of the connective. This yields a rule of formation (type compatibility) for plugging such agents together. Further, the output ought to be regarded as principal while the inputs are non-principal. If we assign the principal output to a singleton partition, we approach the formulation of this paper. To complete the picture, one tacitly needs to assume the existence of axiom agents, with two principal outputs, and a (type anti-compatible) rule of cut formation. The appendix describes all of this explicitly for LLM. Generalising, to allow the connective agents to have more than one strictly inequivalent introduction rule, leads to the study of sets of partitions (as in this section), and of orthogonal sets of partitions (so that cut elimination preserves membership of $\mathcal{STN}$).

7 SUPERSIMPLE NETS

7.1 Static Properties

In this section, we abandon the diverse miscellany of possible ways of constructing switched partition assignments for multiplicative nets and patterns in favour of a single mechanism.

Definition 7.1 A clustering assignment maps each agent label to a partition of its ports, a clustering. Each set in the partition will be called a cluster. We generate a switched partition assignment, which we will call a simple switched partition assignment (SSPA) from a clustering assignment, by doing the following for all labels in all possible ways.

1. Create a distinguished set of the partition by selecting exactly one port from each cluster.
2. Put all remaining ports into singleton sets of the partition.

We will draw clusterings by bracketing ports together, and other than in the next example, will typically omit the bracketing on singletons. Fig. 20 shows some examples.

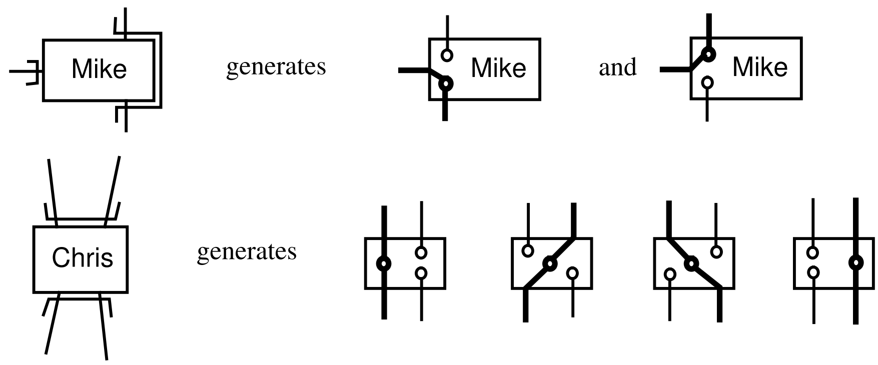


Fig. 20.

Definition 7.2 A multiplicative pattern whose threadings are given by an SSPA, will be called a supersimple pattern or SSP. An SSP without any free ports is a supersimple net or SSN. Clearly distinct switchings of an SSP (SSN) are generated by choosing different port edges in non-singleton clusters (NS clusters) of individual agent nodes to be distinguished. Given an NS cluster in an SSP, the agent node from which it emerges is called its owner.

Lemma 7.3 In an SSP, each port edge occurs in some SF of the pattern.

Proof. If it is in a singleton cluster it occurs in all SFs. Otherwise we can choose a switching of the NS cluster of which it is a member so that it is in an SF. ☺

Corollary 7.4 In an SSP, a path occurs in some SF of the pattern iff it contains at most one port edge from each NS cluster of each agent node of the pattern. We call such a path an SF-path.

In any SSP we can construct a partial ordering on NS clusters. This is not as useful in this section as it is in the one that follows.

Definition 7.5 Consider all port edges belonging to an NS cluster to be oriented from the port node to their owner (and all singleton cluster port edges to be unoriented). Let p and q be NS clusters. We will write $p < q$ iff there is an SF-path starting with some port edge a in p and ending with some port edge b in q, and such that both a and b (but not necessarily intermediate port edges) are traversed in the correct direction.

Proposition 7.6 For an SSP, the relation < is an irreflexive transitive partial order.

Proof. Transitivity is obvious. Suppose there were some a such that $q < q$ along some (minimal) cycle in an SSP. Because the cycle is minimal, we can assume that at most one port edge from each NS cluster occurs in the cycle, otherwise we could find a smaller cycle. By corollary 7.4 we can find a cyclic SF of the SSP, a contradiction. This gives asymmetry and irreflexivity. ☺

Proposition 7.7 Given a clustering assignment for agent labels, a pattern for which

(1) The subpattern consisting of singleton cluster port edges (and their adjacent nodes) is acyclic,

(2) < is an irreflexive transitive partial order,

is an SSP.

Proof. Suppose such a pattern contained a cyclic SF for some switching. If the SF-cycle consisted solely of singleton cluster port edges, it would contradict (1); if it contained a port edge from an NS cluster, it would contradict (2). ☺

Theorem 7.8 The cardinality of each SF of a SSP (in terms of the number of trees it contains) is constant.

Proof. It is well known and provable by induction that for any tree, the nodes outnumber the edges by 1. Thus for a forest of n trees

(*) nodes – edges = n.

Consider the set of SFs of an SSP. Each agent node and each port node is touched by each SF in the set. Furthermore different switchings differ only by which port edges in NS clusters are distinguished, always one distinguished port edge per NS cluster. Thus the LHS of (*) is constant over all switchings, and hence so is the RHS, the cardinality of any SF. ☺

Now we make the connection with principality.

Definition 7.9 A principality assignment is subordinate to a clustering assignment iff for each cluster, either all ports are principal or none of them are.

Fig. 21 shows two possible principality assignments for our Mike agent, subordinate to Mike's clustering assignment.

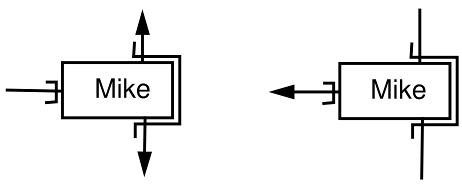


Fig. 21.

In future we will assume that all principality assignments are subordinate to clustering assignments without further comment. For emphasis we may speak of principal clusterings, principal supersimple nets etc. The following is the first key result, possibly the most crucial one in the whole paper in that it shows that the SSP property subsumes principal acyclicity.

Theorem 7.10 A (principal) SSP is principally acyclic.

Proof. Suppose not. Consider a prospective principal cycle. At least two consecutive port edges in this cycle must belong to an NS cluster since if at most one port edge from any NS cluster was in the cycle we could construct a cyclic SF by corollary 7.4. But then these two ports would both be principal or non-principal, leading to either a cut or a link in the supposed principal cycle. ☺

Definition 7.11 We say that a pair of nodes in an SSP are SF-strongly connected (resp. SF-weakly connected, SF-disconnected) iff there is an SF-path between them for every (resp. some, no) switching of the pattern.

Obviously SF-strongly connected implies SF-weakly connected. Given a particular switching of an SSP, we say a given pair of nodes are either SF-connected or not according to whether there is a path between them in the SF or not.

7.2 Sequentialisation

Above, SSPs were characterised by global criteria. This is most natural when working in a purely graph theoretic context. However the proof-theoretic origins most of the material in this paper provokes the question of whether classes of interest such as the SSPs, arise from a sequential construction process, describable by "sequent like" rules of inference. The prevalence of trees in the definition of SSPs, and the uniform and local notion of an SSPA, lend a justifiable sense of optimism to this enterprise.

Definition 7.12 A pattern is an open supersimple net (OSSN) if it can be constructed using the following sequent rules.

EMPTY: An empty pattern is an OSSN.

PORT: An isolated port node is an OSSN.

GRAFT0: Let A be an agent having l free ports grouped into n clusters. Let $ON_1 \ldots ON_m$ ($m \geq n$) be non-empty OSSNs. Let f map the m ON_i OSSNs onto the n clusters of A so that the total number of free ports of the OSSNs in $f^{-1}(p)$ (where p is a cluster of A), is at least as large as the number of free ports of p itself. Create an SSP ON_0 by doing the following for all clusters p of A: Make port connections between all free ports in p and (some) free ports of the OSSNs in $f^{-1}(p)$, (allowing for the degenerate case where some of the OSSNs might be isolated port nodes). Then ON_0 is an OSSN.

MIX: Let $ON_1 \ldots ON_m$ be non-empty disjoint OSSNs. Then ON_0, their union, is an OSSN. (If $ON_1 \ldots ON_m$ are not disjoint, alpha-relabel them.)

Fig. 22 provides an example of an OSSN. We describe one way of constructing it according to definition 7.12. First, port nodes 1, 2, 3, 4 are OSSNs, each in their own right, by PORT. To 1, 2, 3, 4 adjoin the agent D, creating an OSSN called X by

GRAFT0. To X adjoin the agent C, creating an OSSN called Y by GRAFT0. Port node 5 is an OSSN. Join Y and 5 using MIX to get Fig. 22.

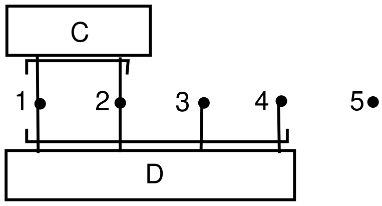


Fig. 22.

We now describe the relationship between SSPs and OSSNs.

Theorem 7.13 Every OSSN is an SSP.

Proof. An easy induction. An empty OSSN is an SSP, as is an isolated port node, and the union of a disjoint set of non-empty SSPs; so EMPTY, PORT and MIX generate SSPs (the latter from SSPs). For GRAFT0, note that any ON_i joined to the agent A, is joined via connections belonging to a single cluster of A. At most one of these connections is in any SF of SN_0, whence it is acyclic, and SN_0 is an SSP. ☺

Theorem 7.14 Every SSP is an OSSN.

Proof. Given an SSP, we must show that there is a schedule for constructing it according to definition 7.12. Consider the reverse of the construction steps, unEMPTY, unPORT, unGRAFT0, unMIX. This permits an induction on the size (agent nodes + port nodes) of the SSP.

Let X be an SSP. If X is empty use unEMPTY, if X is not connected, use unMIX, and if X is an isolated port node use unPORT. This leaves the unGRAFT0 case. If X contains no NS clusters, it is a forest, so the removal of any agent is an unGRAFT0, and splits X into strictly smaller SSPs. Otherwise X contains an NS cluster, so choose a <-maximal one, p say, and remove its owning agent A, leaving a subpattern $X - A$. We claim that not only is the result smaller, but the removal is an unGRAFT0 operation.

Firstly, all NS clusters owned by the owner of a <-maximal cluster are <-maximal (since if we cannot appropriately access any other NS cluster from p through the owner of p, neither can we access one from q, if p and q have the same owner). Secondly, no component T of $X - A$ is connected to A (in X) through more than one cluster of A. Indeed suppose such a T was connected to A through distinct clusters p and q of A. Choose a path through T that starts at a port connection in p and finishes at a port connection in q. Choose a switching of X so that as much as possible of this path is in the SF. The starting and finishing nodes of the path are in the SF, and are moveover SF-connected through A. Therefore there is at least one NS cluster in T such that the path passes through two of its port edges by corollary 7.4, since the SF is acyclic. But such an NS cluster contradicts the <-maximality of any NS cluster of A. ☺

The proof of theorem 7.14 was deliberately imprecise about how the various components of $X - A$ in an unGRAFT0 step were to be grouped into OSSNs. If we take each component to be an OSSN in its own right, we see that by always doing an unGRAFT0 step whenever one is possible (i.e. whenever there is an agent node in the net), we can avoid all uses of the MIX rule. In fact we have proved the following.

Proposition 7.15 Unless an OSSN has more than one port node and no agent nodes, it can be constructed without using the MIX rule.

A converse to this is that we can demand more stringent conditions of the GRAFT0 rule. In fact we could use the GRAFT1 rule of subsection 8.3 instead, and this would force essential use of MIX. However, GRAFT0 is more in keeping with the invariants maintained in this section.

Summarising, we see that the global characterisation of SSPs and the sequential characterisation of OSSNs are equivalent. In subsequent sections the corresponding distinction will turn out to be non-vacuous.

7.3 Rewriting

We now turn to the rewriting mechanism described by the double pushout diagram for $\mathcal{TP}$ in Fig. 16. Since we know that the construction is well defined in $\mathcal{P}$, it remains to consider the preservation of the invariants pertaining to supersimplicity for the threading data.

In the following, we will need to refer to the port nodes of the interface hypergraph K eg. a, b etc., and will need to refer to their images in the various other hypergraphs of the construction eg. $l(a)$, $d(a)$, $r^*(d(a))$, etc. To make the text smoother, we will refer to the images as "a … in L", "a … in D", "a … in H" etc.

Remark 7.16 Note that it is natural to regard a pair of nodes of K that are short circuited, as SF-strongly connected in R.

We start with a property of safe rewrites, in fact a pair of conservation laws. Note that the following theorems apply to rules $L \leftarrow K \rightarrow R$ whose LHSs are of arbitrary shape, as well as those in our standard form which just contain a single cut with all other ports of the LHS free, (provided K is precisely the set of "border" port nodes of L etc.). First some notation.

Let X be an SSP. We write $|X|$ for the number of connected components of X. We write $\langle X\rangle$ for the number of connected components of any SF of X.

Theorem 7.17 Let $G \leftarrow D \rightarrow H$ be a rewrite of an SSP G by a rule $L \leftarrow K \rightarrow R$ such that H is also an SSP. Let s be the number of short circuits in the rule. Then

(1) $\langle D\rangle = \langle r^*(D)\rangle + s$

(2) $\langle H\rangle - \langle G\rangle = \langle R\rangle - \langle L\rangle$

Proof. We tackle the short circuit free case first. Then D is isomorphic to $r^*(D)$, so $\langle D\rangle = \langle r^*(D)\rangle$ and we have (1). For (2) we appeal again to (*) from theorem 7.8. Following Danos and Regnier, consider the forest whose nodes are connected components of any SF of D or of L, and whose edges correspond to nodes of K, i.e. points at which the components cross the interface in order to form the overall SF of G. Thus by (*)

$$\langle D\rangle + \langle L\rangle - \langle K\rangle = \langle G\rangle \quad\text{and}\quad \langle r^*(D)\rangle + \langle R\rangle - \langle r(K)\rangle = \langle H\rangle.$$

Since K and $r(K)$ are isomorphic, subtracting gives (2). To cope with short circuits, note that the preceding includes the case of "fat short circuits", which are isolated agent nodes in the RHS of the rule having two singleton cluster port edges joined to two interface nodes of K. When we eliminate a fat short circuit in favour of a genuine short circuit, $\langle r^*(D)\rangle$ decreases by 1 since the effect of the short circuit must be to join exactly two distinct SF components into one by theorem 7.8. Whence the general case of (1). For (2) nothing changes since a fat short circuit forms an isolated SF component of R just as a genuine one does, and $s = \langle D\rangle - \langle r^*(K)\rangle = \langle K\rangle - \langle r(K)\rangle$. ☺

Remark. The above cheats slightly when there are short circuits since a short circuit isn't really a bona fide SF component of R, being rather a representative of a component of $r^*(D)$. This necessitates the term $\langle r(K)\rangle$ in the second equation above. If we only counted genuine SF components of R in a measure $\langle R\rangle^-$, equation (2) would become $|H| - |G| = \langle R\rangle^- - \langle L\rangle + s$. We chose not to do this because of the different measures needed.

Now a characterisation of safe rewrites.

Theorem 7.18 Suppose G is an SSP, and it is rewritten using a rule $L \leftarrow K \rightarrow R$ to produce H. Then H is an SSP iff:

(1) R is an SSP,

(2) For no nonempty even length sequence of distinct nodes of K, $[a_0, b_0, a_1, b_1, \ldots, a_n, b_n]$ is it true that

a_0 and b_0 are SF-weakly connected in D,
b_0 and a_1 are SF-weakly connected in R,
a_1 and b_1 are SF-weakly connected in D,
b_1 and a_2 are SF-weakly connected in R,
… … … … … … … … … … … …
a_n and b_n are SF-weakly connected in D,
b_n and a_0 are SF-weakly connected in R.

Proof. Suppose H is an SSP. Obviously its subpattern R is an SSP giving (1). For a contradiction suppose there is a sequence as proposed, and for all i there is a switching of D such that a_i and b_i were SF-connected in D. We can assume the individual clusters whose switches need to be set to achieve this SF-connection for a_i and b_i are distinct from those needed for a_j and b_j, else the two SF-paths would at least encounter the same agent node, and so we could "leapfrog" from a_i to b_j say, deleting the intervening subsequence. Working with the shorter sequence is just as effective for generating the one counterexample we need. So there is a switching of D that SF-connects each a_i to its b_i. A similar argument applies to R, and the b_i and their corresponding $a_{(i+1 \bmod (n+1))}$. Combining these two switchings we get a switching of H which has a cyclic SF.

