MONSTR V — Transitive Coercing Semantics and the Church-Rosser Property

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

Abstract: The transitive coercing semantic model for the execution of the MONSTR generalised term graph rewriting language is defined. Of all the operational semantics for MONSTR that one might consider, this one has the cleanest properties. Under intuitively obvious conditions for executions involving redexes permitted to overlap sufficiently to allow the programming of deterministic synchronisations, and despite the failure of exact subcommutativity, a Church-Rosser theorem is proved to hold up to markings and garbage.

1 Introduction

In the first MONSTR paper ([Banach (1996b)], hereafter referred to as **M-I**), we introduced the MONSTR generalised term graph rewriting language, together with its operational semantics, and the architectural rationale behind its design. We also briefly described some other semantic models for MONSTR and the correctness problems that they engender when soundness with respect to the original semantics is desired. In subsequent papers (**M-II** [Banach (1997a)], **M-III** [Banach (1997b)], **M-IV** [Banach (1997c)]), we treated such correctness problems in detail, concentrating on issues connected with serialisability properties of finegrained operational semantic models, such as might reflect the behaviour of actual implementations on distributed architectures.

In this paper we introduce the transitive coercing semantic model. Because the model is already coercing, it enjoys the good serialisability properties of coercing models for finegrained implementation as discussed in **M-IV**. The addition of transitivity of redirections ([Banach (1996a)]) gives the model better subcommutativity properties, and so the emphasis in this paper is on the Church-Rosser property. (In truth, for the models considered earlier in this series, analogous Church-Rosser theorems hold, but because of the weaker subcommutativity properties of those models, collections of rather ugly side conditions have to be included in the hypotheses, and this makes such theorems of less interest. Some indication of what is involved here may be gleaned from Section 6 of **M-II** where subcommutativity results are worked out for the standard (suspending) semantics, as an aid to the verification of systems implemented in MONSTR, without proceeding to a full Church-Rosser theorem.)

MONSTR is worth studying for number of reasons. First it is a graph rewriting framework deliberately cast close to the capabilities of real implementations. Its expressiveness therefore combines the abstractness of graphs and their arbitrary interconnectivity, with very pragmatic considerations, and translations of other systems into MONSTR describe both potential implementation routes, and something of the naturalness or otherwise of the primitives offered by such systems. MONSTR has with some justification been called an abstract assembly language, and a wider ranging discussion setting MONSTR in context and with suitable references to earlier work can be found in **M-I**.

Secondly, MONSTR provides a model for concurrent computations that combines the abstractness of graph structure and of the potential to encode arbitrary information within the labels of the graphs, with a very specific rewrite strategy control mechanism, based on considerations of concrete machine design. MONSTR is also sensitive to the particular feature of many real computing systems, whereby some parts of the current computational state are active and others are passive, the latter being manipulated by the former; and in a real system the active may in time become passive and vice versa. Often the nondeterminism in a real system finds expression in the competition between active parts to perform incompatible manipulations on the passive parts. Ideally, and in a clean model of computation, this would be the only source of nondeterminism as regards the final answer computed by the system. However most low level models of computation that are as operational as MONSTR, do not have such pleasant semantic properties, at least not in an elegantly expressible form. And the same is true of the semantic models for MONSTR considered earlier in this series of papers. However, the model presented here, the culmination of the series, possesses particularly strong properties in this respect. Provided synchronisations are deterministic, i.e. provided there is a (locally) unique outcome whatever the order of arrival of interested processes at a particular piece of passive computational state, the local determinism extends to a global determinism, resulting in a Church-Rosser property. This is what we prove here.

The rest of the paper is as follows. Section 2 defines transitive term graphs, and gives the (abstract) syntax of MONSTR rules and systems. Though the treatment is mathematically self contained, no attempt is made to motivate the definitions. The reader who is left uneasy by Section 2 should consult **M-I** which is largely concerned with such motivational issues. For future reference, notation such as **M-I**.11.4 refers to the fourth listed item of Section 11 of **M-I**. Section 3 defines transitive coercing semantics precisely, and contains additional motivational remarks at the end. Then Section 4 covers the fundamental invariants of MONSTR, balancedness and state saturatedness, while Section 5 discusses garbage, an issue which plays a significant role in the Church-Rosser theorem subsequently.

Typical MONSTR systems feature a lot of sharing and thus a lot of overlapping redexes, in order to express the synchronisations that concurrent systems need. Thus a Church-Rosser theorem that merely deals with the obvious analogue of orthogonal systems in which redexes may not overlap at all (or hardly at all) yields a relatively weak result, with little applicability to practice and to the situations described previously. So Section 6 deals with overlapping redexes, identifying cases in which redexes may overlap in a benign manner. The paper would be of less value were it not for the fact that the synchronisation situations these cases allow us to reason about are in fact just the ones that turn out to be practically useful in typical real MONSTR systems. Section 7 sets out the subcommutativity results on which the main theorem is based. We see there, that the elementary atomic actions of transitive coercing MONSTR do not actually subcommute in all the desired situations. This adds some technical spice to the Church-Rosser theorem of Section 8, where we see that the discrepancies in subcommutativity cause the expected filling in of the Church-Rosser diamond to flake into distinct sheets. Fortunately, a careful analysis reveals that the discrepancies are sufficiently benign that they do not actually block the construction, and all relevant sheets can be filled in as desired. It is perhaps worth mentioning that not all systems with the Church-Rosser property satisfy the hypotheses of the main theorem, though these are fairly rare. An interesting case in point is the efficient translation of untyped interaction nets into

MONSTR ([Banach and Papadopoulos (1997)]), where more drastic overlapping of redexes than permitted here nevertheless leads to confluent results. Section 9 concludes.

2 Transitive Term Graphs and the Syntax of MONSTR Systems

In this section we deal with basic syntactic matters. For readers familiar with **M-I**, the graphs of this paper contain “bottom-nodes” whose nature will become clear in the next section.

We assume we are given an alphabet $\mathbf{S} = \{\mathsf{S}, \mathsf{T}, \dots\}$ of node symbols and in addition, two further special symbols Any and $\perp$ (bottom) which are not in $\mathbf{S}$. When we wish to refer to specific symbols we will write them thus S, T; but when we speak about symbols in general in the meta-language we will use italics thus S, T.

Definition 2.1 A transitive term graph (or just graph) G, is a quintuple $(N, \sigma, \alpha, \mu, \nu)$ where

(1) N is a set of nodes.

(2) σ is a map $N \to \mathbf{S} \cup \{\perp\}$, which labels each node.

(3) α is a map $N \to N^*$, which maps each node to its sequence of children.

(4) μ is a map $N \to \{\varepsilon, *, \#, \#\#, \#\#\#, \dots \#^n\ (n \geq 1)\}$, which maps each node to its node marking (idle, active, once, twice … n times suspended).

(5) ν is a map $N \to \{\varepsilon, \hat{}\}^*$, which maps each node to the sequence of arc markings on the arcs to its children (each either the normal or notification marking).

Clearly we must have for all $x \in N$, $\mathrm{dom}(\alpha(x)) = \mathrm{dom}(\nu(x))$, where the domain of a sequence is the set of its indices. And furthermore, $\perp$-labelled nodes ($\perp$-nodes), satisfy

(BOT) $\sigma(x) = \perp \ \Rightarrow\ \alpha(x) = \nu(x) = \varnothing$

i.e. $\perp$-nodes are always childless.

We write $A(x)$, the arity of a node x, for $\mathrm{dom}(\alpha(x)) = \mathrm{dom}(\nu(x))$. Note that $A(x)$ is a set of consecutive natural numbers starting at 1, or empty. When dealing with more than one graph (or pattern — see below), we subscript the objects defined in (1) – (5) above with the name of the graph in question to avoid ambiguity. Also we allow ourselves to write $x \in G$ instead of $x \in N(G)$ or $x \in N_G$ etc. Each child node c of some node p determines an arc of the graph, and we will refer to arcs using the notation (p_k, c) to indicate that c is the k’th child of p; i.e. that $c = \alpha(p)[k]$ for some $k \in A(p)$. The maps μ, ν are referred to as the markings and are mainly concerned with encoding execution strategies, while N, σ, α are referred to as the graph structure and provide the main information content of the graph.

For ease of use, the names are meant to be reasonably alliterative: σ for symbols, α for arcs, μ for markings, ν for notifications.

Fig. 1 below shows a term graph, in which each node is depicted by its symbol followed by its sequence of out-arcs in brackets, and only non-idle markings are shown. Obviously transitive term graphs are directed graphs. We use standard digraph terminology below where necessary without further comment; eg. path, semipath, and accessibility of one node from another. (Recall a semipath ignores the orientation of arcs.)

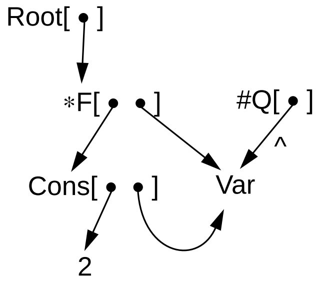

Fig. 1 A graph.

For rewriting, we will need a notion of pattern, and a sufficiently flexible notion of pattern matching.

Definition 2.2 A pattern is defined as in definition 2.1 except that the signature of σ is $N \rightarrow \mathbf{S} \cup \{\mathsf{Any}\}$, and Any-labelled nodes must satisfy

(ANY) $\sigma(x) = \mathsf{Any} \Rightarrow \alpha(x) = \nu(x) = \varnothing$

In patterns and graphs, among the non-$\perp$-nodes, Any-labelled nodes (Any-nodes) are also called implicit whereas other nodes are explicit.

Homomorphisms relate patterns to graphs. Apart from the expected preservation of structure, readers should note the asymmetry of the roles of the Any-nodes and $\perp$-nodes.

Definition 2.3 Let P be a pattern and G be a graph (and let P have a root r). A node map $h : P \rightarrow G$ is a homomorphism, or matching, to G (at $t \in G$) iff ($h(r) = t$ and) for all nodes $x \in P$ such that $\sigma(x) \neq \mathsf{Any}$ and $\sigma(h(x)) \neq \perp$

(1) $\sigma(x) = \sigma(h(x))$, i.e. h is label-preserving.

(2) $A(x) = A(h(x))$, i.e. h is arity-preserving.

(3) For all $k \in A(x)$, $h(\alpha(x)[k]) = \alpha(h(x))[k]$, i.e. h is order-preserving.

A homomorphism is proper iff $\perp \notin \{\sigma(h(x)) \mid x \in P \text{ and } x \text{ is explicit}\}$.

Suppose in addition the following hold:

(4) $\mu(x) = \mu(h(x))$, i.e. h is node-marking-preserving.

(5) For all $k \in A(x)$, $h(\nu(x)[k]) = \nu(h(x))[k]$, i.e. h is arc-marking-preserving.

In such a case we say that h preserves markings. (To emphasise the converse when required, we call ordinary homomorphisms, graph structure homomorphisms.)

Omitting mention of roots, definition 2.3 serves just as well for homomorphisms between graphs and homomorphisms between unrooted patterns, as it does for rooted patterns and graphs.

Definition 2.4 A rule D is a quadruple (P, *root*, *Red*, *Act*) where

(1) P is a pattern, called the full pattern of the rule.

(2) *root* is an explicit node of P called the root, and all implicit nodes of P are accessible from the root. If $\sigma(root) = S$, then D is a rule for S. The subpattern of P of nodes and arcs accessible from (and including) *root* is called the left pattern L of the rule, and nodes of P not in L are called contractum nodes. L is unmarked, i.e. for all $x \in L$, $\mu(x) = \varepsilon$, and $\nu(x)[k] = \varepsilon$ for all $k \in A(x)$.

(3) *Red* is a set of pairs of nodes, (called redirections) such that $Red \subseteq L \times P$, and *Red* satisfies the invariants (RED-1), (RED-2) and (RED-3) below:

(RED-1) *Red* is the graph (in the set theoretic sense) of a partial function.

(RED-2) $(a, b) \in Red \Rightarrow a$ is an explicit node of L.

(RED-3) $\{(a, b), (a', b')\} \subseteq Red$ and $a \neq a' \Rightarrow \sigma(a) \neq \sigma(a')$.

For $(a, b) \in Red$, a is called the LHS and b the RHS of the redirection, and we say that the rule redirects a.

(4) *Act* is a set of nodes (called activations) of P such that $Act \subseteq L$.