Conversely, assume (1) and (2) and suppose there is a cycle in the SF for some switching of H. The cycle cannot lie entirely in $h(R)$ since R is an SSP and h is injective on port edges (which follows since g is similarly injective). Likewise it cannot lie entirely within $r^*(D - d(K))$ since r^* is injective there, and $D - d(K)$ is a subpattern of the SSP

G. So the cycle must intersect $r^*(d(K))$ nontrivially, and so must wander between $r^*(D)$ and $h(R)$, contradicting (2). Note that this result also covers short circuits by remark 7.16. ☺

Theorem 7.18 is not very helpful if our objective is to statically deduce something about the safety of sets of rules, since it requires knowledge of the SF-connectivity of D. However it is easy to define a conservative class of rules which are safe because the SF-connectivity of their RHS closely emulates that of their LHS.

Theorem 7.19 Suppose G is an SSP, and it is rewritten using a rule $L \leftarrow K \rightarrow R$. Suppose

(1) R is an SSP,

(2) For each switching t of R there is a switching s of L such that for all a, b in K,

$$a, b \text{ SF-connected by } t \text{ in } R \Rightarrow a, b \text{ SF-connected by } s \text{ in } L.$$

Then the pattern H produced is an SSP.

Proof. If there was a switching of H that created a cyclic SF, the cycle would have to wander between $r^*(D)$ and $h(R)$ as in theorem 7.18. This switching of H would induce a switching of R from which by assumption (2) we could find a switching of L that gave rise to a cyclic SF of G, a contradiction. ☺

Corollary 7.20 Rules $L \leftarrow K \rightarrow R$ such that for all a, b in K,

$$a, b \text{ are in the same cluster in } L \Leftrightarrow a, b \text{ are in the same cluster in } R,$$

satisfy the hypotheses of 7.10 and thus preserve the SSP property.

That corollary 7.20 is strictly weaker than theorem 7.19 is shown by a beautiful rule of Lafont's in Fig. 23, one of pair of rules for parsing polish strings. (It is no exageration to say that a large part of this paper resulted from a detailed study of this rule).

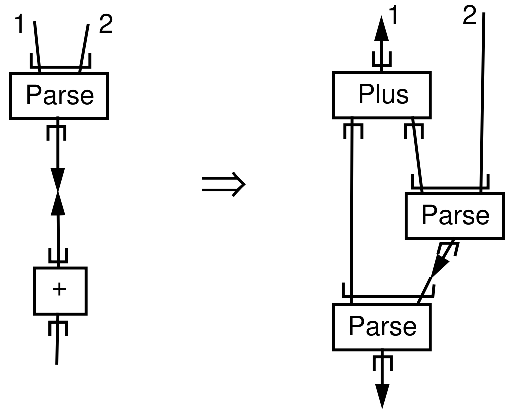


Fig. 23.

In this rule, the free ports (attached to) 1 and 2 in R do not belong to the same cluster in R though they are in the same cluster in L, whence corollary 7.20 fails for this rule. Nevertheless the conditions of theorem 7.19 hold good. In fact it can be checked that 1 and 2 are never in the same component of any SF of R, just as for L.

It is clear that rules good for theorems 7.19 or 7.20 are ultra-conservative in the sense that they always replace a LHS with a RHS for which the SF-connectivities generated by all possible switchings are a subset of those for the LHS. Such rules clearly preserve the SSP property because they connect SF components of D only in ways already sanctioned by the ways that they get connected in G. However this is not the only way of connecting SF components of D so that the result is an SSP. A more general way of ensuring safety is given by the Danos and Regnier theory of multiplicatives which in effect examines other possible ways that sets of trees can be glued together to give a single tree as a result. However, rather than plunge into the details of that somewhat intricate theory, which we promised to avoid, we can explore a further alternative using our dataflow analysis from section 2 to create a conceptually simpler extension of theorem 7.19, which uses global information to preserve the SSP property. We stress at this point that whereas theorems 7.19 and 7.20 are applicable to arbitrary redexes of SSPs because they use properties of contexts of such redexes that necessarily hold for all contexts, what follows uses global information about a system, hence properties of contexts that are merely sufficient. That is why the former can be expressed in terms of rewriting SSPs, which can be viewed as standing for the set of SSNs that result from instantiating their free ports in all possible ways consistent with the SSN invariants, while the latter must be given directly in terms of the SSNs generated by a system.

The next observation applies to rules with LHS in the standard form.

Lemma 7.21 Assume G is an SSP. Let $a \neq b$ in K. If a and b are SF-weakly connected in D, then a and b are in the same cluster in L.

Proof. Since L is connected, if a and b are in different clusters in L there is a switching of L that SF-connects them in L. If they were SF-weakly connected in D then there would a switching of G with cyclic SF, which is impossible. (On the other hand, if they are in the same cluster, then they are SF-disconnected in L, thus may be connected in some SF of D without harm.) ☺

Suppose we have a complete system which we will consider fixed for the rest of this section, and we perform the analysis of section 2 to get a set of germs C that cover all cycles generated by the system. Given a rule, we say that distinct a, b in K are C-related in L iff there is a C-cover of any minimal path from a to b in L. A little thought shows that the property of being C-related is symmetric and transitive for K. Thus its reflexive closure partitions K. If a and b in K are not C-related in L, we say that they are A-related in L.

Theorem 7.22 Suppose G is an SSN generated by the system, and suppose it is rewritten using $L \leftarrow K \rightarrow R$. Suppose C is good for theorem 2.2 for the system. Suppose:

(1) R is an SSP,

(2) For each switching t of R there is a switching s of L such that for all distinct pairs a, b in K, if a, b are C-related in L, then

$$a, b \text{ SF-connected by } t \text{ in } R \Rightarrow a, b \text{ SF-connected by } s \text{ in } L.$$

Then the pattern H is an SSN.

Proof. This is as for theorem 7.19 except we have to check that the extra possibility doesn't spoil things. Suppose *a* and *b* in in *K* are SF-connected by *t* in *R*, not SF-connected by any *s* in *L*, but are *A*-related in *L*. The *A*-relatedness means that they are disconnected and thus SF-disconnected in *D* for any switching of *D*. Thus SF-connecting them by *t* in *R* cannot create a cyclic SF. ☺

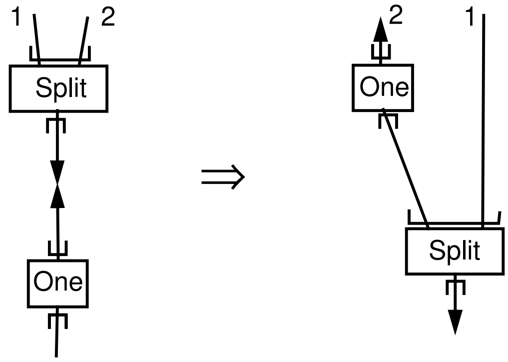


Fig. 24.

As an example, suppose we had the rule illustrated in Fig. 24 for what is essentially a form of unary division. Such a rule could be used in contexts where ports 1 and 2 of instances of Split were connected to some "root" agent which kept links to both outputs of the computation, because the NS cluster of Split would allow for the presence of a cycle in the evolving net without ruining the acyclicity of the SF.

However if such a "root" were not required, and the system was sufficiently benign that a *C*-covering analysis was able to discover that ports 1 and 2 never occurred in a cycle of any net generated by the system, then the 2-port cluster of Split would break up into two singleton "pseudoclusters" and the system could safely include a rule of the form in Fig. 25.

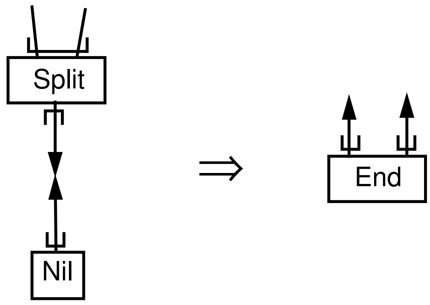


Fig. 25.

Clearly the above rule would be unsafe in a context where 1 and 2 were SF-connected in *D*, such as when a "root" was really needed.

Theorem 7.23 Systems of rules which satisfy the hypotheses of theorems 7.19, 7.20 or 7.22 generate nets which are deadlock free.

Proof. By theorem 7.10. ☺

8 POLYSIMPLE AND MONOSIMPLE NETS

8.1 Static Properties

In this section the distinction between patterns, nets, and open nets that we have been so careful to preserve, really comes into its own. As the reader will have noticed, most of the theory so far was rather insensitive to it.

Definition 8.1 A polysimple net (PSN) is a special supersimple net. If an PSN has precisely one component, we call it monosimple, an MSN.

We will come up with a suitable definition of polysimple pattern (PSP) shortly. It will turn out that there is no particularly useful notion of monosimple pattern different from that of polysimple pattern. On the other hand, open polysimple and monosimple nets naturally become notions different from that of PSPs as the next subsection shows.

Since PSNs are special cases of SSNs, all the safety results of the preceding section apply, and so we already know that (principal) PSN rewriting will be deadlock free. In this section we will elaborate the consequences of the additional cardinality restriction on the SFs of PSNs.

Proposition 8.2 In a PSN, all the port edges in each NS cluster are connected to port edges in singleton clusters, (unless we are dealing with a two port NS cluster, in which case Fig. 26, the only kind of breast permissible in a PSN, is also allowed).

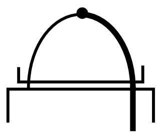


Fig. 26.

Proof. In a PSN, if a free NS cluster port edge is made not distinguished, it disconnects the component of the SF containing the port edge, so NS cluster port edges are not free. So let a and b be "face to face" port edges as in Fig. 27, in which p and q needn't necessarily be distinct NS clusters, but a and b are definitely distinct ports.

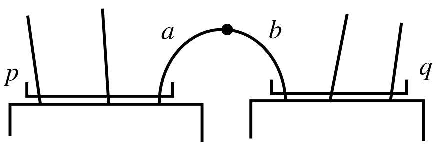


Fig. 27.

If $p = q$, either we have Fig. 26, or the conclusion, or both a and b can be made not distinguished, disconnecting the component of the SF containing them both. If $p \neq q$ and both p and q are owned by the same agent, making both a and b distinguished creates an SF cycle. Finally, if p and q have different owners, making both a and b not distinguished disconnects the component of the SF containing them both. ☺

Proposition 8.3 In a PSN, let X be an agent node owning an NS cluster p, whose ports are connected to singleton cluster ports. Then the set of singleton cluster port edges connected to the NS cluster is SF-strongly connected in the SSP obtained by deleting X and its associated port edges from the PSN. See Fig. 28.

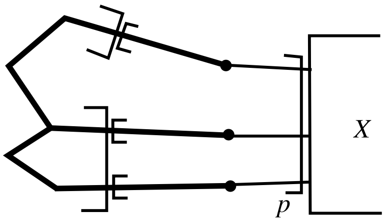


Fig. 28.

Proof. The singleton port edges must be in every SF, and hence for every switching, each distinct ordered pair of them, say a and b, must be SF-connected together via a unique (minimal) SF-path for cardinality reasons. We argue that such an SF-path cannot pass through the NS cluster p as follows. If it both entered and exited p, that would require two of p's port edges to be distinguished. If not, suppose the path entered X at the port edge joined to a and exited through some other cluster of X. This requires that the port edge joined to a be distinguished, and that there is an SF-path from the other cluster of X to b. Now choose the NS cluster port edge joined to b to be distinguished instead. This yields an SF-cycle. Therefore none of the NS cluster's port edges are involved in SF-connecting a and b, so that when X is excised, a and b remain SF-strongly connected in the remaining SSP. ☺

Given an NS cluster in a PSN, let q be the set of port nodes of the port edges in the cluster. By proposition 8.2 this determines the NS cluster (and thus its port edges and its owner) unambiguously. Henceforth we will permit ourselves to refer to an NS cluster as the set q of its port nodes, since we will often need to refer to these port nodes.

Definition 8.4 Let q be an NS cluster in a PSN. The subgraph of the PSN given by the union of all (minimal) SF-paths of the PSN, between pairs of members of q is called the proper scope of the NS cluster q, *proper_scope*(q). The subgraph of the PSN given by the owner of q, together with the port edges of q, together with *proper_scope*(q), is called the scope of q, *scope*(q).

It turns out that we can develop the partial order < of definition 7.5 to give an inclusion order on scopes. First some technicalities.

Lemma 8.5 Let q be an NS cluster in a PSN. Then

(1) *scope*(q) is connected.

(2) For each port node in *scope*(q), both incident port edges are in *scope*(q). This holds for port nodes in *proper_scope*(q) also, with the exception of the nodes in q itself.

(3) If for some switching, a path through the SF passes through *scope*(q), then altering the setting of switches inside *scope*(q) has no effect on the portion of the path

outside *scope*(q). In particular the entry nodes into, and exit nodes out of *scope*(q) remain the same.

(4) If p is an NS cluster in the PSN such that for some port node a of p, the port connection containing a is in *scope*(q), then for every port node b of p, the port connection containing b is in *scope*(q). Stated differently, if any node of p is in *scope*(q), then p is in *scope*(q).

Proof. (1) is obvious. (2) is also obvious by (1) and the observation that for any port node b, in *scope*(q) but not in q, and with only one port edge in *scope*(q), a path in *scope*(q) between members of q and passing through b would have to backtrack on the single port edge. For port nodes in q, (2) is obvious since they connect the proper scope of q and the owner of q. (3) holds because the distinguished/non-distinguished status of each port edge outside *scope*(q) remains the same, thus a portion of SF-path outside *scope*(q) remains unchanged.

To show (4), suppose there was an NS cluster p with port nodes a, b in p, and the port connection containing a was in *scope*(q). Choose a switching yielding an SF-path between two port nodes of q via the port connection through a, which must be possible by definition 8.4. Evidently $a \in$ *scope*(q). Thus by (2), both port edges adjacent to a are in *scope*(q), and thus the owner of p is in *scope*(q). Now change the switch on p so that the port connection through b is in the SF. There is still an SF-path between the two port nodes of q, and by (3) it enters and leaves *scope*(p) at the same crossing points, in particular via the owner of p. Therefore b is in *scope*(q). ☺

Lemma 8.6 In a PSN, let p and q be NS clusters. Then it is impossible to have $p \in$ *scope*(q) and $q \in$ *scope*(p).

Proof. Assume the contrary. Since $q \in$ *scope*(p), there is a minimal SF-path from some port node a of p to some port node b of p that goes via q and q's owner. Suppose such an SF-path traverses a port connection of q, from a node of q to q's owner (take the opposite orientation of the path otherwise). Then changing the switching of q alone, results in a tree of SF-paths from a to the various nodes of q. But this shows that it is possible to get between port nodes of q along minimal SF-paths without going through p or its owner since all these SF-paths from a have at least their first port edge in common by the port invariant. So $q \notin$ *scope*(p) after all, which is enough. In fact similar reasoning shows $p \notin$ *scope*(q), and taking the opposite orientation of the SF-paths to the one used, may be used to prove that there is a cycle in the SF, yielding more contradictions. ☺

Lemma 8.7 In a PSN, there is a partial order $\ll$ on NS clusters given by $p \ll q$ iff $p \in$ *scope*(q).

Proof. By lemma 8.6 and routine arguments. ☺

Proposition 8.8 In a PSN, suppose q is an NS cluster. Then

(1) If p is an NS cluster, and $p \in$ *scope*(q), then *scope*(p) $\subseteq$ *scope*(q).

(2) Every pair of agent nodes $\{x, y\} \subseteq$ *scope*(q) is SF-strongly connected within *scope*(q).

Proof. We proceed by induction on the ordering $\ll$. Suppose q is a minimal NS cluster under $\ll$. Then clearly all port edges in the proper scope of q are in singleton clusters, whence (1) and (2) for the base case.

For the inductive step, suppose for all $p \ll q$ the conclusions hold, and fix some such p such that $p \in scope(q)$. To get (1) we show that $scope(p) - scope(q) = \emptyset$. Suppose not. By lemma 8.5.(2) there must be some SF-path (call it [p*]) in $scope(p)$ with a nonempty subpath in $scope(p) - scope(q)$ (call it [p**]). Now choose an SF-path in $scope(q)$, call it [q*], that enters $scope(p)$ through the owner of p. This is possible because $p \in scope(q)$ and lemma 8.5.(2). It exits $scope(p)$ through some other agent node of $scope(p)$, X say, else there would be a cycle in the SF. Now set the switches in $scope(p)$ so that [p*] is in the SF of $scope(p)$. By hypothesis $scope(p)$ satisfies (2) so the SF of $scope(p)$ is in fact a tree and X is a node of that tree. By the nature of trees, there is a path within the tree from X to [p**]. This path first encounters [p**] either at one end of [p**] or in the middle. If at one end, then if the owner of p is only SF-accessible from X through [p**] (in the given switching), then we are done since the minimal path [q*], modified by the new switching in $scope(p)$ includes [p**]. If not, then there is a port node of p, SF-accessible from X only through [p**], so changing the switching of p to make the corresponding port connection distinguished has the same effect. If in the middle, selecting port nodes of p only SF-accessible from X through the two portions that [p**] gets split into, yields the same result, namely that [p**] $\subseteq scope(q)$. Whence $scope(p) \subseteq scope(q)$.