Fig. 2 is a picture of a rule, with *root* indicated by the short stubby arrow, *Red* indicated by the dashed arrows, and *Act* indicated by adorning the relevant nodes of L with a $*$ (these are unmarked according to definition 2.4.(2)).

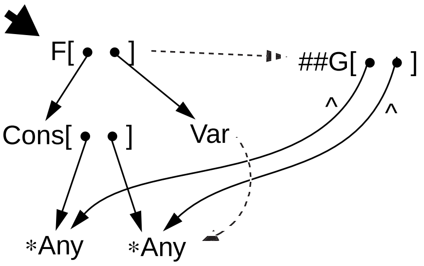

Fig. 2 A rule.

Definition 2.5 A rule $D = (L \subseteq P, root, Red, Act)$ matches (or is applicable to) a graph G at t iff $\mu(t) = *$ and there is a proper matching $g : L \rightarrow G$ at t. We call the image of such a matching a redex.

It is not hard to see that the rule of Fig. 2 matches at the F-labelled node of Fig. 1.

For reasons discussed at length in **M-I**, it is necessary from a distributed implementation standpoint, to circumscribe the generality permitted by definition 2.4. The deliberations in **M-I** distil down to the following combinatorial properties, quoted (almost) verbatim from **M-I**, which define the abstract syntax of MONSTR systems.

Restriction M-I.11.1 The alphabet of symbols **S**, is the disjoint union of three subalphabets

$$\mathbf{S} = \mathbf{F} \cup \mathbf{C} \cup \mathbf{V}$$

F is the alphabet of function symbols. A function symbol may label the root of the left pattern *L* of a rule, but not any subroot node of *L*. Function symbols may label the LHS of a redirection.

C is the alphabet of constructor symbols. A constructor symbol may label a subroot node of the left pattern of a rule, but not the root. Constructors may not label the LHS of a redirection.

V is the alphabet of stateholders, or variables. A stateholder symbol may label a subroot node of the left pattern of a rule, but not the root. Stateholders may label the LHS of a redirection.

The functions act as instigators of rewrites, the constructors encode immutable values, while the stateholders are able to model notions of updatable state, and to play a central role in the coding of synchronisation primitives.

Restriction M-I.11.2

(1) For each $S \in \mathbf{S}$, there is a set of natural numbers $A(S)$, in every case an initial segment of the naturals from 1, or empty.

(2) For each $F \in \mathbf{F}$, there is a subset of $A(F)$, Map(F).

(3) For each $F \in \mathbf{F}$, there is a subset of Map(F), State(F), in every case either a singleton or empty.

(4) $\mathsf{Root} \in \mathbf{C}$.

The above maps each symbol *S* to its arity $A(S)$. The intention is that all *S*-labelled nodes are to have the same arity. For functions *F*, Map(F) is the set of argument positions at which all normal rules for *F* (see below), will always need to pattern match. Similarly State(F), if non-empty, contains the position at which any stateholder argument of *F* must occur in a normal rule for *F*. Clause (4) states that Root is a constructor, a fact used in the theory of garbage collection.

Definition M-I.11.3 Let $F \in \mathbf{F}$. A rule for *F* such that each child of the root is a distinct implicit node is called a default rule for *F*. Otherwise the rule is a normal rule.

Note that with fixed arities, a default rule for *F* will always succeed in matching its left pattern to any active *F*-labelled node of a graph, precisely because no-non trivial conditions need to be satisfied by the children of the root of the redex.

Restriction M-I.11.4 Let $D = (P, root, Red, Act)$ be a rule with left pattern *L*. Then

(1) Each node has the arity dictated by its symbol, i.e.

$$\text{For all } x \in P,\ A(x) = A(\sigma(x))$$

(2) Each normal rule for a symbol matches the same set of arguments of the root, i.e. if $\sigma(root) = F$, and *D* is a normal rule then

$$\alpha(root)[k] \text{ is explicit } \Leftrightarrow k \in \text{Map}(F)$$

(3) A rule for a function may match at most one stateholder, and then only in a fixed position (the stateholder position); all other explicit arguments must be construc-

tors (occurring in constructor positions), i.e. if $\sigma(root) = F$, and D is a normal rule then

$$\sigma(\alpha(root)[k]) \in \mathbf{V} \Rightarrow k \in \text{State}(F)$$

(4) All grandchildren of the root are implicit, i.e. for all $k \in A(\sigma(root))$, and $j \in A(\sigma(\alpha(root)[k]))$

$$\alpha(\alpha(root)[k])[j] \text{ is implicit}$$

(5) Implicit nodes of the left pattern have only one parent in the left pattern, i.e. if $y \in P$ is implicit, there is precisely one $x \in L$ such that for some $k \in A(x)$, $y = \alpha(x)[k]$.

(6) Every $x \in P$ is balanced, i.e.

$$\mu(x) = \#^n \text{ (for } n \geq 1) \Leftrightarrow |\{k \mid \nu(x)[k] = \text{\textasciicircum}\}| = n$$

(7) Every arc (p_k, c) of P is either state saturated or activated, i.e.

$$\nu(p)[k] = \text{\textasciicircum} \text{ and } \mu(c) = \varepsilon \Rightarrow \sigma(c) \in \mathbf{V} \text{ or } c \in Act$$

(8) The root is always redirected, i.e. for some $b \in P$

$$(root, b) \in Red$$

(9) No arc can lose state saturatedness through redirection, i.e.

$$(a, b) \in Red \text{ and } \mu(b) = \varepsilon \Rightarrow \sigma(b) \in \mathbf{V} \text{ or } b \in Act$$

(10) A node which is the LHS but not the RHS of a redirection should be garbaged by a rewrite whenever possible, i.e.

$$(b, c) \in Red \text{ and } b \in Act \Rightarrow \text{ there is a } b \neq a \in L \text{ such that } (a, b) \in Red$$

Restriction M-I.11.6 For each $F \in \mathbf{F}$ there is a pair of sets $(\mathcal{N}_F, \mathcal{D}_F)$, where $\mathcal{N}_F$ consists of normal rules for F, and $\mathcal{D}_F$ is non-empty and consists of just default rules for F. An assignment of such pairs to each $F \in \mathbf{F}$ constitutes a MONSTR system.

The above gives the syntax of MONSTR systems. In addition, **M-I** defines a couple of builtins for testing pointer equality, but we will not be concerned with these in this paper.

3 Transitive Coercing Semantics

Definition M-I.3.13 An initial graph is one which consists of a single node with empty arity, with the active node marking, and labelled by the symbol $\mathsf{Initial}$.

Definition M-I.3.14 A preexecution $\mathcal{G}$ of a system $\mathcal{R}$ is a sequence of graphs $[G_0, G_1 \ldots]$ such that G_0 is initial and for each $i \geq 0$ such that i+1 is an index of $\mathcal{G}$, G_{i+1} results from G_i by some execution step at some arbitrarily selected active node t_i of G_i (i.e. a node t_i for which $\mu(t_i) = *$). If the sequence is of maximal length, it is called an execution. Graphs occuring in (pre)executions are called execution graphs.

Though the above is (essentially) taken from **M-I**, it applies equally well here. All that remains is to define execution steps, and the rules which state how one chooses between them at any particular active node. Transitive coercing execution steps are of three kinds: notifications, rewrites and suspensions. The next definition states the circumstances under which each kind of action is performed.

Definition 3.1 Let G be a graph and t an active node of G, the chosen root. For transitive coercing semantics, the kind of execution step to be performed at t is determined as follows.

```
If  $\sigma(t) \in \mathbf{C} \cup \mathbf{V}$
Then  Perform a notification at $t$
Else  If   For all $k \in$ Map($\sigma(t)$), $\mu(\alpha(t)[k]) = \varepsilon$  (and $\nu(t)[k] = \varepsilon$),
           and for all $k \in$ State($\sigma(t)$), $\sigma(\alpha(t)[k]) \in \mathbf{C} \cup \mathbf{V}$,
           and for all $k \in$ (Map($\sigma(t)$) – State($\sigma(t)$)), $\sigma(\alpha(t)[k]) \in \mathbf{C}$
      Then     Perform a rewrite using a rule chosen
               nondeterministically from $Sel$ where
                    If  some rule from $\mathcal{N}_{\sigma(t)}$ matches the chosen root $t$
                    Then  $Sel = \{D \in \mathcal{N}_{\sigma(t)} \mid D$ matches at $t\}$
                    Else  $Sel = \mathcal{D}_{\sigma(t)}$
      Else (If For some $k \in$ Map($\sigma(t)$), $\mu(\alpha(t)[k]) \neq \varepsilon$,
               or for $k \in$ State($\sigma(t)$), $\sigma(\alpha(t)[k]) \notin \mathbf{C} \cup \mathbf{V}$,
               or for some $k \in$ Map($\sigma(t)$) – State($\sigma(t)$), $\sigma(\alpha(t)[k]) \notin \mathbf{C}$
      Then)    Perform a suspension at $t$
```

We note incidentally that if the conditions for a rewrite hold, then either a normal or a default rule will definitely match according to the criterion of definition 2.5.

Now we define the individual types of step. We start with the simplest cases. Notification causes the chosen root to be quiesced (i.e. to have its active marking removed), and for most notification in-arcs of the chosen root, their notification marking is removed and parent nodes of such in arcs have any non-zero suspension marking decremented. "Most" in the preceding sentence refers to all notification arcs which do not connect functions to stateholder children occuring in matched but not stateholder position.

Fig. 3 shows a notification in a fragment of a graph, (assuming 2 is a constructor).

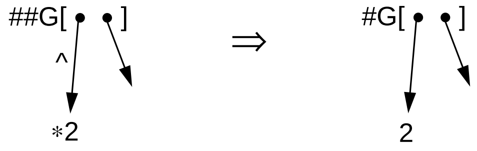

Fig. 3 A notification.

More formally we have the following.

Definition 3.2 Let t be the chosen root in a graph G with $\sigma(t) \in \mathbf{C} \cup \mathbf{V}$. Let the graph H be given by

(1) $N_H = N_G$.

(2) $\sigma_H = \sigma_G$.

(3) $\alpha_H = \alpha_G$.

(4) $\mu_H(x) =$ **If** $\mu_G(x) = \#^n$ (with $n \geq 1$) and
$0 \neq m = |\{k \in A(x) \mid \alpha_G(x)[k] = t$ and $\nu_G(x)[k] = \wedge$ and
not[$\sigma_G(x) \in \mathsf{F}$ and $k \in (\text{Map}(\sigma_G(x)) - \text{State}(\sigma_G(x)))$
and $\sigma_G(t) \in \mathsf{V}$] $\}|$
Then $\#^{n-m}$ (where $\#^0 = *$, and $\#^{-p} = \varepsilon$ for $p \geq 1$)
Else If $x = t$ **Then** ε
Else $\mu_G(x)$.

(5) $\nu_H(x)[k] =$ **If** $\alpha_G(x)[k] = t$ and $\nu_G(x)[k] = \wedge$ and not[$\sigma_G(x) \in \mathsf{F}$ and
$k \in (\text{Map}(\sigma_G(x)) - \text{State}(\sigma_G(x)))$ and $\sigma_G(t) \in \mathsf{V}$]
Then ε
Else $\nu_G(x)[k]$.

The result of the notification is the graph *H*.

Suspensions occur when not all the matched arguments of a function at the chosen root are in the required form, by virtue of being non-idle, or of being ⊥-nodes, or of being functions, or of being stateholders in a non-stateholder position. The suspension makes the chosen root suspended on all such arguments till the required state of affairs obtains, activating any idle functions thus encountered. Fig. 4 shows a suspension step for a fragment of term graph rooted at an F-labelled chosen root. The assumption is that all four arguments of F are in Map(F), that G is a function, that S is a stateholder, and that State(F) = {2}.

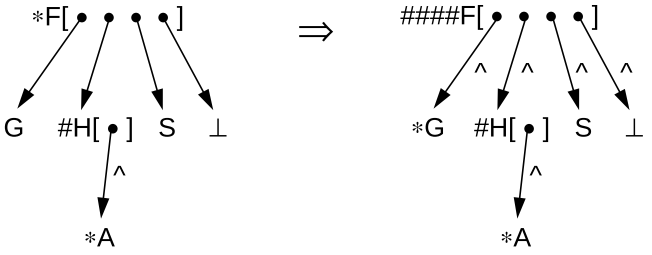

Fig. 4. A suspension step.

More formally we have the following.

Definition 3.3 Suppose *t* is a chosen root in a graph *G*, $\sigma(t) \in \mathsf{F}$ and there is at least one $k \in \text{Map}(\sigma(t))$ such that either $\alpha(t)[k]$ is a ⊥-node, or $\alpha(t)[k]$ is non-idle, or $\alpha(t)[k]$ is an idle function, or $\alpha(t)[k]$ is an idle stateholder with $k \notin \text{State}(\sigma(t))$. Let

Susp(t) = $\{k \in \text{Map}(\sigma_G(t)) \mid \alpha_G(t)[k]$ is non-idle, or
$\alpha_G(t)[k]$ is idle and $\sigma_G(\alpha_G(t)[k]) = \perp$, or

$\alpha_G(t)[k]$ is idle and $\sigma_G(\alpha_G(t)[k]) \in \mathsf{F}$, or
[$k \in$ (Map($\sigma_G(t)$) – State($\sigma_G(t)$)) and
$\alpha_G(t)[k]$ is idle and $\sigma_G(\alpha_G(t)[k]) \in \mathsf{V}$] }

$n = |\,\mathrm{Susp}(t)\,|$

Define the graph H as follows.

(1) $N_H = N_G$.

(2) $\sigma_H = \sigma_G$.

(3) $\alpha_H = \alpha_G$.

```
(4)  $\mu_H(x) =$ **If** $x = t$
         **Then** $\#^n$
         **Else If** $x = \alpha_G(t)[k]$ and $k \in$ Susp($t$) and $\alpha_G(t)[k]$ is idle and
                 $\sigma_G(\alpha_G(t)[k]) \in \mathsf{F}$
         **Then** $*$
         **Else** $\mu_G(x)$.
```

```
(5)  $\nu_H(x)[k] =$ **If** $x = t$ and $k \in$ Susp($t$)
         **Then** ^
         **Else** $\nu_G(x)[k]$.
```

Then H is the result of the suspension.

We define the maps $i_{G,H} = r_{G,H}$ as the identity on nodes for notification and suspension steps, in order to be able to track the fate of nodes through executions using a notation uniform with that for the relevant maps for rewrite steps, which are introduced as we proceed with the definition of the latter now.

Once a redex has been identified, a rewrite consists of four phases, namely contractum building, bottom analysis, redirection and activation. Assume G, t, $D = (P, root, Red, Act)$ and g given as necessary, in the notation of Section 2. We will use the matching of Fig. 2 at the F-labelled node of Fig. 1 as a running example.

Contractum building adds a copy of each contractum node of P to G. Node markings for such nodes are taken from P. Copies of arcs of P from contractum nodes to their children are added in such a way that there is a graph structure homomorphism (called the extended matching) $g' : P \rightarrow G'$ from the whole of P to the graph being created, which agrees with g on L. Arc markings are again taken from P.

Doing this for our running example yields Fig. 5. We see that copies of exactly the contractum nodes and arcs, suitably marked, have been added, and that this enables the extended matching g' of the whole of P to be constructed.

More formally we have the following.

Definition 3.4 Assume the preceding notation. Let the graph G' be given by

(1) $N_{G'} = N_G \uplus (N_P - N_L)$ where $\uplus$ is disjoint union.

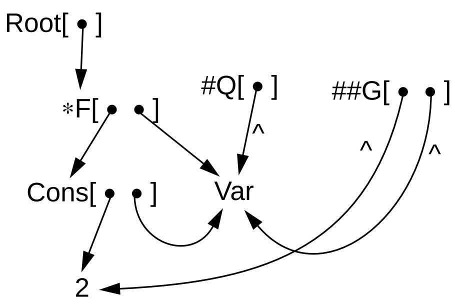

Fig. 5 Contractum Building.

(2) $\sigma_{G'}(x) = \sigma_G(x)$ if $x \in G$,
$\sigma_{G'}(n) = \sigma_P(n)$ if $n \in P - L$.

(3) $\alpha_{G'}(x)[k] = \alpha_G(x)[k]$ if $x \in G$, for $k \in A(x)$,
$\alpha_{G'}(n)[k] = \alpha_P(n)[k]$ if both n and $\alpha_P(n)[k] \in P - L$, for $k \in A(n)$,
$g(\alpha_G(n)[k])$ if $n \in P - L$ and $\alpha_P(n)[k] \in L$, for $k \in A(n)$.

(4) $\mu_{G'}(x) = \mu_G(x)$ if $x \in G$,
$\mu_{G'}(n) = \mu_P(n)$ if $n \in P - L$.

(5) $\nu_{G'}(x)[k] = \nu_G(x)[k]$ if $x \in G$, for $k \in A(x)$,
$\nu_{G'}(n)[k] = \nu_P(n)[k]$ if $n \in P - L$, for $k \in A(n)$.

Note the use of disjoint union above. In constructive definitions of disjoint union, the members of such a union are tagged so that one can discern their origin. Definition 3.4 omitted to do this. This is not normally a source of difficulty unless one is interested in more “global” issues. In fact we will be confronted by some of these later in this paper, and so a proper definition needs to take this into account. In such a case, a node x in G and its representative in G' after contractum building, are no longer the same thing, and there is a natural injection $i_{G,G'} : G \to G'$ that takes x to its representative in G'. We let $r_{G,G'}$ be another name for $i_{G,G'}$, as for notifications and suspensions.

Bottom analysis does nothing to the structure of the graph itself, but prepares the ground for the details of the next phase. It consists of the following observations. Let

$$Red' = \{(x, y) \mid \text{for some } (a, b) \in Red,\ g'(a) = x,\ g'(b) = y\}$$

View Red' as a relation on $N_{G'}$, writing $Red'+$, $Red'*$ for its transitive, reflexive transitive closures.

We write $x \sim y$ iff there is a $z \in N_{G'}$ such that $x\ Red'*\ z$ and $y\ Red'*\ z$. Then $\sim$ is clearly an equivalence relation, because Red is a partial function on P and because (RED-3) ensures g and hence g' is injective on the LHS nodes of redirections. We write $[x]$ to represent the equivalence class containing x as usual. Further, we will write $[x]^\circ$ iff there is a $y \in [x]$ such that $y\ Red'{+}\ y$ (i.e. we write $[x]^{\circ}$ to indicate that $[x]$ contains a non-trivial *Red′*-cycle). We write $[x]^{-}$ otherwise.

Lemma 3.5 With the preceding notation, for all $[x]^{-}$ there is a unique $y^{-} \in [x]^{-}$ such that for all $x \in [x]^{-}$, $x\ Red'{*}\ y^{-}$.

Proof. Basically trivial once one notes that all rules are finite objects, whence each $[x]^{-}$ equivalence class is a tree in $N_{G'}$ and has a unique root y^{-}. ☺

When the context makes the class $[z]^{-}$ clear, we will use the $^{-}$ notation to refer to this root element without further comment.

Redirection takes each arc (p_k, c) such that $c = g'(a)$ for some $(a, b) \in Red$ and replaces it with (p_k, d), where d is determined by interpreting the redirections in *Red* transitively. Thus if c is a member of a $[–]^{-}$ class, $[x^{-}]^{-}$ say, then d is the unique root element x^{-} of that class. Otherwise c is in a $[–]^{\circ}$ class, and d is a freshly introduced ⊥-node for the class.

Performing the redirections on our example yields Fig. 6. Note the new ⊥-node, introduced because of the self-redirection on Var.

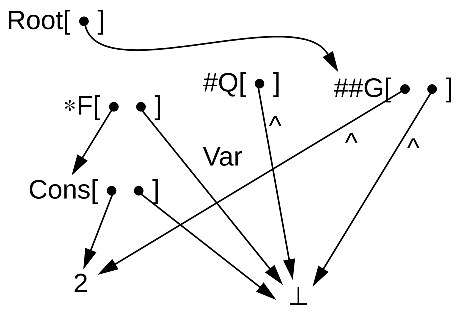

Fig. 6 Redirection.

More formally we have the following.

Definition 3.6 Assume the preceding notation. Let the graph G'' be given by

(1) $N_{G''} = N_{G'} \uplus B$ where $B = \{[x]^{\circ} \mid [x]^{\circ} \subseteq N_{G'}\}$.

(2) $\sigma_{G''}(x) = \sigma_{G'}(x)$ if $x \in N_{G'}$,
$\perp$ if $x \in B$.

(3) $\alpha_{G''}(x)[k] = y^{-} \in N_{G'}$ if $\alpha_{G'}(x)[k] = y$ and $y \in [y^{-}]^{-}$, for $k \in A(x)$,
$[y]^{\circ} \in B$ if $\alpha_{G'}(x)[k] = y$ and $y \in [y]^{\circ}$, for $k \in A(x)$,
$\alpha_{G'}(x)[k]$ for $k \in A(x)$, otherwise.

(4) $\mu_{G''}(x) = \mu_{G'}(x)$ if $x \in N_{G'}$,
ε if $x \in B$.

(5) $\nu_{G''}(x) = \nu_{G'}(x)$ if $x \in N_{G'}$,
$\varnothing$ if $x \in B$.

On G'' the map g' induces a map $g'' : P \rightarrow G''$ which is now just a symbol preserving node map, rather than a homomorphism. Furthermore, there is an obvious injection $i_{G',G''} : G' \rightarrow G''$ that takes each node of G' to its representative in G'' as we discussed above. We also define the map $r_{G',G''} : G' \rightarrow G''$ which identifies redirection targets as follows.

$$
\begin{aligned}
r_{G',G''}(x) \; = \; & x^{-} \in N_{G'} \text{ if } x \in [x^{-}]^{-}, \\
& [x]^{\circ} \in B \text{ if } x \in [x]^{\circ}, \\
& i_{G',G''}(x), \text{ otherwise.}
\end{aligned}
$$

The above is the only place where the $i_{\_,\_}$ and $r_{\_,\_}$ maps differ.

Activation makes active the $(r_{G',G''} \circ g')$-images of idle non-$\perp$-nodes in *Act*, and also makes the g''-image of t idle (root quiescence). Doing this for our running example yields Fig. 7. Note that despite there being two activations in the rule (the two children of Cons), only one is performed as the second of Cons's children is a $\perp$-node.

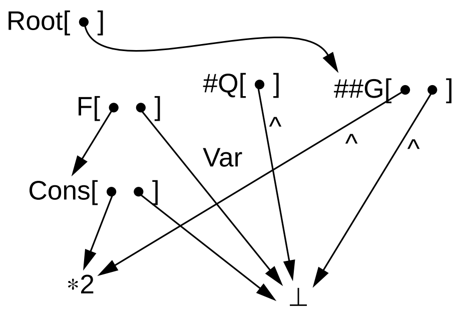

Fig. 7 Activation.

More formally we have the following.

Definition 3.7 Assume the preceding notation. Let the graph H be given by

(1) $N_H = N_{G''}$.

(2) $\sigma_H = \sigma_{G''}$.

(3) $\alpha_H = \alpha_{G''}$.

(4) $\mu_H(x) = *$ if $\exists\, a \in Act$ with $(r_{G',G''} \circ g')(a) = x$ and $\sigma_{G''}(x) \neq \perp$ and $\mu_{G''}(x) = \varepsilon$,
ε if $x = t$,
$\mu_{G''}(x)$ otherwise.

(5) $\nu_H = \nu_{G''}$.

One can easily check that by **M-I.11.4**.(8), quoted above, and definition 3.6, the first two clauses of definition 3.7.(4) are always disjoint.

On H, g'' induces a map $h : P \to H$ which is of course another symbol preserving node map. We also define the maps $i_{G'',H} = r_{G'',H} : G'' \to H$ as the identity on nodes by analogy with previous cases.

Definition 3.8 The result of the rewrite of the redex $g : L \to G$ according to the rule $D = (P, root, Red, Act)$ is the graph H produced by applying definitions 3.4 – 3.7.

By composing the various maps $i_{G,G'}$, $i_{G',G''}$ or $r_{G',G''}$, etc., we can track the history of a node through a rewrite. Thus $i_{G,H}(x) = (i_{G'',H} \circ i_{G',G''} \circ i_{G,G'})(x)$ is the node which is the copy in H of $x \in G$, and $r_{G,H}(x) = (r_{G'',H} \circ r_{G',G''} \circ r_{G,G'})(x)$ is the node of H that x got redirected to. In future we will need to keep a close track of nodes through the phases of a rewrite, particularly when relevant properties of nodes change from one phase of a rewrite to another, so the above notation is a useful alternative to the g, g', g'', h maps, and is also applicable to nodes not directly affected by the particular rewrite.