To get (2), note that the only port edges in $proper\_scope(q) - \bigcup_{(p \ll q)} scope(p)$ are singleton cluster ones, and thus form a subforest of any SF of the PSN. By lemma 8.5.(1), the various $scope(p)$'s must join the connected components of the subforest together. So given two agents X and Y in $proper\_scope(q)$, a path exists between them in $proper\_scope(q)$. Where this path is part of the subforest, it is an SF-path. Where it is not, it is in some $scope(p)$, where it can be modified into an SF-path if necessary by the inductive hypothesis. Including q and its owner is trivial. ☺

Corollary 8.9 Let q be an NS cluster in a PSN, and choose a switching. Then any non-empty intersection of an SF-path with $scope(q)$ is connected.

Proof. Routine. ☺

Theorem 8.10 The relation $<$ defined on scopes by $scope(p) < scope(q)$ iff $p \in scope(q)$, is a reflexive transitive partial order.

Proof. Routine. ☺

Corollary 8.11 The relation $<$ defined on proper scopes by $proper\_scope(p) < proper\_scope(q)$ iff $p \in scope(q)$, is an irreflexive transitive partial order.

Proof. Routine. ☺

Fig. 29 shows an MSN with three NS clusters, p, q, r. We have $p < r$, $q < r$, and p and q $<$-unrelated. Also $scope(q) < scope(r)$ with other scope pairs $<$-unrelated.

The fact that in a PSN, two NS cluters never meet each other "face to face" also allows us to project the partial order $<$ of definition 7.5 onto sets of agent nodes as follows.

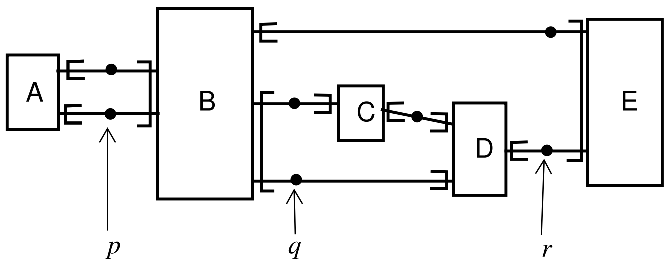


Fig. 29.

Definition 8.12 In a PSN, we partition the agent nodes into clumps. Let the clump containing a given agent node include all other agent nodes connected to it by paths built from singleton cluster port edges only. On clumps P, Q, define a relation $<^-$ given by $P <^- Q$ iff $P \cap scope(q) \neq \varnothing$ for some NS cluster q owned by some agent in Q. Let $<$ be the transitive closure of $<^-$.

Proposition 8.13 In a PSN, the relation $<$ on clumps is an irreflexive transitive partial order.

Proof. Routine, once we notice that $Q < Q$ would imply $q < q$ for some NS cluster q owned by some agent in Q. ☺

In Fig. 29, the clumps are {C, D}, {A}, {B}, {E}, with {A} < {B}, {C, D} < {B}, {B} < {E}.

8.2 Pattern Instantiation

In this subsection, we study the question of what constitutes a suitable definition of a polysimple or monosimple pattern (PSP or MSP). Our key motivation for this is that a PSP is the kind of object that can arise as a subpattern of a PSN. Our main tool will be induction, which thus brings out a relationship with sequential construction processes. However, these are not the material of sequentialisation proper, which is studied in the next subsection, and which suggests a different answer to the question of what definition to take for a monosimple pattern than the one given here. Hence the different nomenclature in the two subsections.

Definition 8.14 If in an SSP (or SSN), for some NS cluster q, deleting the owner of q together with all its port edges leaves each pair of port nodes of q SF-strongly connected, we say that q has a scope.

With this, we can formulate and prove a converse to proposition 8.3.

Theorem 8.15 In an SSN, if every NS cluster has a scope, then the SSN is a PSN.

Proof. We proceed by induction on the ordering $\ll$ as per lemma 8.5. Take the SSN and remove all port edges in all NS clusters. The result contains all and only all those port edges of the original SSN which are in singleton clusters, hence is an SSP for which the number of trees in the (one and only) SF equals the number of connected components of the SSP. (The agent nodes of each connected component in fact form a clump.)

Now add the NS clusters of port edges one by one in an order compatible with $\ll$. For the base case, let q be a $\ll$-minimal NS cluster. Its proper scope is a tree whose agent nodes form a clump. So replacing the NS cluster q, joins the proper scope clump to the clump that contains the owner of q. This preserves the property that the number of trees in any SF equals the number of connected components of the SSP, and the SFs generated are obviously acyclic since they are sub-SFs of the original SSP.

For the inductive step, suppose we have added all clusters $\ll q$. Therefore the SSP at this stage includes the whole of the proper scope of q, and each SF has the same number of components as does the SSP. Adding q again joins the proper scope clump to the clump that contains the owner of q, and again preserves the property that the number of trees in any SF equals the number of connected components of the SSP. The final result of this is the original SSN which is therefore a PSN. ☺

Now we are in a position to define polysimple patterns sensibly.

Definition 8.16 In an SSP, a descending path is an SF-path which starts at a port node of an NS cluster q, and for every NS cluster p, which both meets the SF-path in a port edge, and also does not have a scope, $p < q$.

Definition 8.17 An SSP is a polysimple pattern (PSP) iff each port connection contains at most one port edge from an NS cluster (aside from NS clusters like Fig. 26), and for each NS cluster q, either q has a scope, or for every port node a of q, there is a switching such that a can SF-access a free port node via a descending path.

Lemma 8.18 In a PSP, no switching generates an SF-component which is an isolated non-free port node.

Proof. Apart from Fig. 26, this is because each port connection contains at most one port edge from an NS cluster. ☺

The existential clause on switchings in the definition of a PSP can be strengthened to a universal one as follows.

Proposition 8.19 In a PSP, for each NS cluster q without a scope, for *every* switching of the PSP, every port node a of q can SF-access a free port node along a descending path.

Proof. By induction on $<$ for NS clusters. Suppose q is a $<$-minimal NS cluster without a scope and let a be a port node of q. Choose a switching of the PSP, such that there is a descending path from a to a free port node. We must show that changing the switching does not alter this property of a. Either the SF-path from a consists of singleton cluster port edges, and so is unaffected by change of switching, in which case we are done; or not. If not, let p be an NS cluster encountered by some such SF-path from a. In this case, since q is $<$-minimal, $p \nless q$ and so p has a scope. By lemma 8.5.(3), the SF-path enters and emerges from p's scope at the same points regardless of switching, so a can SF-access the same free port regardless of switching, giving the base case.

Now suppose we have the conclusion for all NS clusters $p < q$, and let a be a port node of q once more. Suppose q has no scope (otherwise there is nothing to prove). Choose a switching of the PSP, such that there is a descending path from a to a free port node. Once more, either the SF-path from a consists of singleton cluster port edges, and so is unaffected by change of switching, in which case we are done; or not. If not, let p be an NS cluster encountered by some such SF-path from a. Either p has a scope or not. If it has, then as above the SF-path enters and emerges from p's scope at the same points regardless of switching. If not, then $p < q$ and we can use the inductive hypothesis. Obviously, changing the switching of NS clusters disjoint from the path has no effect. ☺

Theorem 8.20 Any subpattern of a PSN is a PSP.

Proof. A subpattern of a PSN can be generated by removing some subset of its agent nodes along with all of their associated port edges. We can do this by removing one agent at a time. We use induction on this process. At each step, the "old PSP" will be the pattern as it is before the removal, and the "new SSP" is the SSP generated by the removal, whose credentials for being a PSP we are trying to verify.

First, the PSN is a PSP giving the base case. For the inductive step, suppose we take out an agent node X from the (old) PSP. Either X is in some scope or not. If not we argue as follows. Every NS cluster q that had a scope in the old PSP still has one in the new SSP, and every SF-path from an agent node of an SF-cluster q without a scope, which passed through X on its way to a free port node, now encounters a free port earlier. Further, for any NS cluster p without a scope encountered along the way, $p < q$ still holds, since $p \nless q$ would imply that p had a scope and $X \in scope(p)$ in the old PSP.

Otherwise suppose X is in the scope of an NS cluster in the old PSP. We must show that in the new SSP, any NS cluster q either has a scope, or all port nodes of q can SF-access a free port node along a descending path. Clearly if q had a scope in the old PSP and X was not in $scope(q)$, then q still has a scope afterwards and we are done. Suppose then X was the owner of q. After X's removal, there is no NS cluster q in the new SSP to worry about. Suppose finally that X was in $proper\_scope(q)$ in the old PSP. By proposition 8.8, all nodes of $scope(q)$ were SF-strongly connected within $scope(q)$; so given a port node a of q, and a switching, there was an SF-path within $scope(q)$ from a to X. Clearly this yields an SF-path from a to a free port node after X's removal, and this path does not wander outside of what remains of $scope(q)$. We must check that this path is descending.

Suppose in the new SSP, that the path wanders through a port connection of an NS cluster p, and p has no scope and $p \nless q$. Clearly X must have been in $scope(p)$ in the old PSP. Consider the path in the old PSP. It must at least have done the following. It must have started at a, entered $scope(p)$ at some point other than either X or the owner of p (if it entered at the former of these points it would have reached X without encountering p, if the latter then $p < q$), it must have traversed a port connection of p and exited through the owner of p, re-entered $scope(p)$, and finally reached X. But an SF-path cannot re-enter a scope once it has left it, by lemma 8.9; so we have a contradiction, completing the inductive step. ☺

Theorem 8.21 A PSP can have its free ports instantiated so as to generate a PSN. If each component of the PSP has at least one free port node, then the instantiation can generate an MSN.

Proof. By induction on $<$ for NS clusters. At each stage choose a $<$-minimal NS cluster q without a scope. Each port node a in q is either free already, in which case set $a' = a$, or there is a free port node a' SF-strongly connected to a. (If not, some SF-path from a to a free port would have to traverse an NS cluster p without a scope, but then by proposition 8.19, $p < q$, and q would not be <-minimal without a scope.) Choose such an a' for each a in q, in such a way that if a and b in q are SF-strongly connected, then $a' = b'$. (q having no scope merely means that there is at least one pair for which this fails.) Say there are n of these primed nodes altogether. Introduce a new agent node with n+1 singleton cluster free ports, and connect n of them to the a' port nodes, leaving one port node free. In the new SSP thus created, the NS cluster q has a scope. Further, each node that could access a free port node can still do so, and no new NS clusters have been introduced, so each port node in an NS cluster without a scope can still SF-access a free port node via a descending path, and the SSP is in fact a PSP. Continue in this manner until each NS cluster has a scope.

Unless the whole procedure was null and the PSP was a PSN to start with, the resulting PSP has some free port nodes. We can close off these remaining free ports using one port agent nodes to give an PSN, or if we pick one free port per component and join them using an agent with just the right number of singleton cluster ports, closing off any remaining free ports as previously, we get an MSN. The latter step only fails if we had no free ports at all in some component of the original PSP. ☺

Considering the last paragraph of theorem 8.21 shows why little is accomplished by trying to make a separate definition for monosimple patterns, at least from the point of view of instantiation.

8.3 Sequentialisation

In this subsection we repeat what we did in subsection 7.2, in the context of the stronger invariants that hold in this section.

Definition 8.22 A pattern is an open monosimple net (OMSN) if it can be constructed using the following sequent rules.

EMPTY: An empty pattern is an OMSN.

PORT: An isolated port node is an OMSN.

GRAFT1: Let A be an agent having l free ports grouped into n clusters. Let $ON_1 \ldots ON_n$ be non-empty OMSNs. Let f map the n ON_i OMSNs 1–1 onto the n clusters of A so that the total number of free ports of the OMSN $f^{-1}(p)$ (where p is a cluster of A), is at least as large as the number of free ports of p itself. Create an SSP ON_0 by doing the following for all clusters p of A: Make port connections between all free ports in p and (some) free ports of $f^{-1}(p)$, (allowing for the degenerate case where some of the OMSNs might be isolated port nodes). Then ON_0 is an OMSN.

Fig. 30 provides an example of an OMSN. We describe how to construct it according to definition 8.22. First, port nodes 1, 2, are OMSNs, each in their own right, by PORT. To 1, 2 adjoin the agent C, creating an OMSN called X by GRAFT1. Port nodes 3, 4 are OMSNs. To X, 3, 4 adjoin the agent D to get Fig. 30.

We now describe the relationship between SSPs and OMSNs.

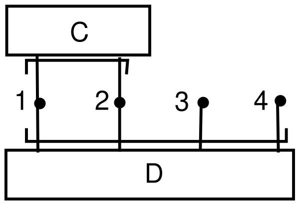


Fig. 30.

Theorem 8.23 Every OMSN is an SSP such that:

(1) The cardinality of each SF is one.

(2) Each NS cluster has a scope.

(3) Each pair of free ports is SF-strongly connected.

(4) Each free port is in a singleton cluster.

Proof. By induction. An empty OMSN is an SSP satisfying (1) – (4), as is an isolated port node; so EMPTY and PORT are sound. For GRAFT1, any ON_i joined to the agent A, is joined via connections belonging to a single cluster (say p) of A, with moreover, all ports in p connected to free ports of this ON_i. By the induction hypothesis, these free ports are all in singleton clusters and are SF-strongly connected to one another. Whence p has a scope, and the rest is immediate. ☺

Remark 8.24 In the above, (3) and (4) follow from (1) and (2) anyway.

Theorem 8.25 Every SSP satisfying (1) and (2) of theorem 8.23 is an OMSN.

Proof. Given an SSP as hypothesised (we will assume also (3) and (4) as necessary), we must show that there is a schedule for constructing it according to 8.22. Consider as in 7.14, the reverse of the construction steps, unEMPTY, unPORT and unGRAFT1.

Let X be an SSP of the given type. If X is empty use unEMPTY, and if X is an isolated port node use unPORT. This leaves unGRAFT1. Here the argument of theorem 7.14 gives us all except the preservation of the stronger conditions (1) – (4) of theorem 8.23. But noting that each NS cluster p of a $<$-maximal agent A has a scope, it is clear that the free ports created by A's removal are SF-strongly connected and in singleton clusters. The rest follows easily, so A's removal is an unGRAFT1 operation. ☺

We can generalise the above to include multicomponent ONs as follows.

Definition 8.26 A pattern is an open polysimple net (OPSN) if it can be constructed using the following sequent rules.

EMPTY: An empty pattern is an OPSN.

MIX: Let $ON_1 \ldots ON_m$ be non-empty disjoint OMSNs. Then ON_0, their union, is an OPSN. (If $ON_1 \ldots ON_m$ are not disjoint, alpha-relabel them.)

The two stage definition of OPSNs via OMSNs, ensures that any MIX step in constructing an OPSN is performed last. This simplifies the definition of the GRAFT1 operation, by avoiding the necessity to distinguish sets of free ports of an ON which are connected (and thus SF-connected), from sets which are not. The latter would introduce an element of global reasoning into the construction process which is rather against the spirit of sequentialisation. Similar considerations would apply with greater force were one to build a sequentialisation theory of PSPs proper, a perfectly feasable objective given the results of the preceding subsection. The following are quoted without proof.

Theorem 8.27 Every OPSN is a SSP such that:

(1) The cardinality of each SF is equal to the cardinality of the OPSN.

(2) Each NS cluster has a scope.

(3) Each pair of free ports is SF-strongly connected iff they are in the same component.

(4) Each free port is in a singleton cluster.

Theorem 8.28 Every SSP satisfying (1) and (2) of theorem 8.27 is an OPSN.

8.4 Interlude

It is worth pausing for a moment to review the static theory developed so far. This section has brought out the point that as we impose more stringent invariants on patterns and nets, the theories of patterns naturally inspired by instantiation, and those naturally inspired by sequentialisation begin to diverge. While instantiation demands that PSPs have the descending path property, the considerations of sequentialisation make OPSNs look like PSNs which have had some number of singleton cluster free ports stuck onto their agents at will. The differences are highlighted by the different appeals to well-foundedness of the $<$ relation made by the two approaches. The first needs $<$ to be well-founded from below, while the second needs $<$ to be well-founded from above. In the finite objects that we implicitly deal with, both are equally available, but it is clear that were one to allow patterns to become infinite, this would no longer be the case.

The various partial orders that we were able to define in subsection 8.1, on NS clusters, proper scopes, scopes, and clumps, also allow us to define a number of "reductions" of nets. Each such reduction is a special set of rewrite rules mainly concerned with connectivity properties, whose purpose is to attempt to reduce a net to a single node. The intuitive basis of such reductions must be obvious to the reader; they shrink scopes and clumps to single nodes, innermost scopes before outer ones. Formulated properly, such reductions characterise MSNs as those nets which are reducible to precisely a single node, and this has the important consequence of giving a recognition algorithm for PSNs and MSNs which is of quadratic rather than exponential complexity. The reductions themselves are very reminiscent of reductions of flow graphs, such as are described in Aho Sethi and Ullman (1986) for example. This is not surprising, since reductions of the latter type are typically driven by depth first spanning trees, which play a role exactly analogous to our SFs. For net-like systems, some of these issues have been explored in the theses of Girard's students (see eg. Danos (1990)), but we will not pursue these aspects of the theory further ourselves except as needed below.