Composing a sequence of $i_{G,H}$ maps or of $r_{G,H}$ maps, allows us to track the history of a node through an execution of the system. The former tracks a node's identity, and the latter tracks what a node "becomes" via redirection. Generically, any such composition will be called $i_{X,Y}$ or $r_{X,Y}$ where X and Y are the first and last graphs in the sequence. An arc (p_k, c) is evidently tracked by $(i_{X,Y}(p)_k, r_{X,Y}(c))$. A last but very important property of these notations is that they are portable to situations in which one wishes to define operations on graphs "universally", i.e. up to (marking preserving) isomorphism. In such approaches one defines the semantics by listing properties that the collection of functions g, g', g'', h, $i_{G,G'}$, $i_{G',G''}$, $r_{G',G''}$, etc. possesses, and any graph H^* related to the input data of the rewrite (or other execution step) by such a collection of maps is an acceptable answer. Of course the list of properties must be such that any H^* satisfying them is guaranteed to be marking preserving isomorphic to H as we constructed it above. We will need some of this below, when we have to say precisely in what manner the two graphs at the ends of the two paths round a cell in the Church-Rosser diamond are "the same".

The above operational semantics is a bit complicated, to say the least. The reasons for this arise from the desire to make the implementation on a certain kind of architectural model relatively straightforward, and thus efficient. In reality, the model divides into the graph structure part (which is intended to encode the actual computation), and the markings, which guide the sheduling policy of any implementation (via activations, suspensions and notifications, and definition 3.1). Thus the essence of the rewriting process is redirection, a digraph version of substitution. Since the RHS of such a "substitution" must connect with the rest of the graph in general, it is more convenient to insist that contractum building comes first. Activations then allow the execution of a rule to influence future rewriting strategy. The specific versions of activations, suspensions, and redirections used in this paper (compared with those of other papers in this series) ensure both a good serialisability theory (see **M-IV**), and a clean Church-Rosser theorem, properties which are not altogether unrelated.

4 Fundamental Properties — Balancedness and State Saturatedness

In this section we treat two rather basic invariants in the context of our operational semantics.

Definition 4.1 A node x in a graph G is balanced iff for $n \geq 1$,

$$\mu(x) = \#^n \;\Leftrightarrow\; |\{k \mid \nu(x)[k] = \text{\textasciicircum}\}| = n$$

We say that a pattern or graph is balanced iff every node is balanced.

Theorem 4.2 Let R be a MONSTR system. Then every execution graph of R is balanced.

Proof. By induction on executions. An initial graph is balanced. Furthermore, notifications preserve balancedness, since for each notification marking removed from an arc, a suspension marking is removed from the parent node. Suspensions preserve balancedness by doing the opposite. We check that the phases of a rewrite do not affect balancedness. Contractum building preserves balancedness, as all new nodes added to the graph are balanced by restriction **M-I.11.4**.(6) quoted above. Redirection only affects the heads of some arcs and introduces balanced $\perp$-nodes, so preserves balancedness. Finally, activation only affects the node markings on non-suspended nodes, thus preserving balancedness. ☺

Definition 4.3 An arc (p_k, c) of a graph G is state saturated iff

$$\nu(p)[k] = \text{\textasciicircum} \text{ and } \mu(c) = \varepsilon \;\Rightarrow\; \sigma(c) \in \mathbf{V} \cup \{\perp\}$$

A node of a graph is state saturated iff all of its in-arcs are state saturated. Likewise, a graph or pattern is state saturated if all of its nodes and arcs are.

Theorem 4.4 Let R be a MONSTR system. Then every execution graph of R is state saturated.

Proof. By induction over executions. An initial graph is state saturated. A notification step clearly preserves state saturatedness, since for the only node which becomes idle, all notification in-arcs become idle unless the node is a stateholder, in which case certain of them are allowed to remain as notification in-arcs according to definition 3.2. A suspension step does the opposite, creating notification arcs, but where these have idle nodes as children, then such children are always stateholders or $\perp$-nodes by inspection of definition 3.3, preserving state saturatedness. We argue that rewrites preserve state saturatedness as follows.

Let G_i be rewritten to G_{i+1}, using a rule $D = (P, \mathit{root}, \mathit{Red}, \mathit{Act})$, and a redex $g_i : L \to G_i$. Assume the usual notation for the pieces of a rewrite (eg. maps g_i, g_i', g_i'', and $g_{i+1} : P \to G_{i+1}$). Consider contractum building. It is easy to check that all new nodes introduced in G'_i are state saturated by restriction **M-I.11.4**.(7) since $\mathit{Act} \subseteq L$. Obviously the nodes of $G_i' - g_i'(P)$ are state saturated since they continue to have ($i_{G_i,G_i'}$ copies of) just the same arcs they had in G_i, and G_i is state saturated by the induction hypothesis. This leaves the nodes of $g_i'(L)$. Nodes in $g_i'(L) - g_i'(\mathit{Act})$ are state saturated because any new in-arcs they acquired are state saturated by **M-I.11.4**.(7). This leaves a set of nodes $g_i'(\chi) \subseteq g_i'(\mathit{Act}) \subseteq g_i'(L) \subseteq G_i'$ which fail to be state saturated as they acquired a non-zero number of notification in-arcs during contractum building, but were idle, not $\perp$-

nodes, not **V**-labelled, and without notification in-arcs in G_i. Therefore G_i' may fail to be state saturated, but just for this reason.

Now consider redirection and activation. All arcs of G_{i+1} are copies, or redirected copies of arcs of G_i'. We check that all arcs of G_i' end up state saturated in G_{i+1} which is sufficient. Leaving aside the phenomenon of root quiescence for the moment, there are three cases. Case (a): in-arcs of nodes in $g_i'(\chi) \subseteq g_i'(Act)$ which are not redirected. These are unchanged by redirection, and have their child nodes activated during activation, restoring state saturatedness to case (a) nodes. Case (b): in-arcs of nodes y which are redirected. Let (x_k, y) be a redirected arc, with $g_i'(a) = y$, $(a, b) \in Red$, and redirection target $r_{G_i',G_i''}(y)$. If $\sigma(r_{G_i',G_i''}(y)) \in \mathbf{V}$ or $\mu(r_{G_i',G_i''}(y)) \neq \varepsilon$, or $r_{G_i',G_i''}(y)$ is a $\perp$-node, then $(i_{G_i',G_i''}(x)_k, r_{G_i',G_i''}(y))$ is state saturated. In case not, we know $b \in Act$ by restriction **M-I.11.4**.(9). Now $r_{G_i',G_i''}(y) = r_{G_i',G_i''}(g'(b))$ by definition of redirection, and we are assuming that $\mu(r_{G_i',G_i''}(g'(b))) = \varepsilon$. Therefore the activation phase, making G_{i+1}, will make $\mu(r_{G_i',G_{i+1}}(y)) = \mu(r_{G_i',G_{i+1}}(g'(b))) = *$ by definition 3.7.(4). This restores state saturatedness to all case (b) nodes. Case (c): in-arcs of all other nodes. These are state saturated in G_i' and do not suffer redirection. They remain state saturated throughout the redirection and activation phases.

Finally we return to the root, to deduce that root quiescence cannot destroy state saturatedness. But this is immediate since $g_i''(root)$ has no in-arcs. This in turn holds by restriction **M-I.11.4**.(8) and because no LHS of a redirection is a redirection target (i.e. the destination of a redirected arc) by definition 3.6. We are done. ☺

5 Liveness and Garbage

In all of the preceding, no node or arc was ever destroyed, which is not really acceptable for a reasonable model of computation. In this section we introduce a suitable notion of liveness, which turns out to be a proof theoretic business. This leads to the appropriate notion of garbage which we prove sound.

Definition 5.1 Let G be a graph, and x a node of G. Then x is live iff it can be proved so on the basis of the following rules of inference:

(1) If $\sigma(x)$ is a special symbol Root, then x is live.

(2) If $\mu(x) = *$, then x is live.

(3) If p is live and (p_k, x) is an idle arc, then x is live.

(4) If c is live and (x_k, c) is a notification arc, then x is live.

Definition 5.1 is nothing more than a proof system. Thus clauses (1) and (2) form base cases of proofs of liveness; and liveness is propagated down normal arcs and up notification arcs, which makes clauses (3) and (4) into analogues of modus ponens. It is evident that the structure of a proof of liveness follows the structure of a certain kind of semipath in the graph.

Definition 5.2 Let G be a graph. The set of live nodes of G is denoted Live(G), and N_G – Live(G) is denoted Gar(G), the garbage set of G. An arc (p_k, c) of G is live iff both p and c are live; otherwise it is garbage.

Note that the inference rules in definition 5.1.(3) and 5.1.(4) give “local” means of proving the liveness of any given live arc (p_k, c). Connectivity properties of the graph G may also give rise to other, completely unrelated proofs for (p_k, c).

Definition 5.3 The live subgraph of a graph G, LSG(G), consists of the live nodes and live arcs of G.

Note that the live subgraph need not be a graph in the sense that it satisfies all the invariants implied by definition 2.1, since a live node may have a garbage notification out-arc to a garbage child node. Live nodes may also have garbage normal in-arcs from garbage parent nodes, though this does not threaten the invariants of definition 2.1.

The most important thing about garbage is its persistence. Once a node of an execution graph is proclaimed garbage, no execution step should cause it to be capable of being proved live ever again. This is the main result of this section.

Theorem 5.4 Let $\mathcal{R}$ be a MONSTR system. Let G be an execution graph of $\mathcal{R}$, and let $G \rightarrow H$ be an execution step. Then

(1) If x is a garbage node of G, then $i_{G,H}(x)$ is a garbage node of H.

(2) If (p_k, c) is a garbage arc of G, then $(i_{G,H}(p)_k, r_{G,H}(c))$ is a garbage arc of H.

Proof. We must check that each possible execution step does not involve any garbage node or arc in any harmful way. In order, we examine notifications, suspensions, and finally rewrites. For each execution step we define the *execution step redex* to consist of all nodes and arcs mentioned in the definition of execution steps of that kind in Section 3.

In all three cases, the structure of the proof is the same. First of all we prove that the execution step redex is live. Then we identify the redex-emergent arcs as those arcs, precisely one of whose nodes is in the execution step redex, and which are capable of progressing a liveness proof out of the redex. Fig. 8 shows how this happens in sequent notation. Finally we show that the transformation that generates the graph H cannot make any hypothesised garbage node x live, by reasoning about the properties of the redex-emergent arcs. The proof for garbage arcs (p_k, c) then follows quickly.

$$\dfrac{\dfrac{\cdots\ \cdots}{z\ \text{live}} \qquad (z_k, q) : \text{norm. arc}}{\dfrac{q\ \text{live}}{\cdots\ \cdots}}$$

Normal arc case

$$\dfrac{\dfrac{\cdots\ \cdots}{z\ \text{live}} \qquad (q_k, z) : \text{notif. arc}}{\dfrac{q\ \text{live}}{\cdots\ \cdots}}$$

Notification arc case

Fig. 8 Redex-emergent steps in proofs of liveness;
z is live because it is in the execution step redex.

Notifications. For a notification from $t \in G$, the notification redex consists of all notification arcs (z_l, t) and their constituent nodes manipulated by the notification, i.e. all notification in-arcs (z_l, t) of t such that t is not a stateholder child in (Map – State) position of the function parent node z. The redex-emergent arcs are therefore: normal arcs (t_m, q); other notification in-arcs (w_m, t) of t (such that t *is* a stateholder child in (Map – State) position of the function parent node w); normal arcs (z_m, q) (for the relevant nodes z); and notification arcs (q_m, z) (for the same relevant nodes z). All such arcs are evidently live in G since t is active in G.

We recall that for notifications, $r_{G,H} = i_{G,H}$. In H, the $i_{G,H}$ image of t is idle, and the $i_{G,H}$ images of all arcs (z_l, t) are normal. The $i_{G,H}$ images of redex-emergent arcs (z_m, q) and (q_m, z) are live iff their corresponding z is live, and the $i_{G,H}$ images of redex-emergent arcs (t_m, q) and (w_m, t) are live iff t is live.

Suppose now that x is garbage in G, but that $i_{G,H}(x)$ is live in H. Obviously x cannot be in the notification redex. Consider a proof of the liveness of $i_{G,H}(x)$, in order to construct one for x, for a contradiction. The proof in H starts at an active or Root-labelled node of H, say u_0. But by the definition of notifications, $u_0 = i_{G,H}(u_0{}^*)$ for some likewise active or Root-labelled node $u_0{}^*$ of G. Evidently $u_0{}^* \neq x$, so the proof must be bigger than just an axiom instance. It therefore continues along either a normal or notification arc. If the arc is a notification arc, it is the $i_{G,H}$ image of a notification arc of G, and we continue the proof in G. If it is a normal arc, then either it is the $i_{G,H}$ image of a normal arc of G and the proof in G continues, or it is the $i_{G,H}$ image of a notification arc of G. In the latter case we are dealing with an arc of the notification redex whose child node must be t in G, which was active and thus live in G. Since x is not in the notification redex in G, the proof emerges from the notification redex along a redex-emergent arc. Therefore we can construct a proof of the liveness of x in G by patching the tail of a proof of the liveness of $i_{G,H}(x)$ in H, the tail in question being from the last (if any) visit to the $i_{G,H}$ image of the notification redex in the proof for H.

So we have our contradiction and $i_{G,H}(x)$ is garbage in H. For a garbage arc (p_k, c), we argue that at least one of p or c is garbage and thus outside the notification redex in G. By the preceding, its $i_{G,H}$ image is still garbage in H. If p is the garbage node, then $(i_{G,H}(p)_k, r_{G,H}(c))$ is obviously garbage. If c is the garbage node, then because $r_{G,H}(c) = i_{G,H}(c)$ for notifications, $r_{G,H}(c)$ is garbage in H, giving the conclusion.

<u>Suspensions</u>. Let Susp(t) be given as in definition 3.3 and let $0 \neq n = |\ \mathrm{Susp}(t)\ |$. The suspension redex consists of all arcs (t_l, z) of G, with $l \in$ Susp(t), and their constituent nodes. Since t is active and all suspension redex arcs are normal by balancedness, the suspension redex is live in G. The redex-emergent arcs are notification arcs (q_m, t); all normal arcs (t_m, q) for $m \notin$ Susp(t); normal arcs (z_m, q) where $z = \alpha(t)[l]$ for $l \in$ Susp(t); and finally notification arcs (q_m, z). Before the suspension step all of these arcs are live.

Suppose now that x is garbage in G, but that $i_{G,H}(x)$ is live in H. Obviously x cannot be in the suspension redex. As in the notification case, consider a proof of the liveness of $i_{G,H}(x)$, in order to construct one for x. The proof in H starts at an active or Root-labelled node of H, say u_0. But by the definition of suspensions, $u_0 = i_{G,H}(u_0{}^*)$ for some likewise active or Root-labelled node $u_0{}^*$ of G. So $u_0{}^* \neq x$, and the proof must be bigger than just an axiom instance. It therefore continues along either a normal or notification arc. If the arc is a normal arc, it is the $i_{G,H}$ image of a normal arc of G, and we continue the proof in G. If it is a notification arc, then either it is the $i_{G,H}$ image of a notification arc of G and the proof in G continues, or it is the $i_{G,H}$ image of a normal arc of G. In the latter case we are dealing with an arc of the suspension redex, whose parent node must be t in G, which was active and thus live in G. Since x is not in the notification redex in G, the proof emerges from the notification redex along a redex-emergent arc. Therefore we can again construct a proof of the liveness of x in G by patching the tail of a proof of the liveness of $i_{G,H}(x)$ in H, the tail in question being from the last (if any) visit to the $i_{G,H}$ image of the suspension redex in the proof for H. The proof for garbage arcs is as in the case of notifications.

Rewrites. Employing the usual notation, for a rewrite step, the redex is $g(L)$ as per definition 2.5. An important consequence of balancedness and of definition 3.1, is that for a rewrite, all arcs of $g(L)$ are normal. Therefore the whole of $g(L)$ is live. The redex-emergent arcs are notification arcs (q_k, z) with $z \in g(L)$, and normal arcs (z_k, q) with $z \in g(L)$ and $q \notin g(L)$.

Consider the garbage node x in G. There is no proof of liveness of x in G so $x \notin g(L)$. After contractum building, all proofs of liveness in G remain valid after being mapped to G' because of the injection $i_{G,G'}$ which preserves markings. New proofs of liveness may have been created involving the contractum nodes, but none of them can prove $i_{G,G'}(x)$ live. For suppose not. To do so such a proof would have to follow a semipath from a contractum node to $i_{G,G'}(x)$. Since such a semipath must pass through $g'(L)$, we would have a redex-emergent step in the proof. Since all redex-emergent steps are unchanged from G, we could patch the final part of such a proof in H to construct a proof of the liveness of x in G, a contradiction.

After redirection, all previous proofs not mentioning redex or contractum nodes remain unchanged, since for arcs not containing a redex or contractum node, $i_{G',G''}$ extends to a marking-preserving homomorphism. Call $g''(P) \uplus B$ (where B is the set of $\perp$-nodes adjoined during the redirection phase) the extended redex for brevity. By the previous paragraph, any proof of liveness in G'' of $i_{G,G''}(x)$ must involve an extended-redex-emergent step. There are two cases.

The Normal Case: Here we note that a normal arc (z_k, q) with z in the extended redex must have z in $g''(L)$ since redirection does not affect the parent nodes of arcs. Thus as before, the final part of the proof would correspond with the final part of a proof in G', and $i_{G,G'}(x)$ would be live in G', a contradiction.

The Notification Case: Here we note that a notification arc (q_k, z) in G'' is the $(i_{G',G''}, r_{G',G''})$ image of a notification arc (q_k, z^*) of G'. If z^* did not get redirected, then z^* is in $g'(L)$ as this is the only part of the extended redex in G' accessible from outside $g'(P)$. But then $g'(L)$ is live, and so the final part of the proof would correspond with the final part of a proof in G' and $i_{G,G'}(x)$ would be live in G'. If z^* did get redirected to z, then z^* is in $g'(L)$ since LHSs of all redirections are. Once more we would find a proof with a redex-emergent step involving (q_k, z^*), showing that $i_{G,G'}(x)$ was live in G'. We conclude that the $i_{G,G''}$ image of x remains garbage in G''.

Finally the root quiescence and activation phase. The root is always made idle according to definition 3.7.(4). Thus since $h(root)$ is idle in H, some proofs of liveness that exist for G'' are destroyed; which cannot make $i_{G,H}(x)$ live. If some nodes of $h(P)$ are activated, some new proofs of liveness in H without counterparts in G'' might be created. However, any such proof which proved $i_{G,H}(x)$ live, must utilise (the $i_{G'',H}$ image of) an extended-redex-emergent step, as argued above. Any such extended-redex-emergent step involves either a normal arc, or a notification arc, and the arguments for these cases are identical to those voiced above for the redirection phase. We conclude that x is garbage in H.

For a garbage arc (p_k, c), we argue that at least one of p or c is garbage and thus outside of $g(L)$ in G. By the preceding, its $i_{G,H}$ image is still garbage in H. If p is the garbage node, then $(i_{G,H}(p)_k, r_{G,H}(c))$ is obviously garbage. If c is the garbage node, then because c is outside of $g(L)$, $r_{G,H}(c) = i_{G,H}(c)$, the latter of which is garbage in H, giving the conclusion. We are done. ☺

We end this section with a simple lemma whose proof is largely implicit in the preceding proof.

Lemma 5.5 Let $\mathcal{R}$ be a MONSTR system. Let G be an execution graph of $\mathcal{R}$, and let $G \rightarrow H$ be a rewrite execution step according to a rule $D = (P, root, Red, Act)$ of a redex $g(L)$ rooted at $t \in G$. Let $x \in g(L)$ be the left node of a redirection prescribed by the rewrite. Then

(1) $i_{G,H}(x)$ has no in-arcs.

(2) $i_{G,H}(x)$ is garbage in H.

Proof. We know that each node that is the left node of a redirection is in either a $[–]^{-}$ class, or in a $[–]^{\circ}$ class. In the first case the redirection target is the $i_{G',G''}$ image of a node of G' that is not itself the left hand side of a redirection. In the second case the redirection target is a $\perp$-node, again not the left hand side of a redirection. We deduce that (1) holds.

To get (2), we note that from the MONSTR restrictions on rules, x is either the root of the redex, or the stateholder child of the root. Neither of these are constructors, therefore $\sigma(x) \neq$ Root, since Root is a constructor by **M-I.11.2**.(4) quoted above, and so $i_{G,H}(x)$ cannot be proved live by recourse to definition 5.1.(1). Also the root is quiesced, and the stateholder, if redirected, is not activated since the nodes which are activated are $(r_{G',G''} \circ g')(Act)$, which does not include any left nodes of redirections by the previous paragraph, so $\mu(i_{G,H}(x)) \neq *$, and $i_{G,H}(x)$ cannot be proved live by definition 5.1.(2). By (1), $i_{G,H}(x)$ has no in-arcs of any kind, so $i_{G,H}(x)$ cannot be proved live by definition 5.1.(3). Finally, the root was quiesced, so is idle in H. Furthermore, the stateholder was in G, idle by definition 3.1, therefore if redirected, remains unactivated in H by our preceding argument, and thus idle in H. By balancedness, $i_{G,H}(x)$ has no notification out-arcs, so $i_{G,H}(x)$ cannot be proved live by definition 5.1.(4). We are done. ☺

6 Overlapping Redexes and Safe Critical Cones

In this section, we define resuspending rules, and discuss critical cones, particularly safe critical cones.

Definition 6.1 Let $F \in \mathsf{F}$ and $S \in \mathsf{V}$. Let $k_f \in$ State(F). A normal rule $D = (P, root, Red, Act)$ for F is a resuspending rule for F and S iff

(1) F matches S (in k_f'th position).

(2) The only contractum node of P is a node f such that

$\sigma(f) = F$,

$\mu(f) = \#$,

For $j \in A(f)$,

$$\alpha(f)[j] = \alpha(root)[j],$$

$$\nu(f)[j] = \textbf{If } \ j = k_f \ \textbf{ Then } \ \wedge \ \textbf{ Else } \ \varepsilon$$

(3) $Red = \{(root, f)\}$

(4) $Act = \varnothing$

Example 6.2 Let A(F) = {1 … 5}, Map(F) = {2, 3, 4}, and State(F) = {3}. Then Fig. 9 shows a resuspending rule for F and S.

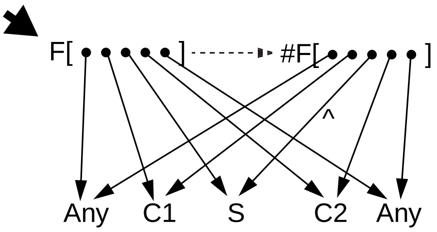

Fig. 9 A resuspending rule.

Definition 6.3 Let G be an execution graph of a MONSTR system R, and let $s \in G$ be a stateholder node of G, and let $S = \sigma(s)$. Let CC(s) contain s and all the active function nodes $f \in G$ such that s occurs in stateholder position of each f, i.e.

$$\mathrm{CC}(s) = \{\ f \in G \mid \mu(f) = *,\ \sigma(f) \in \mathbf{F},\ \exists\ k_f \in \mathrm{State}(\sigma(f)),\ \text{and } s = \alpha(f)[k_f]\ \} \cup \{s\}$$

We call CC(s), the critical cone of s, and we call the arcs (f_{k_f}, s), the arcs of the critical cone. We can write a critical cone using the notation $\{s; f_1 \dots f_n\}$.

When we strive for a Church-Rosser property, we can allow redexes to overlap on constructors, as these are read-only. Likewise we can contemplate the idea of allowing redexes to overlap on their implicitly matched nodes, since these cannot be redirected. Because the operational semantics is transitive and coercing, this turns out to be sound, though non-trivial (see the next section and the one that follows). This concentrates the focus on critical cones, as the seat of non-confluent behaviour. Safe critical cones are those where we can see that non-confluent behaviour can be avoided.

Definition 6.4 Let G be an execution graph of a MONSTR system R, and let $s \in G$ be a stateholder with critical cone CC(s) = $\{s; f_1 \dots f_n\}$. The critical cone is safe iff one of the following conditions holds.