8.5 Rewriting

We now have more than enough ammunition to attack the rewriting mechanism described by the usual double pushout diagram. As before, we know that the construction is well defined in $\mathcal{P}$, and we already have a feel for part of what is needed to preserve poly or momosimplicity from section 7.

First a strengthening of the conservation laws of theorem 7.17.

Theorem 8.29 Let $G \leftarrow D \rightarrow H$ be a rewrite of a PSN G by a rule $L \leftarrow K \rightarrow R$ such that H is also a PSN. Let s be the number of short circuits in the rule. Then

(1) $\langle D \rangle = \langle r^*(D) \rangle + s$

(2) $|H| - |G| = \langle H \rangle - \langle G \rangle = \langle R \rangle - \langle L \rangle$

Proof. By theorem 7.17 and the observation that $|G| = \langle G \rangle$ and $|H| = \langle H \rangle$. ☺

Next a characterisation of safe rewrites in STN.

Theorem 8.30 Suppose G is a PSN, and it is rewritten using a rule $L \leftarrow K \rightarrow R$ to produce H. Then H is a PSN iff:

(1) R is an SSP,

(2) For no nonempty even length sequence of distinct nodes of K, $[a_0, b_0, a_1, b_1, \ldots, a_n, b_n]$ is it true that

> a_0 and b_0 are SF-weakly connected in D,
> b_0 and a_1 are SF-weakly connected in R,
> a_1 and b_1 are SF-weakly connected in D,
> b_1 and a_2 are SF-weakly connected in R,
> … … … … … … … … … … … …
> a_n and b_n are SF-weakly connected in D,
> b_n and a_0 are SF-weakly connected in R.

(3) Every NS cluster p in D (resp. R) which does not have a scope in D (resp. R), has a scope in H.

Proof. Firstly any subpattern of a PSN is obviously an SSP. So (1), (2) and (3) are necessary by theorems 7.18, 8.3 and the observation that NS clusters in D which do not intersect L non-trivially in G are preserved unchanged in H, and thus have scopes. For sufficiency, by theorem 7.18, (1) and (2) guarantee that H is an SSN, since it has no free ports. That H is actually a PSN follows from theorem 8.15 and the above-mentioned observation. ☺

Note above that R is a subpattern of the PSN H, and thus must actually be a PSP by theorem 8.20. That this is not needed as an assumption in theorem 8.30 just reflects the fact that the conditions 8.30.(2) – (3) are sufficient to guarantee it already.

Theorem 8.30 isn't too impressive. Beyond theorem 7.18, it does little more than restate the definition of a PSN. This is because the requirements of preserving the PSN property essentially reduce to the complexities of something very like the theory of decomposable multiplicatives, a not significantly simpler subset of the theory of multiplicatives than the general Danos Regnier theory. Even in the case of LHSs of our standard form, i.e. two agents connected by a cut, if there are n free ports in the LHS, they could occur nested in up to n–1 distinct scopes of D. Fortunately however, the conservatism of the results following theorem 7.18 leans in a direction suited to preserving the PSN property, and even the MSN property.

Theorem 8.31 Suppose G is a PSN, and it is rewritten using a rule $L \leftarrow K \rightarrow R$. Suppose

(1) R is a PSP,

(2) For each switching t of R there is a switching s of L such that for all a, b in K,

  (i) a, b connected in $R \Rightarrow a$, b connected in L.

  (i) a, b SF-connected by t in $R \Leftrightarrow a$, b SF-connected by s in L.

Then the pattern H produced is a PSN.

Proof. We know by theorem 7.19 that H is an SSP since our condition (2) is stronger than theorem 7.19.(2). Observing that H will have no free ports if G has none, we conclude that H is an SSN. We must show that under the conditions stated, each NS cluster of H has a scope. Obviously any such cluster with scope wholly contained in D, has a scope in H. Equally, any such cluster with scope wholly contained in R, has a scope in H. This leaves those NS clusters of D without a scope in D and those of R without a scope in R.

Suppose q is an NS cluster of D without a scope in D and let there be no SF-path between port nodes a' and b' of q in some arbitrary switching u of D. Since D is a subpattern of G, it is a PSP, so a' and b' can each SF-access a port node of K (in D). Choose a switching t of R, and let s be a switching of L that witnesses the SF-connectivity of the nodes of K mentioned in (2).(ii) above. The cluster q certainly has a scope in G. Therefore, given u and s, there is an SF-path from a' to b' in G. Suppose a, b in K are nearest to a', b' respectively in this path in G. Obviously there is an SF-path between a and b in G, a subpath of the one just mentioned. The SF-path may wander between L and D in G, but in any event, the sections nearest a and b are both in L. Each section in L corresponds to a pair of port nodes of K, connected and SF-connected by s in L, and for each such pair, there must therefore be a path in R, connected and SF-connected by t in R. Hence there is an SF-path from a' to b' in H. Since u and t were arbitrary, this shows that q has a scope in H.

Now suppose q is an NS cluster of R without a scope in R and let there be no SF-path between port nodes a' and b' of q in some arbitrary switching t of R. Let s be a switching of L that witnesses the SF-connectivity of nodes of K mentioned in (2).(ii) above. Choose a switching u of D. Since R is a PSP, a' and b' can each SF-access (by t in R) one or more port nodes of K, say $\{a_1 \dots a_m\}$ and $\{b_1 \dots b_n\}$ respectively. We claim that it suffices to show that there is an SF-path $[w_{ij}]$ between a_i and b_j in H, given t and u, for all pairs a_i, b_j. For then, for at least one pair a_i, b_j the end portions of $[w_{ij}]$ nearest to a_i and b_j, are in D, (else a' and b' would be SF-connected in R, a contradiction); and hence, exactly one such pair will witness the SF-connectivity of a' and b'. Since a' and

b' were arbitrary in q, and t and u were arbitrary switchings of R and D, we conclude that q has a scope in R.

We substantiate the claim. By assumption, each pair a_i and b_j are connected (via the owner of q) and SF-disconnected by t in R. Therefore they are connected and SF-disconnected by s in L by (2).(i) and (2).(ii). This means that for every path $[z]$ say, between a_i and b_j in L there is at least one NS cluster p in L such that $[z]$ passes through two port edges of p, (else lemma 7.4 would furnish an SF-connection between a_i and b_j in L). We can choose all such p through which $[z]$ passes via *two* of p's edges, to not have scopes in L. (If such a p did have a scope, choose a variant of $[z]$ through *proper_scope*(p) and continue to work with that.) Equally, we can choose any other NS cluster p' through which $[z]$ passes, i.e. where $[z]$ passes through only *one* port edge of p', to either have no scope in L, or for the path to pass through the distinguished port of p'. (If p' has a scope in L, all its agents are SF-strongly connected in *scope*(p') and so $[z]$ may be rerouted through *scope*(p') so as to go through the distinguished port of p' without altering the entry or exit points in *scope*(p').) $[z]$ thus passes through L and meets at least two distinct components of the SF of L. However all the non-distinguished port edges that $[z]$ passes through belong to NS clusters which have scopes in G. Thus the agents in those scopes are SF-strongly connected, and in particular, there is an SF-path between a_i and b_j in G, given s and u. Each portion of this SF-path in L is SF-connected by s (and hence connected) in L, and thus corresponds to a path which is SF-connected by t (and hence connected) in R by (2).(ii). ☺

Corollary 8.32 Suppose G is a PSN, and it is rewritten using a rule $L \leftarrow K \rightarrow R$. Suppose

(1) R is a PSP with every component connected to at least one node of K,

(2) For each switching t of R there is a switching s of L such that for all a, b in K,

 (i) a, b connected in $R \Leftrightarrow a$, b connected in L, and

 (i) a, b SF-connected by t in $R \Leftrightarrow a$, b SF-connected by s in L.

Then the net H produced is a PSN with the same number of connected components as G. Therefore if G is monosimple then so is H.

Proof. Observing that nodes of distinct components of D are connected in H via R iff they are connected in G via L, and that no new isolated components are introduced in R, we see that the number of connected components remains the same. Further, if that number happens to be 1, we have rewritten one MSN to obtain another. ☺

Corollary 8.33 Rules $L \leftarrow K \rightarrow R$ such that for all a, b in K,

(i) a, b connected in $R \Leftrightarrow a$, b connected in L.

(ii) a, b are in the same cluster in L, $\Leftrightarrow a$, b are in the same cluster in R

satisfy the hypotheses of theorem 8.31 and corollary 8.32 and thus preserve the PSN property.

We see that Lafont's rule, treated in the previous section preserves both polysimplicity and monosimplicity, as well as supersimplicity.

Again as in the previous section, we can exploit the dataflow analysis from section 2 to create an extension of the above results, which uses once more the concepts of C-relatedness and A-relatedness.

Theorem 8.34 Suppose G is a PSN generated by the system, and suppose it is rewritten using $L \leftarrow K \rightarrow R$. Suppose C is good for theorem 2.2 for the system. Suppose:

(1) R is a PSP,

(2) For each switching t of R there is a switching s of L, such that for all distinct pairs a, b in K, if a and b are C-related in L, then

(i) a, b connected in $R \Rightarrow a$, b connected in L, and

(ii) a, b SF-connected by t in $R \Leftrightarrow a$, b SF-connected by s in L.

Then the net H produced is a PSN.

Proof. By theorem 7.22, and the observation that conditions (1) and (2) are stronger than the corresponding conditions of theorem 7.22, we know that H is at least an SSN. Now observe that in a pattern or net, each NS cluster which has a scope must occur in a cyclic subgraph. So since dataflow analysis is safe, each NS cluster of L or R will have a C-cover, and the free port node pairs of L and R that belong to potential incomplete scopes of such NS clusters will be C-related (as will the free port nodes of L and R that belong to potential incomplete scopes of NS clusters in D). Therefore the conditions for the proof of theorem 8.31 are only required for C-related pairs of nodes of K, as given in (2).(i) – (ii) above. ☺

We note that the more delicate circumstances under which the conditions of theorem 8.31 are required to apply give us greater flexibility in disconnecting and reconnecting parts of D which are not involved in the preservation of the integrity of scopes.

9 SUBORDINATE AGENTS AND SIMPLICITY

In this section we direct our treatment to dealing with the simple and semi-simple nets of Lafont, especially as described on p.102 of Lafont (1990), though in truth it is hard to be categorical about having captured the intended meaning exactly, due to the masterpiece of succinctness that is that single page.

To accomplish the above, we need to introduce subordinate agents. Now just as a subordinate principality assignment has the effect of decoupling considerations of deadlock avoidance and threading structure, by subsuming the former in the latter via theorem 7.10, so the notion of a subordinate agent decouples considerations of SFs and their acyclicity from connectedness issues in the underlying net. But first, to motivate what follows, we introduce Lafont's nets.

9.1 Lafont's Nets

Lafont's nets (LNs) are typically permitted to have free ports. Furthermore, their agents must have exactly one princpal port, and the notion of clustering is used only for the non-principal, or auxiliary ports. They are thus special cases of our patterns, if one puts the principal port of an agent into a singleton cluster. Fig. 31 shows an MSN which is not a net in Lafont's sense.

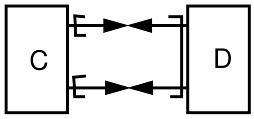


Fig. 31.

The one principal port (OPP) restriction means that half of the conditions for the Church-Rosser theorem (proposition 4.5), hold right away for LNs.

The notion of simplicity for LNs is defined via sequentialisation, and our subsections 7.2 and 8.3 supply much useful material for understanding this. The LN sequent rules are LINK, GRAFT and CUT. LINK is analogous to our PORT rule, except that instead of introducing an isolated port made, it introduces a piece of wire with two free ends. This highlights the difference between our formulation of ports and port connections, and the original "piece of wire" intuition of Lafont. One advantage of the wire fromulation is that enables us to distinguish between a piece of wire with two free ends and a closed loop of wire; giving us the option of regarding the former as acceptable and the latter as pathological. Both reduce to an isolated port node in our formulation, which is why such nodes are regarded both as being somewhat pathological in a net, but nevertheless permitted by the net port invariant.

The GRAFT rule is an instance of our GRAFT1 rule of definition 8.22, in that the free ports in the clusters of an agent A are joined to an equal number of LNs. Since the principal port of A is not regarded as being in a cluster, it always ends up free after a GRAFT operation. The other rule CUT, allows the free principal ports of two LNs to be joined to form a single cut connection, which is always an articulation point of the new net so formed.

If is clear on the basis of theorems 8.23 and 8.25 that every LN thus constructed is an OMSN in the algebraic formulation, and therefore our GRAFT1 rule subsumes both GRAFT and CUT. Fig. 32 shows the essential step of a "CUT avoidance" theorem, which transforms any construction of an LN using Lafont primitives to one using ours. The reverse of this step shows that any non-empty OMSN satisfying the OPP property is an LN.

Semisimplicity for LNs is the concept that results if we admit EMPTY and MIX into the repertoire of construction rules. We have already argued at the end of subsection 7.2, that the combination of GRAFT1 and MIX is equivalent to GRAFT0, so it is easy to see that semisimple LNs are just the OSSNs that satisfy the OPP property.

The final question that we have to deal with, and one that actually gives rise to this whole section, is that of subordinate agents. In Lafont (1990), the partition into clusters of an agent's auxiliary ports is allowed to include the empty set (of ports). First of all, such a use of the notion of partition is sufficiently non-standard that we abandon it immediately, and call agents having this feature subordinate agents. Second, we must describe the intended use of this concept. In a GRAFT operation, each cluster of ports of

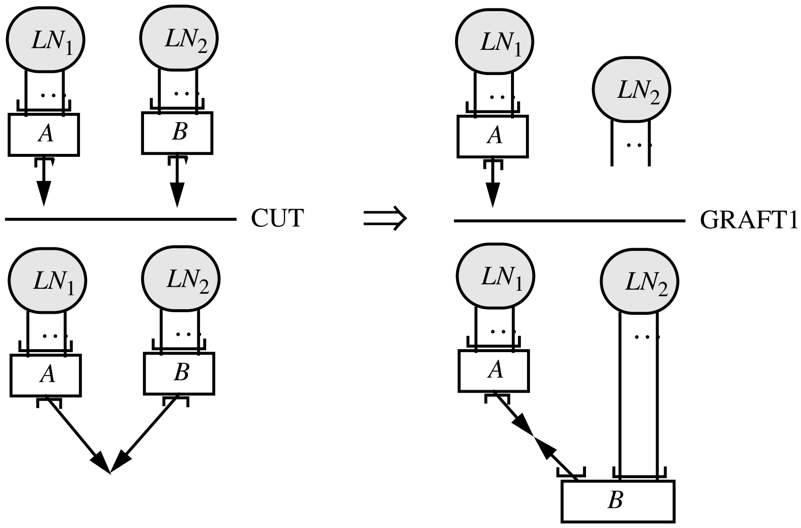


Fig. 32.

the agent introduced, gets connected to some net. In the limit of an empty cluster, there are no visible connections, but the net still has to be there. So the introduction of a subordinate agent is subordinate to the presence of such a net. Hence the name.

The intuition comes from logic. Consider the constant False. If one knows that X is a valid formula, then one can assert the truth of $X \vee$ False, but not otherwise. Certainly one cannot assert the truth of False alone. False is therefore a subordinate symbol which cannot validly be asserted independently. Below we extract the combinatorial content of this idea and place it in the algebraic framework.

9.2 Subordinate Agents and Relativisation

In future, an agent with label L, will not only come equipped with an L-dependent set of ports as previously, but also an L-dependent set of probes. If the set of probes is empty then we have a normal agent as before, while if it is non-empty we have a subordinate agent. If there are k probes, we have a k-subordinate agent. A 0-subordinate agent is normal of course. Fig. 33 shows a 2-subordinate agent S with probes x and y. We always show probes dashed and directed.

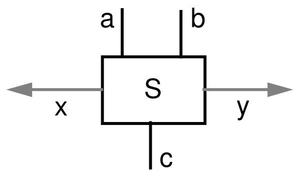


Fig. 33.

There are two key features of subordinate agents and their probes. The first is that unlike ports, which have to conform to the social niceties of the port invariant, probes directly invade other agents. Subordinate agents thus parasitise other agents (known as their hosts or victims), and although a given agent invades a fixed number of victims (not all necessarily distinct), depending on how many probes it has, it may itself be the victim of an arbitrary number of invasions by probes owned by other agents.

The second key feature is that probes are “invisible”, in the sense that although one knows what probes each agent in a net has, one is not normally privy to the knowledge of what agent is the victim of any particular probe. Later on, arrangements of victims will become closely intertwined with such concepts as connectedness and switching, and to qualify as a net, a graph of agents and ports must possess a set of arrangements of probes’ victims, each conforming to the invariants of the day, and non-empty if at least one agent of the graph has a probe. The assertion that such and such a graph is a net, therefore amounts to the statement that the appropriate set of arrangements is non-empty when necessary. Verifying that a given graph is indeed a net, can thus be done at worst by exhaustive search.