(1) For $i = 1 \dots n$, no rule for $\sigma(f_i)$ which matches at $f_i \in G$, redirects its stateholder argument.

(2) There is exactly one $i \in \{1 \dots n\}$ such that there is a rule for $\sigma(f_i)$ which matches at $f_i \in G$ and redirects its stateholder argument. Furthermore it redirects it to a non-idle node or activated node. For $i \neq j \in \{1 \dots n\}$, the only normal rules for $\sigma(f_j)$ which match at f_j are resuspending rules, and each $\sigma(f_j)$ has such a rule.

The idea behind these possibilities should be clear. In (1), the stateholder behaves as just another constructor. In (2), precisely one of the functions is capable of "doing anything"; the others, should any of them rewrite, merely resuspend, which modulo markings and garbage does not alter the structure of the graph. A cone of type (1) can give rise to one of type (2) on the same stateholder, as once all the "read only" functions have rewritten, other rewriting activity containing references to (a suitable $i_{\_,\_}$ image of) s, may create functions having references to (a suitable image of) s in stateholder position.

Despite these promising features, the critical cones we have described would not be interesting were it not for the fact that the resuspending behaviour we focus on is exectly what realistic MONSTR systems typically use for synchronisation purposes. (See eg. the references to applications discussed in **M-I**.)

7 Subcommutativity Lemmas

In this section, we present the basic subcommutativity lemmas that hold for MONSTR systems under transitive coercing semantics. These are the building blocks for the main theorem of the paper.

Lemma 7.1 Let $\mathsf{G}_{\mathrm{N}} = [G_0, \ldots, G_{\mathrm{N}}]$ be a transitive coercing preexecution of a MONSTR system R. Suppose G_{N} contains two active nodes $t_1 \neq t_2$ with $\{\sigma(t_1), \sigma(t_2)\} \subseteq \mathbf{C} \cup \mathbf{V}$. For either choice of $i \in \{1, 2\}$, let j denote the other choice. Let H_i be obtained by performing a notification from t_i in G_{N}. Then

(1) H_1 and H_2 are graph structure isomorphic.

(2) $r_{G_{\mathrm{N}},H_i}(t_j) = i_{G_{\mathrm{N}},H_i}(t_j)$ is an active constructor or stateholder, hence the root of a potential notification step, in H_i.

Let K_i be obtained from H_i by notifying from $r_{G_{\mathrm{N}},H_i}(t_j)$. Then

(3) K_1 and K_2 are marking preserving isomorphic via a map $\psi : K_1 \to K_2$.

Proof. This is relatively easy. Since notifications merely manipulate markings, (1) follows immediately since both H_1 and H_2 are graph structure isomorphic to G_{N}. Since t_j is active in G_{N}, it cannot be a suspended parent of a notification arc of t_i; thus it is not notified in t_i's notification, and $r_{G_{\mathrm{N}},H_i}(t_j)$ is active in H_i so that (2) holds. As for (1), K_1 and K_2 are graph structure isomorphic, so we must check that the markings coincide. We know that the sets of notification arcs that comprise the notification redexes of t_1 and t_2 in G_{N} are disjoint. After notification, all of them end up as normal arcs in K_1 and K_2. Other arcs are unaffected.

For nodes, t_1 and t_2 lose their active marking; nodes not in either notification redex keep their marking; parent nodes of t_i in the notification redex of t_i but not of t_j decrement their suspensions by the same amount during the notification of either t_i or of $r_{G_{\mathrm{N}},H_j}(t_i)$; and parent nodes of both t_1 and t_2 in both notification redexes decrement their suspensions by the sum of two such amounts, ending with the same marking since $(n - a) - b = (n - b) - a$. So we have (3), and thus the whole lemma. ☺

Lemma 7.2 Let $\mathsf{G}_{\mathrm{N}} = [G_0, \ldots, G_{\mathrm{N}}]$ be a transitive coercing preexecution of a MONSTR system R. Suppose G_{N} contains two active function nodes $s_1 \neq s_2$ with $\mathrm{Susp}(s_1) \neq \varnothing \neq \mathrm{Susp}(s_2)$, where the Susp set of a function node is given in definition 3.3. For either choice of $i \in \{1, 2\}$, let j denote the other choice. Let H_i be obtained by performing a suspension from s_i in $G_{\rm N}$. Then

(1) H_1 and H_2 are graph structure isomorphic.

(2) $r_{G_{\rm N},H_i}(s_j) = i_{G_{\rm N},H_i}(s_j)$ is an active function node, and hence the root of a potential suspension step, in H_i.

Let K_i be obtained from H_i by performing a suspension from $r_{G_{\rm N},H_i}(s_j)$. Then

(3) K_1 and K_2 are marking preserving isomorphic via a map $\psi : K_1 \rightarrow K_2$.

Proof. This is pretty similar to lemma 7.1, in that notifications turn notification arcs into normal arcs, while suspensions turn normal arcs into notification arcs. So we will be fairly brief.

Since suspensions merely manipulate markings we have (1) immediately. Also since the only node markings that change in a suspension step are those of the suspension root and of any activated idle functions, and all nodes are idle afterwards only if they were idle before, (2) follows, and K_1 and K_2 are graph structure isomorphic. Since the sets of normal arcs constituting the two suspension redexes are disjoint in $G_{\rm N}$, and any idle function that is activated ends up activated regardless of the order of the suspensions, we get (3) easily. ☺

Lemma 7.3 Let $\mathcal{G}_{\rm N} = [G_0, \ldots, G_{\rm N}]$ be a transitive coercing preexecution of a MONSTR system $\mathcal{R}$. Suppose $G_{\rm N}$ contains an active function node s with $\mathrm{Susp}(s) \neq \emptyset$, where the Susp set of a function node is given in definition 3.3. Suppose $G_{\rm N}$ also contains an active constructor or stateholder t. Let

$$
\begin{array}{l}
\mathrm{Susp} = \{k \in \mathrm{Map}(\sigma(s)) \mid \alpha(s)[k] \text{ is non-idle, or} \\
\qquad\qquad\qquad\qquad\qquad\qquad \alpha(s)[k] \text{ is idle and } \sigma(\alpha(s)[k]) = \bot, \text{ or} \\
\qquad\qquad\qquad\qquad\qquad\qquad \alpha(s)[k] \text{ is idle and } \sigma(\alpha(s)[k]) \in \mathbf{F}, \text{ or} \\
\qquad\qquad\qquad\qquad\qquad\qquad [\ k \in (\mathrm{Map}(\sigma(s)) - \mathrm{State}(\sigma(s))) \text{ and} \\
\qquad\qquad\qquad\qquad\qquad\qquad \alpha(s)[k] \text{ is idle and } \sigma(\alpha(s)[k]) \in \mathbf{V}\ ]\ \} \\
\mathrm{SuspNodes} = \{x \in G_{\rm N} \mid x = \alpha(s)[k] \text{ for some } k \in \mathrm{Susp}\} \\
\Pi \ \equiv\ \mathrm{SuspNodes} = \{t\} \text{ and not}[\ \sigma(t) \in \mathbf{V} \text{ and } t = \alpha(s)[k] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{for some } k \in (\mathrm{Map}(\sigma(s)) - \mathrm{State}(\sigma(s)))\ ]
\end{array}
$$

Let H_s be obtained by performing a suspension from s in $G_{\rm N}$, and let H_t be obtained by performing a notification from t in $G_{\rm N}$. Then

(1) H_s and H_t are graph structure isomorphic.

(2) (a) $r_{G_{\rm N},H_s}(t) = i_{G_{\rm N},H_s}(t)$ is an active constructor or stateholder, hence the root of a potential notification step, in H_s.

(b) $r_{G_{\rm N},H_t}(s) = i_{G_{\rm N},H_t}(s)$ is an active function node, and unless Π holds, is the root of a potential suspension step, in H_t.

Let K_s be obtained from H_s by performing a notification from $r_{G_{\rm N},H_s}(t)$, and let

K_t = **If** Π **Then** H_t
Else The result of performing a suspension from $r_{G_{\rm N},H_t}(s)$ in H_t

Then

(3) K_s and K_t are marking preserving isomorphic via a map $\psi : K_s \to K_t$.

Proof. As in the previous lemmas, (1) is immediate. Since s cannot be in the notification redex of t in G_{N}, and since although t might be in the suspension redex of s in G_{N}, the node markings of non-idle non-root nodes of suspension redexes do not change during suspensions, we conclude (2), noting that if t was the only element of s's SuspNodes set, unless t is a stateholder in a constructor-only position of s, there is no potential suspension from $r_{G_{\mathrm{N}},H_t}(s)$ in H_t since $r_{G_{\mathrm{N}},H_t}(t)$ is idle. Obviously we find that K_s and K_t are graph structure isomorphic, so we need to check the markings.

For arcs there are four disjoint cases: (a) all arcs (s_k, t), for any applicable k, which must all be normal arcs in G_{N}; (b) other arcs of the suspension redex; (c) arcs of the notification redex; (d) all remaining arcs.

For (a), there are two subcases: (a1) arcs (s_k, t) such that t is a stateholder in constructor-only position k of s for any such k; (a2) all other case (a) arcs. For both subcases, if the suspension is done first, the constituent arcs become notification arcs of H_s, and then the (a2) arcs become normal arcs of K_s after the notification, (a1) arcs remaining suspended. If the notification is done first, since $r_{G_{\mathrm{N}},H_t}(t)$ is idle, (a2) arcs disappear from the suspension redex in H_t. If there were (a1) arcs in the suspension redex of G_{N}, the suspension step causes them to become notification arcs. For cases (b) and (c) it is clear that they become notification arcs and normal arcs respectively regardless of the order of the steps. Also case (d) arcs are unaffected.

For nodes there are also four disjoint cases: (a) s; (b) the nodes of the notification redex; (c) nodes in the suspension redex other than case (a) and case (b) nodes; (d) all remaining nodes.

For (a), if the suspension is done first, $r_{G_{\mathrm{N}},H_s}(s)$ becomes suspended in H_s, and in the notification step receives notifications along all case (a2) arcs (if there are any). If the notification is done first, s is unaffected during notification, but becomes suspended (on potentially fewer arguments) during the subsequent suspension (if any). It is clear that the net suspension markings on $r_{G_{\mathrm{N}},K_s}(s)$ in K_s and on $r_{G_{\mathrm{N}},K_t}(s)$ in K_t are the same, as the extra suspensions when the suspension is done first, match the notifications received from case (a2) arcs in the following notification. Obviously if the suspension redex consists solely of case (a2) arcs and their nodes, then all the suspensions that s acquires when suspension is first, are released in the notification, leaving $r_{G_{\mathrm{N}},K_s}(s)$ active in K_s; corresponding to the complete removal of the suspension redex (because there are no remaining elements in the SuspNodes set of $r_{G_{\mathrm{N}},H_t}(s)$ in H_t) where notification is first, followed by a null suspension, also leaving $r_{G_{\mathrm{N}},K_t}(s)$ active in K_t. For case (b) and case (c) nodes, it is easy to see that they undergo the same net change regardless of the order of the steps; likewise case (d) nodes remain unaffected. This is enough for (3). ☺

Lemma 7.4 Let $\mathcal{G}_{\mathrm{N}} = [G_0, \ldots, G_{\mathrm{N}}]$ be a transitive coercing preexecution of a MONSTR system $\mathcal{R}$. Suppose G_{N} contains an active constructor or stateholder node t. Suppose G_{N} also contains an active function node f, all of whose $(\mathrm{Map}(\sigma(f)) - \mathrm{State}(\sigma(f)))$ arguments are idle constructors, and whose $\mathrm{State}(\sigma(f))$ argument (if any) is an idle constructor or stateholder, and which is thus the root of a redex $g : L \to G_{\mathrm{N}}$ for some rule $D = (P, root, Red, Act)$. Let

$$\Pi \;\equiv\; t \in g(Act \cup \{b \mid (a, b) \in Red,\ a \in Act,\ b \in L\})$$

Let H_t be obtained by performing a notification from t in G_{N}. Let H_f be obtained by rewriting the redex rooted at f in G_{N}, via the usual phases $g' : P \to G_{\mathrm{N}}'$, $g'' : P \to G_{\mathrm{N}}''$, $h_f : P \to H_f$, and associated i and r maps. Then

(1) (a) $r_{G_{\mathrm{N}},H_f}(t) = i_{G_{\mathrm{N}},H_f}(t)$ is an active constructor or stateholder, hence the root of a potential notification step, in H_f.

(b) $r_{G_{\mathrm{N}},H_t}(f) = i_{G_{\mathrm{N}},H_t}(f)$ is an active function node, and

$$h_t = r_{G_{\mathrm{N}},H_t} \circ g : L \to H_t$$

is a redex for D, such that all the $(\mathrm{Map}(\sigma(r_{G_{\mathrm{N}},H_t}(f))) - \mathrm{State}(\sigma(r_{G_{\mathrm{N}},H_t}(f))))$ arguments of $r_{G_{\mathrm{N}},H_t}(f)$ are idle constructors, and any $\mathrm{State}(\sigma(r_{G_{\mathrm{N}},H_t}(f)))$ argument is an idle constructor or stateholder, hence is the redex of a potential rewrite in H_t.

Let K_f be obtained from H_f by performing a notification from $r_{G_{\mathrm{N}},H_f}(t)$. Let J_t be obtained from H_t by rewriting the redex rooted at $r_{G_{\mathrm{N}},H_t}(f)$ in H_t, via the usual phases $h_t' : P \to H_t'$, $h_t'' : P \to H_t''$, $j_t : P \to J_t$, and associated i and r maps. Then

(2) **If** Π **Then** $r_{G_{\mathrm{N}},J_t}(t)$ is an active constructor or stateholder,
hence the root of a potential notification step, in J_t

Let

K_t = **If** not Π **Then** J_t
Else The result of performing a notification from $r_{G_{\mathrm{N}},J_t}(t)$ in J_t

Then

(3) K_f and K_t are marking preserving isomorphic via a map $\psi : K_f \to K_t$.

Proof. A little thought shows that neither f nor any of f's $\mathrm{Map}(\sigma(f))$ arguments can be in the notification redex, either because of the node markings or the node symbols involved. However this does not preclude the notification redex nodes from occurring as implicitly matched nodes of the rewriting redex. Because of the respective arc markings, it is clear that the sets of arcs of the two redexes are disjoint.

Consider performing the notification to create H_t. Evidently G_{N} and H_t are graph structure isomorphic. And since the only node whose active marking changes in this process is t itself, and no node becomes non-idle which was not non-idle previously, $r_{G_{\mathrm{N}},H_t}(f)$ is active in H_t and (1).(b) follows. Let us compare the rewriting processes that create H_f from G_{N} and J_t from H_t using the rule D. Let

$$\theta : G_{\mathrm{N}} \to H_t$$

be the graph structure isomorphism mentioned already. The respective contractum building phases clearly allow its extension to a graph structure isomorphism

$$\theta' : G_{\mathrm{N}}' \to H_t'$$

such that the obvious triangle involving $g' : P \to G_{\mathrm{N}}'$ and $h_t' : P \to H_t'$ commutes. Evidently the redirection phase admits a further extension to a graph structure isomorphism

$$\theta'' : G_{\mathrm{N}}'' \to H_t''$$

such that the triangle involving the node maps $g'' : P \to G_{\mathrm{N}}''$ and $h_t'' : P \to H_t''$ commutes too. Likewise the activation phase finally yields the graph structure isomorphism

$$\theta''' : H_f \to J_t$$

such that the triangle involving $h_f : P \to H_f$ and $j_t : P \to J_t$ commutes.

The definition 3.8 of rewriting shows that the only active node of the rewritten graph that ends up idle in the result, is the root of the redex. The only other nodes that can undergo a change of marking are the activated nodes which, if they start off idle, end up active. Thus we conclude that since $t \neq f$, $r_{G_{\mathrm{N}},H_f}(t)$ is active in H_f, whence we have (1).(a). To get (2) and (3), we must follow what happens to the markings of the other nodes, and to the markings of the various arcs too.

For nodes there are five disjoint cases: (a) t; (b) f; (c) nodes of the notification redex other than t; (d) contractum nodes; (e) all other nodes.

For case (a), regarding t, if rewriting is done first, we know that it is active in H_f so ends up idle in K_f after the notification. If notification is done first, then it is idle in H_t, and then either is idle in J_t if Π does not hold, or is active in J_t if Π holds, giving us (2). In the latter case, we know that $r_{G_{\mathrm{N}},J_t}(t) = i_{G_{\mathrm{N}},J_t}(t)$ because the non-idle marking on $t \in G$ means that it can only have been matched to an implicit node of L and this precludes it from being one of the redirected nodes of the rewrite. Also in the latter case, $r_{G_{\mathrm{N}},J_t}(t)$ is a notification root in J_t, and doing the notification, makes it idle in K_t, as required.

For case (b), f ends up idle regardless of the order of execution steps.

For case (c) nodes, we note that they start out non-idle, and when notified, change their marking from one non-idle marking to another (non-idle marking). By the definition of rewriting, their markings are unaffected by activation. The relative order of rewriting and notification(s) is thus immaterial for them and they end up with the same node marking regardless.

For case (d), regarding (the g' image or the h_t' image of) a $P - L$ node q, there are two contributing subcases depending on the out-arcs of q. Subcase (d1) concerns all notification out-arcs of q whose child node is (a node whose g' image, resp. h_t' image, is the $r_{G_{\mathrm{N}},G_{\mathrm{N}}'}$ image, resp. the $r_{G_{\mathrm{N}},H_t'}$ image, of) t, or whose child node is the LHS of a redirection where the RHS node is (the $r_{G_{\mathrm{N}},G_{\mathrm{N}}'}$ image, resp. the $r_{G_{\mathrm{N}},H_t'}$ image, of) t. If there are such notification out-arcs, then we have Π by the above quoted **M-I**.11.4.(7) or **M-I**.11.4.(9), since t can only have been matched to an implicit node of L because of its active marking. Subcase (d2) concerns all other out-arcs of q.

Regarding the images of q in the various graphs, if notification is done first, the child node of (d1) out-arcs of $h_t'(q)$ is idle in H_t', but active in J_t, whereupon $j_t(q)$ receives notifications along the (d1) out-arcs which decrease its suspension marking in K_t. (N.B. Because of the earlier notification from t, the only suspended parents that $r_{G_{\mathrm{N}},J_t}(t)$ has, are the parent nodes of these (d1) out-arcs.) If rewriting is done first, the child node of (d1) out-arcs of $g'(q)$ is active in G_{N}', hence in H_f, whereupon the (d1) out-arcs join the image of the notification redex in H_f. $h_f(q)$ therefore receives notifications along the (d1) out-arcs which decrease its suspension marking in K_f. Since by contractum building, the images of q start with the same number of suspensions, and also have the same number of (d1) out-arcs, the markings on them in K_t and K_f are the same. The (d2) out-arcs do not affect the node markings of contractum nodes.

Finally for case (e) nodes, it is clear that they end up with the same marking regardless of the order of the steps, since either they retain the same marking throughout, or they start idle and fall into the appropriate image of *Act* at some point, thence acquiring the active marking.

For arcs, there are four disjoint cases: (a) arcs of the notification redex; (b) contractum arcs in the (d1) subcase of case (d) for nodes discussed above; (c) all other contractum arcs; (d) all other arcs.

For case (a) arcs, they start off as notification arcs, and end up as normal arcs, regardless of the order of steps. Likewise for case (b) arcs; depending on order of steps, they either become normal arcs at the same time as the case (a) arcs, or later, during the extra notification. Case (c) and case (d) arcs retain their arc marking throughout, regardless of the order of steps. We are done. ☺

Lemma 7.5 Let $\mathcal{G}_{\rm N} = [G_0, \ldots, G_{\rm N}]$ be a transitive coercing preexecution of a MONSTR system $\mathcal{R}$. Suppose $G_{\rm N}$ contains an active function node s with $\text{Susp}(s) \neq \emptyset$, where the Susp set of a function node is given in definition 3.3. Suppose $G_{\rm N}$ also contains an active function node f, all of whose $(\text{Map}(\sigma(f)) - \text{State}(\sigma(f)))$ arguments are idle constructors, and whose $\text{State}(\sigma(f))$ argument (if any, let it be v) is an idle constructor or stateholder, and which is thus the root of a redex $g : L \longrightarrow G_{\rm N}$ for some rule $D = (P, \mathit{root}, \mathit{Red}, \mathit{Act})$. Let

$$\begin{aligned}
\text{Susp} = \{k \in \text{Map}(\sigma(s)) \mid\ & \alpha(s)[k] \text{ is non-idle, or} \\
& \alpha(s)[k] \text{ is idle and } \sigma(\alpha(s)[k]) = \bot, \text{ or} \\
& \alpha(s)[k] \text{ is idle and } \sigma(\alpha(s)[k]) \in \mathbf{F}, \text{ or} \\
& [\ k \in (\text{Map}(\sigma(s)) - \text{State}(\sigma(s))) \text{ and} \\
& \alpha(s)[k] \text{ is idle and } \sigma(\alpha(s)[k]) \in \mathbf{V}\ ]\ \}
\end{aligned}$$

$$\text{SuspNodes} = \{x \in G_{\rm N} \mid x = \alpha(s)[k] \text{ for some } k \in \text{Susp}\}$$

$$\overline{\text{Susp}} = \{k \in \text{Map}(\sigma(s)) \mid k \notin \text{Susp}\}$$
$$\overline{\text{SuspNodes}} = \{x \in G_{\rm N} \mid x = \alpha(s)[k] \text{ for some } k \in \overline{\text{Susp}}\}$$

Suppose for every redirection $(a, b) \in \mathit{Red}$, either b is non-idle or $b \in \mathit{Act}$. Let

$$\begin{aligned}
\overline{\text{Susp}}\text{Act} = (&\overline{\text{SuspNodes}} \cap g(\mathit{Act} \cup \{b \mid (a, b) \in \mathit{Red}, a \in \mathit{Act}, b \in L\})) \\
&- (\{\alpha(f)[k] \mid k \in \text{State}(\sigma(f))\} \cap \{\alpha(s)[k] \mid k \in \text{State}(\sigma(s))\})
\end{aligned}$$

$$\Pi \ \equiv\ \overline{\text{Susp}}\text{Act} \neq \emptyset$$

Let H_s be obtained by performing a suspension from s in $G_{\rm N}$. Let H_f be obtained by rewriting the redex rooted at f in $G_{\rm N}$, via the usual phases $g' : P \longrightarrow G_{\rm N}'$, $g'' : P \longrightarrow G_{\rm N}''$, $h_f : P \longrightarrow H_f$, and associated i and r maps. Then

(1) (a) $r_{G_{\rm N},H_f}(s) = i_{G_{\rm N},H_f}(s)$ is an active function node of H_f with non-empty Susp set. Hence $r_{G_{\rm N},H_f}(s)$ the root of a potential suspension step in H_f.

(b) $r_{G_{\rm N},H_s}(f) = i_{G_{\rm N},H_s}(f)$ is an active function node, and

$$h_s = r_{G_{\mathrm{N}},H_s} \circ g : L \to H_s$$

is a redex for D, such that all (Map($\sigma(r_{G_{\mathrm{N}},H_s}(f))$) – State($\sigma(r_{G_{\mathrm{N}},H_s}(f))$)) arguments of $r_{G_{\mathrm{N}},H_s}(f)$ are idle constructors, and any State($\sigma(r_{G_{\mathrm{N}},H_s}(f))$) argument is an idle constructor or stateholder, hence is the redex of a potential rewrite in H_s.

Let J_f be obtained from H_f by performing a suspension from $r_{G_{\mathrm{N}},H_f}(s)$. Let J_s be obtained from H_s by rewriting the redex rooted at $r_{G_{\mathrm{N}},H_s}(f)$ in H_s, via the usual phases $h_s' : P \to H_s'$, $h_s'' : P \to H_s''$, $j_s : P \to J_s$, and associated i and r maps. Then

(2) **If** Π **Then** Every node in $r_{G_{\mathrm{N}},J_f}(\overline{\mathrm{Susp}}\mathrm{Act})$ (resp. $r_{G_{\mathrm{N}},J_s}(\overline{\mathrm{Susp}}\mathrm{Act})$) is an active constructor, hence the root of a potential notification step in J_f (resp. J_s).