A particular choice of victims for the probes in a net will be called an E-structure for the net. Assuming that S and T are 1-subordinate agents, and that M and N are normal agents, Fig. 34 shows a net together with some E-structures for it. Fig. 34.(a) shows the visible structure of the net while Figs. 34.(b) – (e) show the invisible part. It will later turn out that Fig. 34.(a) is an ESN, and Figs. 34.(b) – (e), the set of E-structures that verify this property.

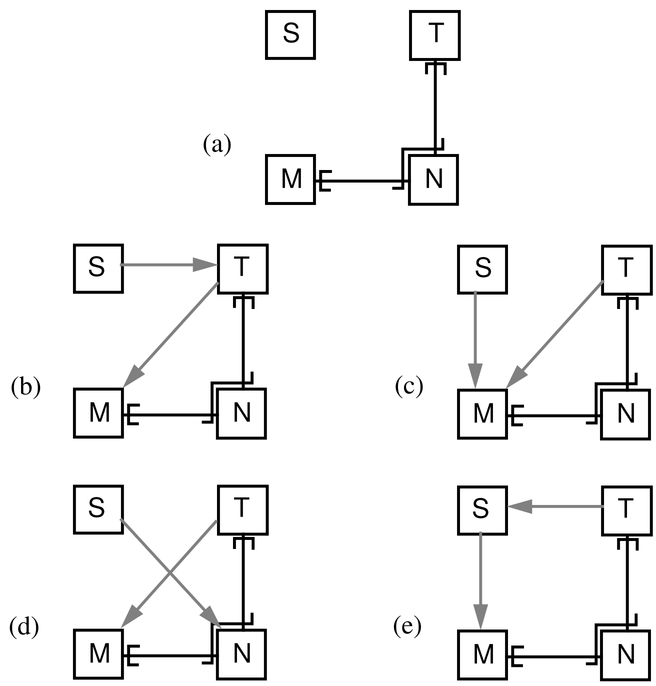


Fig. 34.

We will need to allow patterns to have free probes, because of the double pushout construction, but in patterns which are being regarded as open nets or nets, every probe must have a victim.

With the above motivation, we can begin the more formal constructions associated with subordinate agents. We must retrace the steps of the preceding sections in the light of the new data pertaining to the objects under discourse. Fortunately, we can do this rather quickly. As we do so, the concepts we deal with acquire an E-prefix to denote their new versions. The E- stands for "extended".

To retrace section 3, to each agent label $L \in AL$ we associate an alphabet of probe labels SL_L, possibly empty. To the signatures considered there, we must also add the operations

$$\mathrm{OperProbe}_s\ (s \in SL_L) : \mathrm{SortAgentNode}_L \to \bigcup_{L \in AL} \mathrm{SortAgentNode}_L$$

Above, union over agent labels is used because a probe may invade a victim regardless of its agent label, so the carrier of the union sort is the union of the carriers.

Definition 9.1 An E-pattern is a set E of partial algebras over the extended signature, such that their subalgebras over the signature of section 3 are all isomorphic and form a pattern (i.e. are total). We write an E-pattern as (P, E) where P is (a representative of) the common pattern of the algebras in E. An open E-net is a set E of total algebras with a common pattern ON, written (ON, E). An E-net is a set E of total algebras, whose common pattern is a net, written (N, E).

By abuse of language, when (P, E) is an E-pattern etc., we will when convenient, allow ourselves to refer to probes of agents of P.

Definition 9.2 A subE-pattern of an E-pattern (P, E) is an E-pattern (P, E') where $E' \subseteq E$. Similarly for a subE-net. An E-subpattern of (P, E) is an E-pattern (P', E') where P' is a subpattern of P, and E' is the induced set of subalgebras of members of E. (If a probe of an agent in P' has victim outside P' in some partial algebra of E, that probe is free in the corresponding partial algebra of E'.) Similarly for E-subnets, (which by the preceding remark needn't always exist).

The "subE" convention (contrasted with "E-sub") will be useful when we impose increasingly strong invariants on (partial) algebras, forcing exclusion of some of them.

The partiality of the Probe_s operations allows for free probes in E-patterns, and we see that in Fig. 34, (a) showed the common net, while (b) – (e) showed the underlying algebras of E.

Definition 9.3 In a partial algebra of the extended signature, let there be a path consisting of port edges and/or probe edges between port or agent nodes x and y. (Probe edges are unoriented $(a{:}s, f(s)(a))$ pairs with a an agent, s a probe of a, and $f(s)$ the Probe_s operation on s and a, necessarily defined on a — in other words probe edges as we would intuitively imagine them.) Then we say x and y are E-connected. The reflexive transitive closure of the symmetric relation of E-connectedness, yields the E-components of the partial algebra.

Obviously, E-components consist of the contents of a number of (ordinary) components of the common pattern P of an E-pattern. When we speak of E-components of an E-pattern where we only have the visible part, ie. P, to hand, we are tactly asserting the existence of a suitable partial algebra in the other data of the E-pattern. Similar remarks apply to (open) E-nets.

Parial algebras of the extended signature support a natural notion of homomorphism which demands that the image of an operation is at least as well defined as its pre-image. Thus a free probe may map onto a probe with or without a victim, but a probe with a victim may only map onto another one with a victim.

Definition 9.4 E-patterns form a category $\mathcal{EP}$. The arrows of $\mathcal{EP}$ are pairs

$$(h, \Psi) : (P, E) \to (Q, F)$$

such that $h : P \to Q$ is an arrow of $\mathcal{P}$, and $\Psi(E, F)$ is a maximal relation on the partial algebras in E and F such that whenever $\Psi(e, f)$ holds, there is a partial algebra homomorphism $\psi : e \to f$ that extends $h : P \to Q$. Open E-nets and E-nets form the sub-categories of $\mathcal{EP}$, $\mathcal{OEN}$ and $\mathcal{EN}$, with the proviso that $h : P \to Q$ is an arrow of $\mathcal{P}$ and $\mathcal{N}$ respectively, and all the algebras in E and F are total.

When $\Psi(e, f)$ holds for partial algebras e and f in definition 9.4, we say that e and f witness the arrow $(h, \Psi) : (P, E) \to (Q, F)$. For brevity we can also say that e witnesses the arrow $h : P \to Q$ if the rest is understood.

The rather weak condition on Ψ above captures the notion that there are merely *some* E-structures for P and Q that make $h : P \to Q$ extend to the whole signature, and is connected with the idea that such E-structures are invisible.

Now we turn to the business of threadings. The key point is that a probe must contribute to a threading iff it has a victim. To permit this we must allow agent vertices in a hyperforest to be adjacent, but still insist that port vertices may only be adjacent to (at most two) agent vertices. Thus hyperforests now fail to be bipartite graphs.

Definition 9.5 A hyperforest threads a partial algebra iff it threads it in the sense of definition 5.2 except that

(a) agent vertex to agent vertex edges are mapped injectively onto probe edges with victims,

(b) free probe edges are not in the image of the f_H map.

Definition 9.6 A threading $t : T \to e$ of a partial algebra e by a hyperforest T is E-special iff the cardiality of T equals the number of E-components of e.

Definition 9.7 A threaded partial algebra is a partial algebra equipped with a set of threadings (by possibly different hyperforests). Likewise for an E-specially threaded partial algebra.

Definition 9.8 A threaded E-pattern (P, E, V) is an E-pattern (P, E) such that each partial algebra e in E, comes equipped with a set of threadings $t : T \to e$ from V (and V is exactly a set of such threadings).

Definition 9.9 Threaded E-patterns form a category $\mathcal{ETP}$. The arrows of this category are triples

$$(h, \Psi, \Phi) : (P, E, V) \to (Q, F, W)$$

such that $h : P \to Q$ is an arrow of $\mathcal{P}$, $(h, \Psi) : (P, E) \to (Q, F)$ is an arrow of $\mathcal{EP}$, and $\Phi(V, W)$ is a relation on threadings as follows. Suppose for a given $e \in E$ and $f \in F$, $\Psi(e, f)$ is witnessed by a partial algebra morphism $\psi : e \to f$. Then $\Phi(V, W)$ asserts that for every threading $t : T \to e$ of e in V, there is a threading $u : U \to f$ of f in W, such that $\Phi(t, u)$ holds; and whenever $\Phi(t, u)$, there is a hyperforest homomorphism $\phi(t, u)$ such that Fig. 35 commutes.

$$\begin{tikzcd} T \arrow[r, "t"] \arrow[d, "\phi(t,\, u)"'] & e \arrow[d, "\psi"] \\ U \arrow[r, "u"'] & f \end{tikzcd}$$

Fig. 35.

Definition 9.10 The subcategories $\mathcal{ETN}$ and $\mathcal{OETN}$ of (open) threaded E-nets arise as subcategories of $\mathcal{ETP}$ such that all algebras are total, and in the case of $\mathcal{ETN}$, no agents have free ports. $\mathcal{ESTN}$ is the subcategory of $\mathcal{ETN}$ for which all threadings of all algebras are E-special.

So threaded E-nets can be viewed as sets of threaded nets with extra machinery to deal with the probes. The reader will be able to imagine the situation for multiplicative E-nets, so we move straight on to the criteria pertaining to supersimplicity and associated invariants.

As in previous sections, we use clusters to generate sets of partitions for agent nodes. The rules of section 7 for generating these partitions need to be modified as follows.

1. Create a distinguished partition by selecting exactly one port from each cluster, and include all probes for which the given agent node is either the owner or the victim.

2. Put all remaining ports into singleton clusters of the partition.

Note that the number of connections in the distinguished partition of an agent node is no longer a function of the agent label because of the arbitrary number of probes it may host. Fig. 36 shows the partitions for an agent with four ports, and two probes, which is the victim of infestation by three other probes.

As usual, a choice of a distinguished partition for each agent is called a switching. Given a switching, deletion of non-distinguished port edges yields an object which we shall call an ESF because of the inclusion of probe edges, and which we will normally require to be acyclic. Similarly we have the concepts of ESF-connectedness, and also of ESF-(strong, weak, dis-) connectedness. More generally, the inclusion of the probe edges in any of our preceding notions will be signalled by the prefix E- as noted above, and

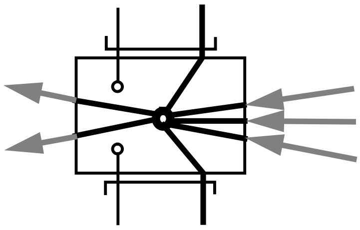


Fig. 36.

whenever the corresponding definitions are obvious, we will pluck such notions out of thin air as we need them, eg. E-scope, E-proper-scope.

Definition 9.11 An E-pattern (P, E) is an E-supersimple pattern (ESSP), iff for every $e \in E$, every switching (of every partial algebra in E, together with the deletion of the non-distinguished port edges) yields an acyclic ESF, and E is exactly the set of all partial algebras with common pattern P, for which this is true. If a pattern P is the common pattern of some ESSP (P, E), then we say that P is an ESSP. Analogous considerations hold when there are neither free ports nor free probes, in which case we are dealing with an E-supersimple net (ESSN).

Clearly, demanding the maximality of E in definition 9.11 makes a pattern P into a unique ESSP if it is one at all. Actually, it is easy to characterise such patterns.

Theorem 9.12 A pattern P is an ESSP iff P is an SSP.

Proof. The only non-trivial part entails noting that an SSP P has a partial algebra with acyclic ESF, namely the one with all probes free. ☺

The above illustrates the way we view ESSPs and similar objects as being given by their visible part, with their status as genuine ESSPs, a matter to be inferred, eg. by search. Indirectly this also explains why in definitions 9.4 and 9.9, Ψ and Φ are functional and not relational as in definition 5.5. The point is that in sections 7 and 8 all SFs were generated by local choice of distinguished port edges, and thus that any SF of some object was "assembled from its parts", leading naturally to a relational definition. In this section, probes act non-locally and invisibly, and the criteria for acyclicity of SFs become more complex. Insisting that definitions 9.4 and 9.9 were relational, i.e. that Ψ and/or Φ were at least onto, and possibly one-many, would destroy many useful pushouts, and would impair the theory of rewriting. Similar remarks emerge when we consider sequentialisation. Of course one could have made definition 5.5 functional, to no great effect in sections 7 and 8.

Definition 9.13 A subESSN (N, E) is an E-polysimple net (EPSN), iff for each $e \in E$, every switching is E-special, and E is exactly the set of partial algebras with common net N for which this is true. If the number of E-components in question is one, then we have an E-simple net (ESN). If an E-pattern (N, E') has a visible part N which is the common pattern of some EPSN or ESN (N, E), then we say that N is an EPSN or ESN.

The two definitions 9.11 and 9.13 are the crucial ones with regard to the previous sections. More definitions will follow below, but for now we look at some examples.

Consider the visible part P in Fig. 34.(a). If we make the probes of both S and T free, then we conclude that P is an ESSP since there is at least one E-structure with acyclic ESF for P. We illustrate the two switchings for this particular E-structure in Fig. 37.

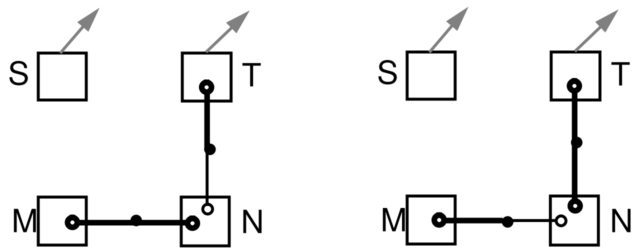


Fig. 37.

In fact P is also an ESN. The E-structures that witness this have already been illustrated in Fig. 34.(b) – (e). In Fig. 38, we show the two switchings for one of those E-structures.

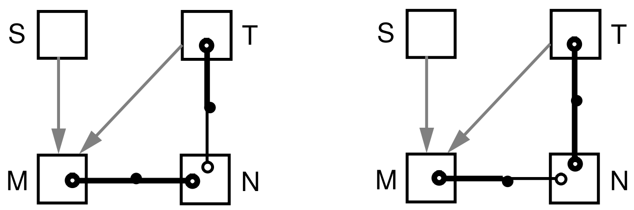


Fig. 38.

In Fig. 39 we show that the collection of switchings of an SSN alone, gives little clue as to the number or distribution of probe assignments that could extend the SSN to an ESSN. This is of interest when we come to consider rewriting, where the redex occurs in an unknown context.

By now the reader will be thinking that technically, the E-version of events looks much like what we had before. It seems only that where we had connectedness we now have E-connectedness, and where we had SFs we now have ESFs. The relationship between ESFs and E-components, is just like that between SFs and components. This allows us to relativise most of the arguments in section 7 and 8 to the new situation. In fact the following rather vague statement is the main objective of this subsection, and one that will be used repeatedly later.

Theorem 9.14 (Relativisation Metatheorem) All decision results which rely solely on the relationship between SFs and connectedness, may be transliterated mutatis mutandis to refer to ESFs and E-connectedness.

Typically we have in mind the vital early results in sections 7 and 8 on acyclicity, the $<$ partial order on NS clusters, and the existence of scopes. Since almost the whole of these two sections rests on these foundations, we quickly gain a large number of useful results.

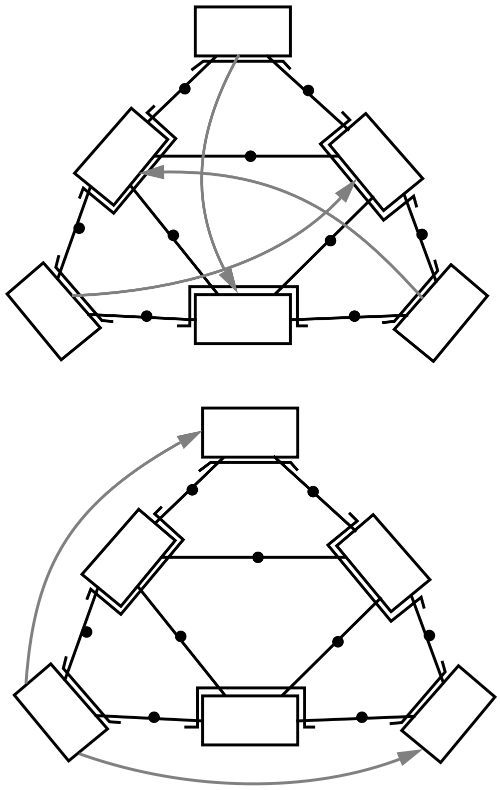


Fig. 39.

Remark 9.15 We can regard agents with a single probe, as the natural combinatorial analogues of subordinate agents with an empty cluster in LN theory. Our use of agents with several probes, generalises this somewhat. But one could go further. In view of the potential for relativisation, one could just as easily admit E-clusters of probes into the theory without changing much at all. We refrain from pursuing this variant of the theory here. Having an agent possess several of our present probes enables us to connect their owner to ESF-disjoint parts of a net, which is a potentially useful notion. In the alternative case, in an EPSN, all probes of an E-cluster would have to have victims that were ESF-strongly connected. Since they are all supposedly invisible, having several of them doing a job that a single probe could do seems unproductive. Nevertheless we point out the possibility.

9.3 Static Properties

By relativisation, we immediately obtain the most important property of ESSPs.

Theorem 9.16 A principal ESSP (P, E) is deadlock free.

This is to be read as saying that P is deadlock free in the conventional sense, and follows because each partial algebra in E is E-deadlock free and each SF is a subforest of every

ESF. However the direct relativisation of theorem 7.8 fails because the different E-structures in E may make certain probes free in some partial algebras, and not free in others, whereupon the cardinality of the different ESFs may differ. However we have the following.

Let (P, E) be an ESSP, and e a partial algebra of E. We write $|e|_E$ for the number of E-components of e, and $\langle e\rangle_E$ for the number of components of any ESF of e. We write $[e]_E$ for the number of free probes in e, and S_P for the total number of probes in e, which is just the total number of occurrences of probes in agents of P and is independent of e.

Theorem 9.17 Let (P, E) be an ESSP. Then there is a constant k such that for all $e \in E$

(*) $\quad \langle e\rangle_E - [e]_E = k = \langle e\rangle - S_P = \langle P\rangle - S_P$

Proof. Take the free E-structure for P. Considering all switchings and theorem 7.8, gives us a k for this E-structure, and the right two equalities in (*). Any other E-structure in E may be constructed by choosing victims for a sequence of probes in such a way that at each stage, the ESF remains acyclic for every switching. Every time we choose such a fresh victim, $[e]_E$ and $\langle e\rangle_E$ both decrease by 1, preserving (*). ☺

Theorem 9.18 Let (P, E) be an ESSP. Let $\langle\langle P\rangle\rangle_E$ be the minimum value reached by $\langle e\rangle_E$, and $[[P]]_E$ be the minimum value reached by $[e]_E$, as we allow e to vary over E. Then

$$k = \langle\langle P\rangle\rangle_E \quad \text{and} \quad [[P]]_E = 0 \ \text{ or,}$$
$$k = 1 - [[P]]_E \quad \text{and} \quad \langle\langle P\rangle\rangle_E = 1$$

Proof. As the process in theorem 9.17 continues towards the minimum values, eventually we run out of either free probes or disjoint ESF-connected components. ☺

Corollary 9.19 For an ESSP (P, E)

$$\langle\langle P\rangle\rangle_E - [[P]]_E = \langle P\rangle - S_P$$

Of course for an ESSN (N, E), for each $e \in E$, $[e]_E$ is zero, so that by corollary 9.19, $\langle e\rangle_E$ is independent of e, and we can thus write $\langle N\rangle_E$ (which of course equals $\langle\langle N\rangle\rangle_E$), unambiguously.

For ESSNs, we have a straightforward relativisation of theorem 7.8.

Theorem 9.20 In an ESSN (N, E), the cardinality of each ESF is constant (regardless of algebra $e \in E$ or of switching of e).

The remaining results in this subsection pertain to EPSNs and their subpatterns. Armed with the insights of subsections 8.1 and 8.2, the following are quoted without proof.

Theorem 9.21 In an EPSN there is a reflexive transitive partial order $<_E$ on E-scopes defined by

$$\text{E-scope}(p) <_E \text{E-scope}(q) \Leftrightarrow p \in \text{E-scope}(q)$$

The above is in addition to the partial orders $<_E$, $\ll_E$ on NS clusters (the former of which exists also in ESSPs).

Theorem 9.22 A subESSN (N, E) is an EPSN iff every NS cluster has an E-scope (in every algebra $e \in E$).

Definition 9.23 In a subESSP, given a particular partial algebra and switching, an E-descending path is an ESF-path which starts at a port node of an NS cluster q, and for every NS cluster p, which both meets the ESF-path in a port edge, and also does not have an E-scope, $p <_E q$.

Definition 9.24 An EsubESSP (P, E) of some ESSP (Q, F) is an E-polysimple pattern (EPSP) iff each port connection contains at most one port edge from an NS cluster (aside from NS clusters like Fig. 26), and for each NS cluster q, either q has an E-scope, or every port node a of q can ESF-access a free port node, a free probe, or a victim of a probe owned by an agent in $Q - P$, via an E-descending path. Also E is the largest set of partial algebras for which this is true.

Theorem 9.25 Any E-subpattern of an EPSN is an EPSP.

Remark 9.26 Note that these are less useful definitions than their counterparts in section 8 because they rely implicitly on the embedding of the putative EPSP (P, E) in a larger structure, to pin down which of the agents of P might be victims of probes from outside.

9.4 Sequentialisation

In this subsection, we will examine the sequentialisation theory for open E-nets of various kinds. We will concentrate on open E-monosimple nets (OEMSNs), after which the theory of OEPSNs and OESSNs will emerge rather easily.

Definition 9.27 An E-pattern is an open E-monosimple net (OEMSN) if it can be constructed using the following sequent rules.

EMPTY: An empty E-pattern is an OEMSN.

PORT: An isolated port node (with the obvious trivial algebra) is (the visible part of) an OEMSN.

GRAFT1S: Let A be an m-subordinate agent having l free ports grouped into n clusters. Let $ON_1 \ldots ON_{n+m}$ be non-empty OEMSNs. Let f map the n+m ON_i 1–1 onto the n+m clusters and probes of A so that if p is a cluster of A the number of free ports of the OEMSN $f^{-1}(p)$ is at least as large as the number of free ports of p itself, and if p is a probe of A, there is at least one agent in $f^{-1}(p)$. Create an E-net ON_0 by doing the following.

(i) If p is a cluster, make port connections between all free ports in p and (some) free ports in $f^{-1}(p)$, (allowing for the degenerate case where some of the OEMSNs might be isolated port nodes).

(ii) If p is a probe, choose a victim for p from among the agents in $f^{-1}(p)$.

Keep f and the port connections in (i) fixed. Allow the choice of victim in (ii) to vary over all agents in $f^{-1}(p)$, and for all p, allow the E-structures in $f^{-1}(p)$ to vary over all those allowed by the algebras of $f^{-1}(p)$. This gives a set of algebras with an underlying common net for ON_0. Then ON_0 is an OEMSN.

Let us examine an example. Fig. 34 will again be our focus. There are a number of ways that (a) can be seen to be an OEMSN. We will discuss this in terms of linear schedules for constructing the various algebras of Fig. 34. We will ignore the specific introduction of the port nodes in Fig. 34 as they can be inferred from the rest. Suppose then we introduce the agents in the order MTNS. This sequence is able to form the algebras (b), (c), (d) of Fig. 34, depending on the choice of victim for S. The order MTSN can form (b) and (c). Finally the order MSTN generates (c) and (e), depending on the choice of victim for T. Readers can check that these are the only sequential construction schedules that add the owner of an E-scope only after the proper E-scope is complete, the essential criterion enforced by GRAFT1S.

We see that (a) is the visible part of the three OEMSNs ((a), {(b), (c), (d)}), ((a), {(b), (c)}), ((a), {(c), (e)}). There is no especially simple relationship between the OEMSNs arising from different schedules. No two of the ones we have examined are even disjoint, and none of them correspond to the EMSN ((a), {(b), (c), (d), (e)}), because the sequential construction regime prevents agents introduced later, from being victims of agents introduced earlier. Nevertheless one can make the following fairly straightforward statements.

Theorem 9.28 Every OEMSN is a subESSP such that:

(1) The cardinality of each ESF is one.

(2) Each NS cluster has an E-scope.

(3) Each pair of free ports is ESF-strongly connected.

(4) Each free port is in a singleton cluster.

(5) Every algebra is total.

Proof. Relativisation of 8.26 gives (1) – (4); (5) is obvious. ☺

For the converse we need to refine the partial order $<$ on NS clusters to include the probes.

Definition 9.29 Let e be a partial algebra of an ESSP. Consider probes to be oriented from their victim to their owner, rather as NS cluster port edges are in definition 7.5. Let p be either a probe or an NS cluster port edge, and similarly q. We write $p <_E q$ iff there is an ESF-path in e, which starts with p, ends with q, and such that both p and q (but not necessarily intermediate port edges or probes) are traversed in the correct direction.

Lemma 9.30 Let e be a partial algebra of an ESSP. Then $<_E$ on NS clusters and probes is an irreflexive transitive partial order.

Theorem 9.31 Let (P, E') be an ESSP such that there is an $e \in E'$ for which (1) – (5) of theorem 9.28 hold. Then there is an OEMSN (P, E) such that $e \in E$.

Proof. Consider decomposing e as in theorem 8.25 except with proper regard for the extended $<_E$ relation of definition 9.29. The inclusion of probes in the new partial order ensures that when removing $<_E$-maximal probes, no victim is removed by an un-GRAFT1S before any probe which invades it. Now consider the sequential construction of e corresponding to this decomposition. The various steps can be reinterpreted as constructions of OEMSNs by allowing the E-structures to vary in permitted ways, away from those dictated by e, at every stage. The result is an OEMSN (P, E) with e certainly in E. ☺

We have thus seen that under the conditions of this section, the naturally arising concepts of OEMSN and EMSN diverge even more than their counterparts did in previous sections. Still, to every algebra of an EMSN there is an OEMSN which constructs it and possibly others, sequentially. We are now in a position to dispose of the main facts about OEPSNs without much comment.

Definition 9.32 An E-pattern is an open E-polysimple net (OEPSN) if it can be constructed using the following rules.

EMPTY: An empty E-pattern is an OEPSN.

MIX: Let $ON_1 = (P_1, E_1) \ldots ON_m = (P_m, E_m)$ be non-empty disjoint OEMSNs. Then $ON_0 = (P_0, E_0)$ is an OEPSN where P_0 is the union of the P_i, and E_0 is the set of all $e_1 \cup e_2 \cup \ldots \cup e_m$ where the $e_i \in E_i$ are selected in all possible ways. (If $ON_1 \ldots ON_m$ are not disjoint, alpha-relabel them).

If in theorem 9.28, we change (1) to say that the number of ESF components equals the number of E-components, then theorems 9.28 and 9.31 carry over to OEPSNs.

Predictably, for OEPSNs there are a number of approaches. One could permit the MIX rule above to co-exist with EMPTY and GRAFT1S. Alternatively one could define a GRAFT0S rule, the obvious analogue of GRAFT0, by allowing f to be many-one, whereupon all but a few uses of MIX could be avoided. The main result which the reader can easily show, is the next one.

Theorem 9.33 Every OEPSN is a subESSP. If (P, E') is an ESSP and $e \in E'$, then there is an OEPSN (P, E) with $e \in E$.

9.5 Rewriting

We can now consider the rewriting of our various E-prefixed objects using the double pushout construction. Fig. 40 shows how the new data gets incorporated into the double pushout diagram.

$$\begin{tikzcd}
(L, E_L, \Lambda) \arrow[d, "{(g, \Psi_g, \gamma)}"'] & (K, E_K, 1) \arrow[l, "{(l, \Psi_l, \lambda)}"'] \arrow[r, "{(r, \Psi_r, \rho)}"] \arrow[d, "{(d, \Psi_d\ \delta)}"] & (R, E_R, \mathrm{P}) \arrow[d, "{(h, \Psi_h, \eta)}"] \\
(G, E_G, \Gamma) & (D, E_D, \Delta) \arrow[l, "{(l^*, \Psi_{l^*}, \lambda^*)}"] \arrow[r, "{(r^*, \Psi_{r^*}, \rho^*)}"'] & (H, E_H, \Omega)
\end{tikzcd}$$

Fig. 40.

The easiest case to consider is the rewriting of ESSPs, i.e. when Fig. 40 is a double pushout in $\mathcal{EP}$. By theorem 9.12, we know that a pattern P is an ESSP iff it is an SSP. This means that we can take over most of the results on SSP rewriting from subsection 7.3. We point out how this works.

Since we treat free probes differently from free ports, theorem 7.17 does not carry over directly. Nevertheless we have the following adaptation of theorem 7.17.

Theorem 9.34 Let $G \leftarrow D \rightarrow H$ be a rewrite of an ESSP by a rule $L \leftarrow K \rightarrow R$, such that H is also an ESSP. Let s be the number of short circuits of the rule. Then

(1) $\langle\langle D\rangle\rangle_E - [[D]]_E = \langle\langle r^*(D)\rangle\rangle_E - [[r^*(D)]]_E + s$

(2) $\langle\langle H\rangle\rangle_E - [[H]]_E - (\langle\langle G\rangle\rangle_E - [[G]]_E) = \langle\langle R\rangle\rangle_E - [[R]]_E - (\langle\langle L\rangle\rangle_E - [[L]]_E)$

Proof. This follows by using theorem 9.19 on equations 7.17.(1) and 7.17.(2), and noting that $S_{r^*(D)} = S_D$ and $S_H - S_G = S_R - S_L$. ☺

Theorems 7.18 onwards carry over as follows.

Theorem 9.35 Suppose G is an ESSP, and it is rewritten using a rule $L \leftarrow K \rightarrow R$ to produce H. Then H is an ESSP iff

(1) R is an ESSP,

(2) For no nonempty even length sequence of distinct nodes of K, $[a_0, b_0, a_1, b_1, \ldots, a_n, b_n]$ is it true that

a_0 and b_0 are SF-weakly connected in D,
b_0 and a_1 are SF-weakly connected in R,
a_1 and b_1 are SF-weakly connected in D,
b_1 and a_2 are SF-weakly connected in R,
… … … … … … … … … … … …
a_n and b_n are SF-weakly connected in D,
b_n and a_0 are SF-weakly connected in R.

Note that we do not have to worry about E-structures or ESF connectivity because for each ESSP, we have at least the free E-structure, which trivialises all such issues.

Theorem 9.36 Suppose G is an ESSP and it is rewritten using a rule $L \leftarrow K \rightarrow R$. Suppose

(1) R is an ESSP,

(2) For each switching t of R there is a switching s of L such that for all distinct pairs a, b in K,

a, b SF-connected by t in $R \Rightarrow a$, b SF-connected by s in L.

Then the pattern H produced is an ESSP.

Note that theorem 9.36 implicitly restricts itself to the free E-structures of G and H (and L and R) which is sufficient. There is a marginally stronger form which allows the switching s of L, to be a switching of a non-free partial algebra e_L of L, which witnesses the arrow $L \rightarrow G$. In this case, the RHS of the implication in (2) would read “a, b ESF-connected in e_L by s in L”. Such a strengthening would only have force if L were disconnected though.

Corollary 9.37 Rules $L \leftarrow K \rightarrow R$ such that for all distinct a, b in K,

a, b are in the same cluster in $L \Leftrightarrow a$, b are in the same cluster in R,

satisfy the hypotheses of theorem 9.35 and so preserve the ESSP property.

This is as much as we wish to say about the rewriting of ESSPs. In particular theorem 7.22 does not relativise immediately, because in dealing with SSNs, slightly different criteria apply regarding probes, which being non-free, contribute to E-connectedness. It is to ESSNs and especially to EPSNs that we now turn.

One thing that is a potential nuisance in the rewriting of nets, is the possiblity that probes may cross the boundary between the redex and its environment, i.e. a probe owned by an agent in L, may have a victim in D, and vice-versa. We call a probe owned by L and with victim in D outgoing, while one owned by D with victim in L will be called incoming.

Another potential nuisance to consider is whether the stronger invariants that apply to net rewriting cause problems in demanding that the squares in Fig. 41 are genuine pushouts. But here we note that the rewriting mechanism is still restricted to double pushouts in $\mathcal{EP}$, so that the free E-structures for L, K, R, D witness arrows between them, and also witness arrows into G and H whenever the latter are at least ESSPs. So we indeed only need to check that at least one algebra for H satisfies whatever stronger conditions we require, in order to know that we have a genuine rewrite. The drawback of this of course, is that there are no useful analogues of theorems 7.18 and 8.30 which do not just restate that H is an ESSN or EPSN. This is because the assignment of victims is an essentially global process. However the conservation laws adapt as below.

Theorem 9.38 Let $G \leftarrow D \rightarrow H$ be a rewrite of an ESSN by a rule $L \leftarrow K \rightarrow R$, so that H is an ESSN. Then

$$\langle H\rangle_E - \langle G\rangle_E = \langle R\rangle - \langle L\rangle - (S_R - S_L)$$

Proof. Using theorem 9.19, this is just another way of writing theorem 9.34.(2). ☺

Corollary 9.39 If in theorem 9.38, G and H are both EPSNs then

$$|H|_E - |G|_E = \langle H\rangle_E - \langle G\rangle_E = \langle R\rangle - \langle L\rangle - (S_R - S_L)$$

The safety theorems become the following.

Theorem 9.40 Suppose G is an ESSN, and it is rewritten using a rule $L \leftarrow K \rightarrow R$. Suppose for every partial algebra e of L, there is a partial algebra f of R with all switchings ESF-acyclic, such that there exist maps

θ : Agent nodes of $L \rightarrow$ Agent nodes of R
ζ : Free probes of $f \rightarrow$ Free probes of e

(the latter of which induces a map

ξ : Owners of free probes of $f \rightarrow$ Owners of free probes of e)

with ζ injective, and such that for every switching t of f there is a switching s of e such that

(1) For all a, b in K,

a, b ESF-connected by t in $f \Rightarrow a$, b ESF-connected by s in e,

(2) For all X, Y in L,

$$\theta(X), \theta(Y) \text{ ESF-connected by } t \text{ in } f \Rightarrow X, Y \text{ ESF-connected by } s \text{ in } e,$$

(3) For all X, Y in the domain of ξ

$$X, Y \text{ ESF-connected by } t \text{ in } f \Rightarrow \xi(X), \xi(Y) \text{ ESF-connected by } s \text{ in } e,$$

(4) For all X in L and Y in the domain of ξ

$$\theta(X), Y \text{ ESF-connected by } t \text{ in } f \Rightarrow X, \xi(Y) \text{ ESF-connected by } s \text{ in } e,$$

(5) For all X in L and a in K,

$$\theta(X), a \text{ ESF-connected by } t \text{ in } f \Rightarrow X, a \text{ ESF-connected by } s \text{ in } e,$$

(6) For all Y in the domain of ξ and a in K,

$$Y, a \text{ ESF-connected by } t \text{ in } f \Rightarrow \xi(Y), a \text{ ESF-connected by } s \text{ in } e,$$

Then H is an ESSN.

Proof. The basic idea is as in the proof of theorem 7.19, except that things are made much more elaborate by the need to cater for the probes. Suppose therefore H was not an ESSN. Then there would be no E-structure for it for which all switchings yielded an acyclic ESF. We will construct an algebra for H for which the assumption of a cyclic ESF in some switching yields a contradiction.

We know that G is an ESSN, so has an algebra e' for which all switchings are ESF-acyclic. This induces a partial algebra e for L, in which a probe is free iff its victim in G lies outside $g(L)$. Removing the image of L leaves D and induces a partial algebra c for D. In general c has some probes free (which used to have victims in L), and has fewer victim agents than e' (since probes owned by L have been removed). When we glue in R to make H, we must find victims for the free probes of both D and R. Let f be a partial algebra for R determined by e according to the hypothesis, (this fixes θ, ζ, and ξ). Let x be a free probe of D, and suppose its victim in G was $g(X)$ (for a suitable agent X of L). In H, let the victim of x be $h(\theta(X))$ (where $\theta(X)$ is in R). Similarly, if y is a free probe of R, let its victim in H be the agent of D which is the victim of $\zeta(y)$. This yields an algebra f' for H, and we must show that for all switchings, the ESF is acyclic.

Suppose then, there was a switching t' of f' with cyclic ESF. The ESF-cycle could not be entirely in $r^*(D)$ or in $h(R)$ because of the properties of c and f. (Assume for convenience r^* and h injective, by emulating short circuits using fat short circuits). Therefore the cycle must wander between the two. We show that each piece in $h(R)$ corresponds to a piece in $g(L)$, and yields an ESF-cycle in G.

Let t be the switching of f, and u the switching of c, induced by t'. Let s be the switching of e corresponding to t by hypothesis. Consider a portion of the ESF-cycle in $h(f)$. It can be connected to $r^*(c)$ in a number of ways. It may enter $h(f)$ through a node of K, through an incoming probe, or through an outgoing probe. It may leave $h(f)$ in these three ways too. Disregarding orientation, this gives six cases, covered by the six clauses in the hypothesis.

(1) Entry and exit through K. This is as in theorem 7.19.

(2) Entry and exit through incoming probes. The victims in $h(R)$ of the two distinct probes in question will be $h(\theta(X))$, $h(\theta(Y))$ say, and so will correspond to victims X, Y of $g(L)$ also ESF-connected by s in e. Hence their owners in D are ESF-connected in G.

(3) Entry and exit through outgoing probes. The owners in $h(R)$ of the two distinct probes in question will be X, Y say, and so $\xi(X)$, $\xi(Y)$ will be ESF-connected by s in e. The victims in D will thus be ESF-connected in D. The injectivity of ζ prevents the collapse of ESF-cycles in pathological cases where both probes of $h(R)$ have the same victim and $\xi(X) = \xi(Y)$.

(4) Entry via incoming and exit via outgoing probes, (or vice versa). The victim in $h(R)$ of the incoming probe will be $h(\theta(X))$ say, and the owner of the outgoing probe will be Y say. Thus since X and $\theta(Y)$ are ESF-connected by s in e, the owner of the incoming probe and the victim of the outgoing probe are ESF-connected in G.

The other two cases are similar. We are done. ☺

Theorem 9.40 is very general, but is rather intemperate in its use of inversions of quantifiers, and thus specific instances of rules may be hard to check for safety. Usually though we are interested in rules with simple LHSs, where all the agents are at least SF-weakly connected. In such cases, simpler results suffice.

Theorem 9.41 Suppose in a rule $L \leftarrow K \rightarrow R$, all pairs of agents of L are SF-weakly connected. Suppose R has at least one agent. Suppose there is a partial algebra f of R for which $[f]_E \leq S_L$, and for which the conditions in the hypothesis of theorem 9.40 hold. Then if an ESSN G is rewritten using the rule, the object H produced, is an ESSN.

Proof. If all agents of L are pairwise SF-weakly connected, no probe owned by L can have a victim in L. Referring to theorem 9.40, there is thus only the free partial algebra e of L to consider, and only one partial algebra f of R to find. If f has no more free probes than e, victims in D can be found for all of them. Similarly, R has at least one agent which can act as victim for incoming probes. Otherwise the proof is as for theorem 9.40. ☺

Theorem 9.42 Suppose in a rule $L \leftarrow K \rightarrow R$, all agents of L are SF-strongly connected. Suppose R has at least one agent. Suppose there is a partial algebra f of R for which $[f]_E \leq S_L$, and suppose each switching of f is ESF-acyclic. Suppose also that for each switching t of f there is a switching s of (the free partial algebra e of) L such that for all a, b in K

$$a, b \text{ ESF-connected by } t \text{ in } f \Rightarrow a, b \text{ ESF-connected by } s \text{ in } L.$$

Then if an ESSN G is rewritten using the rule, the object H produced, is an ESSN.

Proof. If the agents of L are SF-strongly connected, L can be regarded as (perhaps only part of) the centre of a "star" of SF-components of G to which SF-components are joined by probes outgoing from, and incoming to, L. Under these circumstances, $[f]_E \leq S_L$ ensures that victims can be found for free probes of f. Incoming probes have a victim in R. The rest of the argument reduces to a relativised form of theorem 7.19. ☺

The weaker version of theorem 9.42, dealing as does theorem 7.20 with nodes of K in the same cluster in L and R, we leave to the reader.

These results adapt in the expected way when we strengthen the invariants to focus on EPSNs. We end up with a blend of the preceding and of theorem 8.31.

Theorem 9.43 Suppose G is an EPSN, and it is rewritten using a rule $L \leftarrow K \rightarrow R$. Suppose for every partial algebra e of L, there is a partial algebra f for R with all switchings ESF-acyclic, such that there exist maps

θ : Agent nodes of $L \rightarrow$ Agent nodes of R
ζ : Free probes of $f \rightarrow$ Free probes of e

(the latter of which induces a map

ξ : Owners of free probes of $f \rightarrow$ Owners of free probes of e)

with ζ injective and surjective, and such that for every NS cluster p in R which does not have an E-scope in f (for at least some switching of f), for every switching of f, every port node of p can ESF-access either a free port node of R, or an agent in the domain of ξ, or an agent in the image of θ. Also for every switching t of f there is a switching s of e such that

(1) For all a, b in K,

- (i) a, b E-connected in $R \Rightarrow a$, b E-connected in L,
- (ii) a, b ESF-connected by t in $f \Leftrightarrow a$, b ESF-connected by s in e,

(2) For all X, Y in L,

- (i) $\theta(X)$, $\theta(Y)$ E-connected in $R \Rightarrow X$, Y E-connected in L,
- (ii) $\theta(X)$, $\theta(Y)$ ESF-connected by t in $f \Leftrightarrow X$, Y ESF-connected by s in e,

(3) For all X, Y in the domain of ξ

- (i) X, Y E-connected in $R \Rightarrow \xi(X)$, $\xi(Y)$ E-connected in L,
- (ii) X, Y ESF-connected by t in $f \Leftrightarrow \xi(X)$, $\xi(Y)$ ESF-connected by s in e,

(4) For all X in L and Y in the domain of ξ

- (i) $\theta(X)$, Y E-connected in $R \Rightarrow X$, $\xi(Y)$ E-connected in L,
- (i) $\theta(X)$, Y ESF-connected by t in $f \Leftrightarrow X$, $\xi(Y)$ ESF-connected by s in e,

(5) For all X in L and a in K,

- (i) $\theta(X)$, a E-connected in $R \Rightarrow X$, a E-connected in L,
- (ii) $\theta(X)$, a ESF-connected by t in $f \Leftrightarrow X$, a ESF-connected by s in e,

(6) For all Y in the domain of ξ and a in K,

- (i) Y, a E-connected in $R \Rightarrow \xi(Y)$, a E-connected in L,
- (ii) Y, a ESF-connected by t in $f \Leftrightarrow \xi(Y)$, a ESF-connected by s in e,

Then H is an EPSN.

Proof. We sketch the main points, leaving some of the fine detail to the reader. We know that G is an EPSN, whence it is an ESSN, and has an algebra e' in which every NS cluster has an E-scope. We must ensure the same for H. From theorem 9.40, H is an ESSN already, so we need to display an algebra f' for H in which each NS cluster has an E-scope. As usual e' induces partial algebras e for L and c for D. Let f for R correspond to e for L by hypothesis. Assign victims for the free probes of f and c when constructing H, as per theorem 9.40, giving an algebra f' for H. Choose a switching of f', amounting to switchings t of f and u of c. Let switching s of e correspond to t by hypothesis.

We claim f' is as required. As usual, we only need to worry about NS clusters in f and c which do not have E-scopes in f or c respectively. Let q have no E-scope in c. Since q had an E-scope in e', in c every port node of q can ESF-access either a port node of K, or the owner of a probe with victim assigned by θ in R, or the victim of a probe of R, (use the surjectivity of ζ in the latter case). Using the implications in (1).(ii) to (6).(ii) from right to left, allows us to construct on ESF-path in f' by t and u between any two port nodes of q, whence q has an E-scope in f'. Now suppose that q in f has no E-scope. This means that there is a pair of "objects" A and B in R, which are E-connected and ESF-disconnected by t in f along an E-path through q. By an "object" we mean a port node of K, or an agent in the image of θ, or an agent in the domain of ξ. Rather as in theorem 8.31, this pair of objects corresponds to a pair of objects A' and B' in L, E-connected and ESF-disconnected by s in e. Thus an E-path between A' and B' must go through two consecutive port edges of an NS cluster p in L. Since p has an E-scope in e', A' and B' must be ESF-connected by s and u in e', from which, using the conditions (1).(ii) to (6).(ii) from right to left as many times as necessary, we can deduce that A and B are ESF-connected by t and u in f'. But this means that q has an E-scope in f, since t and u were arbitrary. We are done. ☺

Corollary 9.44 If in theorem 9.43 we replace all occurrences of "$\Rightarrow$" by "$\Leftrightarrow$" (in (1).(i) – (6).(i)), and assume that each E-component of a partial algebra f of R of interest, is either connected to at least one node of K, or has a free probe, or intersects the image of θ, then the number of E-components of G and H is the same. Therefore if G is E-monosimple, so is H.

Again there are simplified versions of these results which make stronger demands on the LHS or RHS of the rule. There are obvious analogues of theorem 9.41 which we do not quote, assuring preservation of the EPSP or EMSP property, when the agents of the LHS are ESF-weakly connected. More worthy of explicit mention are the following.

Theorem 9.45 Suppose in a rule $L \leftarrow K \rightarrow R$, all agents of L are SF-strongly connected, and suppose R has at least one agent, and posesses a partial algebra f such that all agents of R are ESF-strongly connected in f, and also $[f]_E = S_L$. Suppose that for each switching t of f there is a switching s of (the free partial algebra e of) L, such that for all a, b in K,

$$a, b \text{ ESF-connected by } t \text{ in } f \Leftrightarrow a, b \text{ ESF-connected by } s \text{ in } L.$$

Then if an EPSN G is rewritten using the rule, the object H produced is an EPSN. Furthermore if G is an EMSN, then so is H.

Proof. As in theorem 9.42, there is only one partial algebra of L to consider, and L acts like the centre of a star of SF-components. Since R is ESF-strongly connected in f, R also acts like the centre of the star. Since the number of incoming and outgoing probes in R and L are equal, the number of ESF components stays fixed. The ESF-strong connectedness of L and R ensures that all NS clusters have E-scopes in H. ☺

Note that in theorem 9.45, the equivalent of condition 8.32.(2).(i) i.e. nodes of K E-connected in R iff they are E-connected in L, comes for free since all agents in both L and R are ESF-strongly connected together. There is a weaker version of theorem 9.45 dealing with nodes of K in the same cluster in L and R which we leave to the reader. Likewise, there are stronger versions which allow for more of the possibilities offered by the full ramifications of the phrase "R is an EPSP without any E-components not E-connected to D either through K or a probe" but they are not substantially more succinct than theorem 9.43 so we do not discuss them here. In the appendix we will see how to express the proof nets of LLM including the constants as EMSNs.

10 DISCUSSION AND CONCLUSIONS

It is time to summarise what we have achieved in this paper. We have explored INs from a primarily graph theoretic and algebraic perspective. In doing so, we freed the exposition from any unnecessary dependence upon techniques from logic. The result is arguably a smoother theory, and this shows up well when we develop the sequentialisation properties of INs, where particularly in the case of the extended theory of section 9, the details of the sequential construction of an open net can interfere with its global properties. For an alternative approach to some aspects of the extended theory in the specific context of linear logic and distributive categories, and with a greater emphasis on sequentialisation, see the recent paper of Blute et al. (1993).

We explored a number of classes of safe rewriting for INs, all ultimately founded on theorem 7.10, which showed that a suitable relationship between principality assignments and clustering assignments is all that is needed to reduce deadlock avoidance to acyclicity of (E)SFs, and thence to criteria about rewrite rules that are determinable from the rule structure. At the end of the day, many classes of interest emerged, requiring a variety of hypotheses for their safety theorems. No doubt there are others that could be considered, and ultimately their taxonomy begins to tax our powers to invent fresh acronyms as much as anything else. The ones we investigated are the ones that display the traits most apparent in proof nets for LLM, treated in the appendix.

The inclusions that hold among the normal classes are illutrated in Fig. 41, while those for the extended classes appear in Fig. 42. These figures could be combined into one rather unwieldy diagram in two ways, depending on whether one wishes to view the normal classes as merely proper subclasses of the extended classes, or whether one prefers to view them as arising as a result of forgetting the details of E-structure in the extended case. Of course these two viewpoints are mutually adjoint.

A number of further points deserve some discussion, particularly garbage collection, synchronisation and type systems.

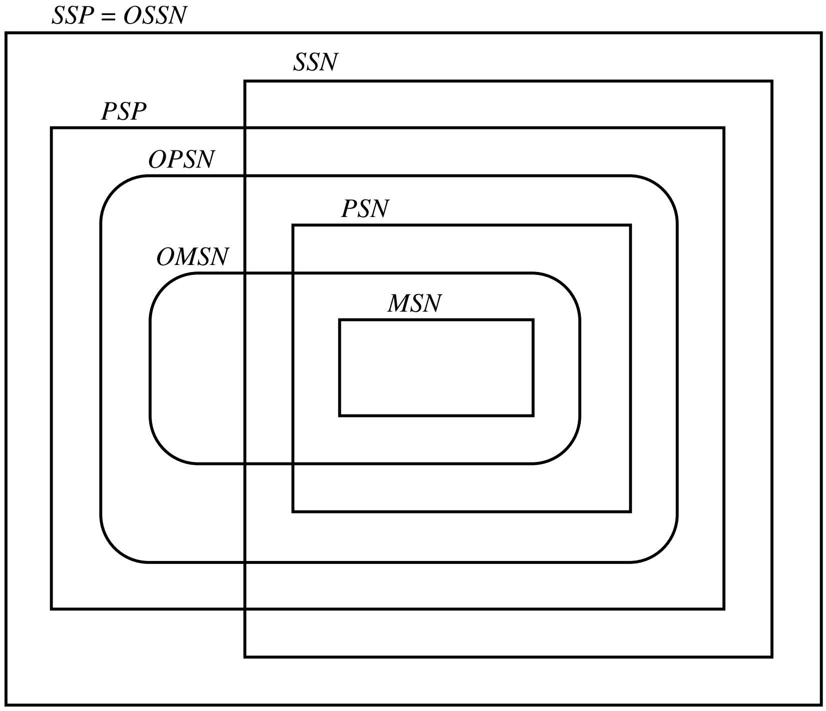


Fig. 41.

IN systems are rather well behaved regarding garbage collection because of the port invariant. One always knows right there during the rewrite that one should discard both the LHS agents and replace them with the RHS, so an autonomous process which goes round looking for occupied but casually abandoned parts of store is unnecessary. This isn't to say that garbage management is unnecessary. If the RHS is a different size than the LHS then one cannot just overwrite the one with the other, and free list handling is needed.

The easily achievable Church-Rosser properties of IN systems tempt one to imagine IN systems being rewritten in parallel. However the synchronisation issues raised demand a little care. Of course much depends on how the INs are represented in a real store, but some general remarks may be made. Assuming a fairly literal representation, there are few problems if there are no short circuits. Even adjacent and concurrent rewrites can freely access the ports which connect them since neither requires to access data relevant to the other. Neverthless we still need to discover cuts, and this requires synchronisation. When there are short circuits things get worse since port nodes need to be destroyed or overwritten by indirections, and this requires mutual exclusion between adjacent concurrent rewrites at a coarser level of granularity than cut detection, since now the boundary between two rewrites in the net need not be independent of time. Ob-

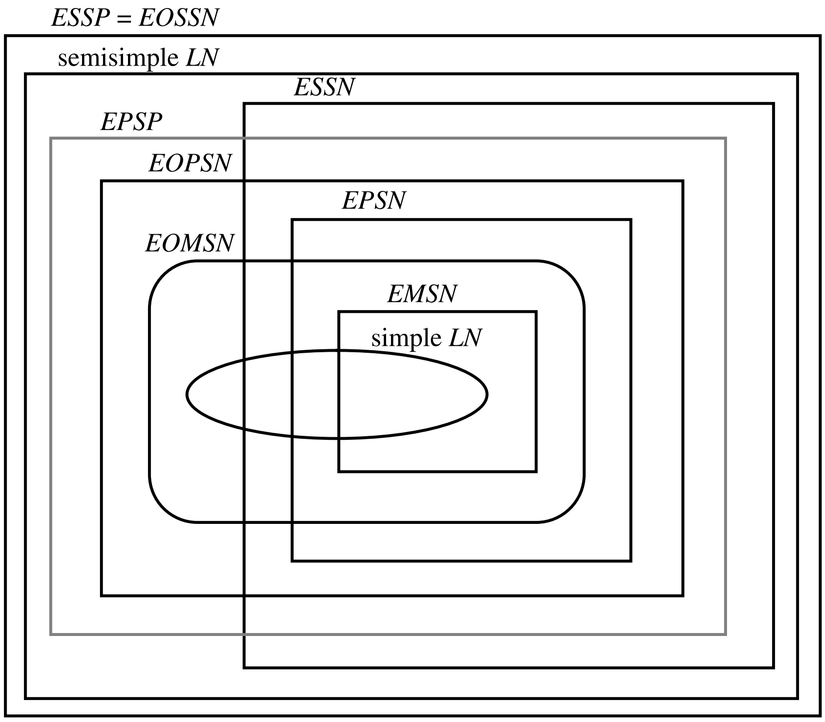


Fig. 42.

viously, things get even worse when genuinely overlapping redexes are permitted as for axiom strings in LLM below.

We have pointed out the easy conditions under which Church-Rosser properties arise for IN systems, as spotted by Lafont, but our general formulation of IN theory prompts the question whether CR properties can be proved for IN systems in which the "one principal port" restriction does not hold. Such rule systems are clearly suited to modelling non-deterministic computations (as are rule systems in which the "at most one rule per cut" restriction fails). Nevertheless classes of such rules for which CR holds, can be identified, just as there are term rewrite systems for which the completion problem has a solution. LLM as described in the appendix forms a prime example. The more general theory of CR properties for such systems will be explored elsewhere.

Similar remarks apply to the question of termination, where again, only a fairly trivial criterion has been given. Here there are undoubtedly many interesting results to be described, building on the extensive termination theory that already exists for TRSs. Again, this is material for another paper.

Finally type systems. We have purposely avoided any mention of the topic till now since the objective of this paper is to thoroughly explore principality, deadlock avoidance and simplicity. However, the original treatment in Lafont (1990) proposes a form of type system in which the type symbols each come in two dual flavours, and only port connections which feature a pair of port edges which are typed by type symbols which are duals of one another are allowed. This new invariant is trivial to preserve during rewriting, and can quickly be used to give completeness of rulesets by providing rules for all correctly typed cuts that can be created from the set of agents of the system. But this is not the only possibility. In the appendix, we show how the proof nets of LLM can be understood as INs, and this construction uses a different type system altogether. One could presumably go further and adapt the essence of a type system for graph rewriting such as in Banach (1992) to yield yet other and more powerful type disciplines. Once more, these issues will be explored elsewhere. As with CR and termination, the reader who is with us thus far is probably grateful that we do not launch into these things here.

Acknowledgement

It is a pleasure to thank Yves Lafont for comments on an earlier draft of this paper.

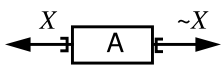
Appendix. LLM Proof Nets as INs.

In this appendix we will briefly describe the proof nets of classical multiplicative linear logic with constants (LLM) in the terminology of interaction nets.

Proof nets are built up as follows. Fig. 43 shows on the left, the traditional right handed sequent introduction rules, and on the right, what would be the consequents of the corresponding IN introduction rules.

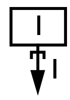
Axioms:

$$X, \sim X$$



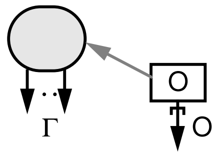
Constant I:

$$\mathsf{I}$$



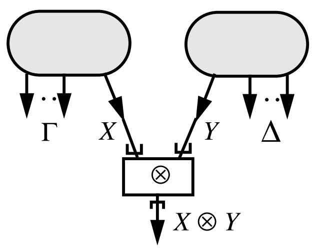
Constant O:

$$\frac{\Gamma}{\Gamma, \mathsf{O}}$$



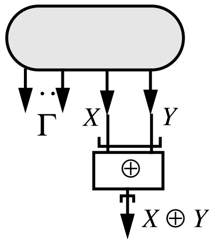
Times:

$$\frac{\Gamma, X \qquad Y, \Delta}{\Gamma, X \otimes Y, \Delta}$$



Par:

$$\frac{\Gamma, X, Y}{\Gamma, X \oplus Y}$$



Cut:

$$\frac{\Gamma, X \quad \sim X, \Delta}{\Gamma, \Delta}$$

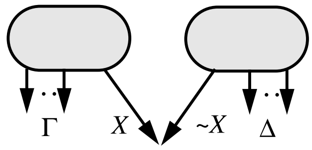


Fig. 43.

Note that the port edges of the agents are typed (by X, Y …), which range over formulae of LLM, which are made up of variables (x, y …) and the constants I and O, via the connectives $\otimes$, $\oplus$, ~ , and are such that the obvious double negation and De Morgan Laws are meta level equivalences. The Cut rule imposes the type consistency appropriate to LLM cuts, and there is an implicit Compose rule stating that a principal port may be connected to a non-principal port, whereupon their types must match; but two non-principal ports may never be connected together. The next result is standard enough and the proof is merely sketched.

Theorem A.1 For each right handed sequent proof there is a corresponding OEMSN, and hence an ESSP satisfying (1) – (5) of theorem 9.28 with the additional properties that it is breast-free and each free port is principal. Conversely, for each such ESSP, there is a corresponding right handed sequent proof.

Proof. The first part is more or less self-evident by induction, given Fig. 43. For the second part, given a fixed algebra of the EMSN, theorem 9.31 does not quite give us what we want since it implicitly (by theorem 8.25) encourages the removal of all Par agents and Constant-O agents first, and subsequently the removal of other agents in any order, both being activities which potentially leave free ports which are not principal. Neither does theorem 9.31 or theorem 8.25 deal with Cuts.

We adopt the following procedure, which we apply repeatedly to a net, to reduce it to the empty net in such a way that the pieces left over at each stage are of the required kind. The general properties of EMSNs are enough to show that while the net is still non-empty, an application of the procedure will do some work.

Procedure REDUCE:

If the net is just a single axiom agent or single constant-I agent, remove it.

Else if there is a free principal port which is the conclusion of a $<_E$-maximal Par or constant-O agent, remove the agent. (NB. Here $<_E$ is the partial order of theorem 9.31.

This step leaves behind a net as required, since a $<_E$-maximal Constant-O agent may obviously be removed, and a $<_E$-maximal Par, being breast-free, has a proper scope of the required form.)

Else if there is a Cut node in the net which is an articulation node of the net, split it apart (which obviously preserves the properties required).

Else if there is a free principal port which is the conclusion of a Times agent which is an articulation agent of the net, remove the agent (ditto).

The above sketch follows the customary inductive form. For an alternative geometric treatment, also in sympathy with the techniques of this paper, see Banach (1996).

Cut elimination consists of two net rewrite rules. The first is in Fig. 44.

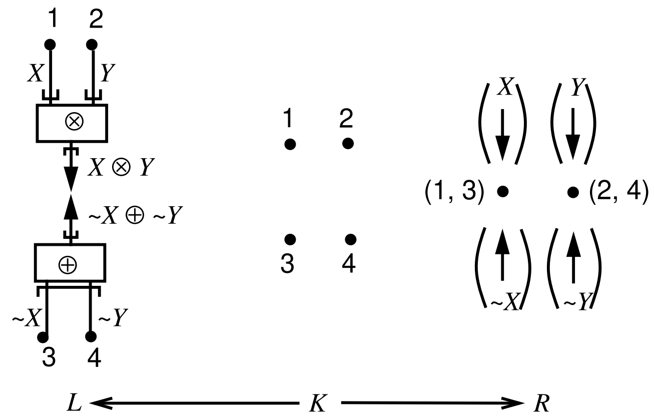


Fig. 44.

Note that if L occurs in some EMSN G, then in the corresponding D, port nodes 1 and 2 are ESF-disconnected, while 3 and 4 are ESF-strongly connected. For this reason, if either agent of L is a victim of an incoming probe (in some algebra of G), an alternative algebra may be found in which the victim is an agent adjacent to L (there must be at least three such agents since the Par is breast-free); whence we may assume L has no victims. It is now easy to see that the rule preserves the EMSN property, though not for the reasons described in theorem 9.45.

Note also that because of the typing rules, the Cuts in R created by short circuiting (1, 3) and (2, 4) are well typed.

The second rule is given as a rule schema in Fig. 45 and involves the Axiom agent.

Note that since Axiom agents have more than one principal port, Fig. 45 is in fact ambiguous. Which of the two principal ports of the Axiom agent is actually in the Cut? Well of course the schema can — and is intended to — cover both possiblities.

As a special case of Fig. 45 note Fig. 46 which is even more ambiguous, covering three essentially different LHS's, and two ways of connecting the RHS to the one irreflexive instance of the LHS, making four rules in all. Thus we have a more general situation than is covered by proposition 4.5, both because Axiom agents have more than one prin-

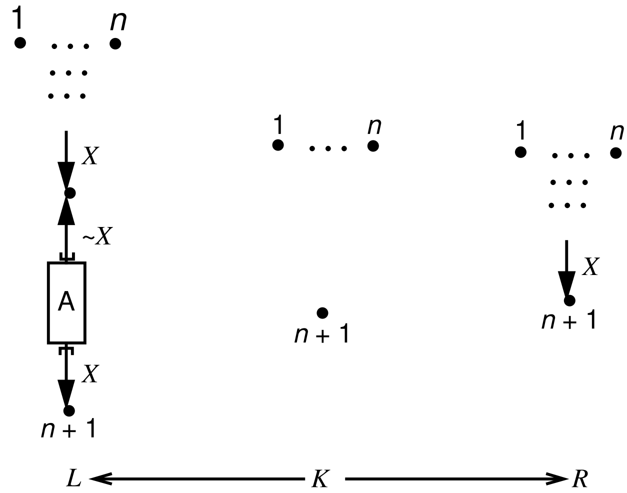


Fig. 45.

cipal port, and because there is a reflexive rule which is not symmetric. Nevertheless, it is not hard to see that regardless of which of the many different rules covered by Fig. 45 and Fig. 46 we consider, any overlapping redexes that can arise, occur only when we have chains of axioms of compatible type, which can be reduced in any order to give results isomorphic up to the replacement of any axiom agent by one which is oppositely oriented and oppositely typed; i.e. up to the swapping of port labels on axioms. Thus we get an acceptable form of the Church-Rosser property anyway. (Axiom agents are of course "fat short circuits" in the terminology of section 7).

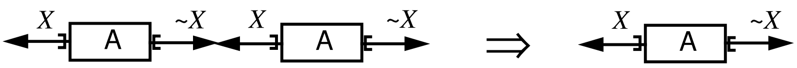


Fig. 46.

Obviously all of these rules satisfy the hypotheses of proposition 4.2 so Cut elimination in LLM is strongly normalising. Thus we recover the unique normal form theorem of LLM.

To the author's mind, the system we have presented is almost completely "logic-free". What the type system gives us, is that only cuts of the kind for which we presented rules, actually occur, i.e. it is easy to check that given a well typed net, any rewrite yields another well typed net because of the Cut and Compose rules. This is what type systems are for of course.

But we can abandon the type system. Suppose we introduced suitable rules whose RHSs had fewer nodes than their LHSs, for all possible Cuts that we could make out of the agents we have defined and containing no new agents, and such that rewriting was suitably subcommutative for the cases involving the Axiom agent, (all of which is perfectly possible). Then we could still have a unique normal form deadlock-free system, even if a perhaps rather meaningless one. This is because we would still have a principal ESS system that satisfied theorem 7.10 (say). The point is that the very notion of principality and the restriction of interaction to Cuts, is already a type system of a kind, and moreover one that in conjunction with E-supersimplicity and the other ingredients, is powerful enough to yield the strong properties we want.

To the author's mind linear logic contains many layers, and it is not always easy to see which parts are responsible for which properties of the system. He hopes that the remarks above and the other contents of this paper shed light from a slightly different angle onto LLM, and particularly onto what its strong properties for rewriting might be attributable to.

References

Aho A.V., Sethi R., Ullman J.D. (1986), Compilers, Principles, Techniques and Tools. Addison Wesley, Reading Mass.

Banach R. (1992), Simple Type Inference for Term Graph Rewriting Systems. in Proc. CTRS-92, Rusinowitch, Remy (eds.), Lecture Notes in Computer Science, **656**, 51-66, Springer, Berlin.

Banach R. (1996), Sequent Reconstruction in LLM — A Sweepline Proof. Annals of Pure and Applied Logic, to appear.

Bawden A. (1986), Connection Graphs. in: Proc. A.C.M. Conference on Lisp and Functional Programming, 258-265.

Bellin G., Scott P.J. (1994), On the Pi-Calculus and Linear Logic, Theoretical Computer Science, **135**, 11-65.

Blute R.F., Cockett J.R.B., Seely R.A.G., Trimble T.H. (1993), Natural Deduction and Coherence for Weakly Distributed Categories. manuscript, Maths. Dept., McGill University.

Danos V. (1990), La Logique Linéare Appliquée a' l'Éttude de Divers Processus de Normalisation (Principalement du Lambda-Calcul). Thése de Doctorat, Université Paris VII.

Danos V., Regnier L. (1989), The Structure of Multiplicatives. Archive for Mathematical Logic **28**, 181-203.

Ehrig H. (1979), Introduction to the Algebraic Theory of Graph Grammars (A survey), in: Lecture notes in Computer Science **73**, 1-69, Springer, Berlin.

Ehrig H. (1986), A Tutorial Introduction to the Algebraic Approach of Graph Grammars. in: Third Int. Workshop on Graph Grammars, Lecture Notes in Computer Science **291**, 3-14, Springer, Berlin.

Ehrig H., Habel A., Kreowski H-J., Parisi-Presice F. (1990), From Graph Grammars to High Level Replacement Systems. in: Fourth Int. Workshop on Graph Grammars and their Applications to Computer Science, Ehrig, Kreowski, Rozenberg (eds.), Lecture Notes in Compter Science **532**, 269-291, Springer, Berlin.

Girard J-Y. (1987), Linear Logic. Theoretical Computer Science **50**, 1-102.

Girard J-Y. (1988), Multiplicatives. in: Rendiconti del Seminario Mathematico dell'Università e Politecnico di Torino; special issue on logic and computer science, 11-33.

Hillis W.D. (1985), The Connection Machine. MIT Press.

Lafont Y. (1990), Interaction Nets. in: Proc. Seventh A.C.M. Symposium on Principles of Programming Languages, 95-108.

Lafont Y. (1991), The Paradigm of Interaction. manuscript.

Löwe M. (1993), Algebraic Approach to to Single Pushout Graph Transformation. in Theoretical Computer Science, **109**, 181-224.