Let K_f and K_s be given by

If not Π **Then** $K_f = J_f$ and $K_s = J_s$
Else K_f (resp. K_s) = the result of performing notifications from each node in $r_{G_{\mathrm{N}},J_f}(\overline{\mathrm{Susp}}\mathrm{Act})$ in J_f (resp. $r_{G_{\mathrm{N}},J_s}(\overline{\mathrm{Susp}}\mathrm{Act})$ in J_s)

Then

(3) K_f and K_s are marking preserving isomorphic via a map $\psi : K_f \to K_s$; apart from the exceptional case in which f and s share the same stateholder or constructor node, both in stateholder position, and the f rewrite either redirects it (to a non-idle node, or activated node, or $\perp$-node), or merely activates it without redirection. In symbols if: State($\sigma(s)$) = $\{k_s\}$; State($\sigma(f)$) = $\{k_f\}$; $\alpha(s)[k_s] = \alpha(f)[k_f] = v$; $\mu(r_{G_{\mathrm{N}},K_f}(v)) = \mu(r_{G_{\mathrm{N}},K_s}(v)) \neq \varepsilon$ or $\sigma(r_{G_{\mathrm{N}},K_f}(v)) = \sigma(r_{G_{\mathrm{N}},K_s}(v)) = \perp$: In such a case, $r_{G_{\mathrm{N}},K_f}(s)$ has an extra suspension marking compared with $\psi(r_{G_{\mathrm{N}},K_f}(s)) = r_{G_{\mathrm{N}},K_s}(s)$, and the arc $(r_{G_{\mathrm{N}},K_f}(s)_{k_s}, r_{G_{\mathrm{N}},K_f}(v))$ is a notification arc, whereas $\psi((r_{G_{\mathrm{N}},K_f}(s)_{k_s}, r_{G_{\mathrm{N}},K_f}(v))) = (r_{G_{\mathrm{N}},K_s}(s)_{k_s}, r_{G_{\mathrm{N}},K_s}(v))$ is a normal arc. Even in the exceptional case, K_f and K_s are marking preserving isomorphic via ψ aside from the stated details.

Proof. Obviously $f \neq s$ since f has an empty Susp set while s does not. Equally obviously, f's Map($\sigma(f)$) arguments do not include most kinds of suspension redex nodes since the latter are non-idle, or idle functions, or $\perp$-nodes, or idle stateholders in the wrong place. (In fact one of the latter could be the stateholder argument of f, but this is the only possible exception.) However, this does not prevent the suspension redex nodes from occurring as implicitly matched arguments of the rewriting redex. Because the out-arcs of f and s are disjoint, and the out-arcs of implicitly matched nodes of the rewriting redex are not part of that redex, it is clear that the sets of arcs of the two redexes are disjoint.

Consider performing the suspension to create H_s. Evidently G_{N} and H_s are graph structure isomorphic. And since for suspensions, the only nodes whose markings change are s itself and any idle function activated in the suspension, $r_{G_{\mathrm{N}},H_s}(f)$ is active in H_s and

(1).(b) follows. Let us compare the rewriting processes that create H_f from G_N and K_s from H_s using the rule D. Let

$$\theta : G_\mathrm{N} \rightarrow H_s$$

be the graph structure isomorphism mentioned already. The respective contractum building phases clearly allow its extension to a graph structure isomorphism

$$\theta' : G_\mathrm{N}' \rightarrow H_s'$$

such that the obvious triangle involving $g' : P \rightarrow G_\mathrm{N}'$ and $h_s' : P \rightarrow H_s'$ commutes. The redirection phase admits a further extension to a graph structure isomorphism

$$\theta'' : G_\mathrm{N}'' \rightarrow H_s''$$

such that the triangle involving the node maps $g'' : P \rightarrow G_\mathrm{N}''$ and $h_s'' : P \rightarrow H_s''$ commutes too. Likewise the activation phase finally yields the graph structure isomorphism

$$\theta''' : H_f \rightarrow J_s$$

such that the triangle involving $h_f : P \rightarrow H_f$ and $j_s : P \rightarrow J_s$ commutes. (In particular the equality $\sigma(r_{G_\mathrm{N},K_f}(v)) = \sigma(r_{G_\mathrm{N},K_s}(v))$ mentioned in clause (3) of the lemma is guaranteed to hold.)

As in the previous lemma, the definition of rewriting 3.8 shows that the only active node of the rewritten graph that ends up idle in the result, is the root of the redex. The only other nodes that can undergo a change of marking are the activated nodes which, if they start off idle, end up active. Thus we conclude that since $f \neq s$, $r_{G_\mathrm{N},H_f}(s)$ is active in H_f. To get 1.(a), we must show that $r_{G_\mathrm{N},H_f}(s)$ has a non-empty Susp set. For this is it is sufficient to notice that the rewrite cannot make a non-root non-idle node idle, nor change the symbol on a node, nor redirect a node to an idle non-$\perp$-node. So each Susp argument of s becomes a Susp argument of $r_{G_\mathrm{N},H_f}(s)$, and in fact these may be joined by others if $\overline{\mathrm{Susp}}\mathrm{Act} \neq \varnothing$, or if a node in stateholder position for both f and s, is redirected or activated. So 1.(a) holds.

By the rather stringent conditions for rewrite redexes, (2) holds trivially. To get (3) we must follow what happens to the markings on the nodes and arcs.

For arcs there are five disjoint cases: (a) all arcs (s_k, f), for any applicable $k \in$ Susp, which must all be normal arcs in G_N; (b) other arcs of the suspension redex (i.e. whose child nodes are in SuspNodes); (c) all arcs (s_k, x) not in the suspension redex, but with $k \in (\mathrm{Map}(\sigma(s)) - \mathrm{State}(\sigma(s)))$; (d) any arc (s_k, x) not in the suspension redex, but with $k \in \mathrm{State}(\sigma(s))$; (e) all remaining arcs, whether already existing in G_N, or introduced during rewriting, (this includes all arcs (s_k, x), for any $k \notin \mathrm{Map}(\sigma(s))$).

For case (a) arcs, if rewriting is done first, they remain normal during the rewrite, and since f is redirected to a non-idle node or activated node or $\perp$-node, they become notification arcs after the suspension. If the redirection target was a $\overline{\mathrm{Susp}}\mathrm{Act}$ node, they become normal after the final notification; otherwise not. If the suspension is done first, they become notification arcs immediately, and remain so during the rewrite. During the final notification, they become normal if the redirection target was a $\overline{\mathrm{Susp}}\mathrm{Act}$ node; otherwise not.

For case (b) arcs, they are unaffected by rewriting, and become notification arcs after the suspension, regardless of the order of steps, remaining so in the final notification (if any). For case (c) arcs, if the rewrite is first, they remain normal throughout; unless their child node was in $\overline{\text{SuspAct}}$, in which case they become notification arcs after the suspension, returning to normal after the final notification. If the suspension is first, they remain normal through both the suspension and rewrite and final notification.

If there is a case (d) arc, its child is an idle constructor or stateholder. Either may get activated, and a stateholder may get redirected by the rewrite, which will make the arc's child a non-idle node or $\perp$-node. Therefore if any of these happen and the rewrite is done first, it joins the suspension redex, and becomes a notification arc during the suspension; this does not happen if the rewrite comes second. The arc is unaffected by the final notifications whatever the order. For case (e) arcs, they retain the marking they had in G_{N}, or were given during contractum building, regardless of the order of steps, except for contractum notification arcs with contractum parent nodes and $\overline{\text{SuspAct}}$ child nodes, which become normal after the final notification, regardless of order of steps.

For nodes there are six disjoint cases: (a) s; (b) f; (c) nodes in SuspNodes other than f and its child in stateholder position (if applicable); (d) nodes in $\overline{\text{SuspNodes}}$ other than the child of f in stateholder position if any; (e) the child of f in stateholder position if any; (f) all remaining nodes, whether already existing in G_{N}, or introduced during rewriting.

For the case (b) node f, its marking is unaffected by the suspension, and it is quiesced during the rewrite. This holds regardless of the order of the steps. For the case (c) nodes, the non-idle nodes remain so, regardless of the order of steps, being unaffected by any activations from the rewrite, or final notification. Any idle functions are activated either (perhaps) by the rewrite, or by the suspension, and remain thus. Idle stateholders, might be activated by the rewrite, or not regardless of order; $\perp$-nodes remain so. For the case (d) nodes, we know they must be idle constructors. During the rewrite, they might be activated, but will subsequently notify in the final notification. In such a case, if the rewite is first they join the suspension redex, otherwise not.

For case (e), if there is a child of f in stateholder position, if it occurs in SuspNodes, the argument is as for the SuspNodes nodes, since it must be an idle stateholder in constructor position for s. Thus it may (or may not) be activated, or redirected to a non-idle node or activated node or $\perp$-node by the rewrite regardless of order. If it occurs outside of the Map($\sigma(s)$) arguments of s, it is unaffected by the suspension and notification, whatever the order of steps. If it occurs in $\overline{\text{SuspNodes}}$, either it is a constructor, in which case the rewrite may (or may not) activate it regardless of order. If the rewrite does activate it, it joins the suspension redex if the suspension occurs second, and provided it is not in stateholder position of s, it notifies in the last step, again regardless of order. Otherwise if it occurs in $\overline{\text{SuspNodes}}$, it must be a stateholder, in which case it must be the stateholder argument of s as well as that of f. In this case, the rewrite may activate it, or redirect it (to a non-idle or activated or $\perp$-node) whereupon, if the suspension occurs second, it joins the suspension redex. (In any case the equality $\mu(r_{G_{\text{N}},K_f}(v)) = \mu(r_{G_{\text{N}},K_s}(v))$ (when $\alpha(s)[k_s] = \alpha(f)[k_f] = v$) mentioned in clause (3) of the lemma is guaranteed to hold.)

For case (f) nodes, either they retain the marking they had in G_{N}, or were given during contractum building; or they undergo an activation. This holds regardless of the order of the steps.

For the case (a) node s, if suspension is done first, its marking changes from active to suspended, with as many suspensions in total, as there are: Map($\sigma(s)$) arcs to case (c) nodes, plus Map($\sigma(s)$) arcs to f if f is a matched argument of s, plus (Map($\sigma(s)$) – State($\sigma(s)$)) arcs to the stateholder child of f (in stateholder position of f) if any, if it is a constructor position argument of s. The marking remains during the rewriting step and final notification.

If rewriting is done first, the Map($\sigma(s)$) argument arcs to f (if any), become redirected to an activated node or non-idle node or $\perp$-node. Similarly for the Map($\sigma(s)$) argument arcs to the child v of f in stateholder position if it got activated or non-root redirected. If in fact either occured, and v was also in stateholder position of s, the fact that $r_{G_N,H_f}(v)$ is non-idle or a $\perp$-node, means it joins the suspension redex of $r_{G_N,H_f}(s)$. The rewrite also potentially activates some nodes, and those that are in $\overline{\text{SuspAct}}$ join the suspension redex of $r_{G_N,H_f}(s)$ as well. These latter, notify during the final notification, so that the suspension marking on $r_{G_N,K_f}(s)$ is one more than that on $r_{G_N,K_s}(s)$ if $r_{G_N,H_f}(v)$ joined the suspension redex of $r_{G_N,H_f}(s)$, otherwise being the same. We are done. ☺

Lemma 7.6 Let $\mathcal{G}_N = [G_0, \ldots, G_N]$ be a transitive coercing preexecution of a MONSTR system $\mathcal{R}$. Suppose G_N contains two active function nodes $f_1 \neq f_2$. Suppose for $i \in \{1, 2\}$, all of the (Map($\sigma(f_i)$) – State($\sigma(f_i)$)) arguments of f_i are idle constructors, and any State($\sigma(f_i)$)) argument of f_i is an idle constructor or stateholder, and suppose therefore that f_i is the root of a redex $g_i : L_i \rightarrow G_N$ for some rule $D_i = (P_i, root_i, Red_i, Act_i)$. Suppose for each redirection $(a, b) \in Red_i$, either b is non-idle or b is in Act_i. For either choice of $i \in \{1, 2\}$, let j denote the other choice. If L_i (the left subpattern of P_i) contains an explicit stateholder, let it be s_i. If for some $t_i \in P_i$, $(s_i, t_i) \in Red_i$, then we say D_i redirects s_i, otherwise not.

Let

$$\begin{aligned}
&\text{MapNodes}_i = \{x \in G_N \mid x = \alpha(f_i)[k] \text{ for some } k \in \text{Map}(\sigma(f_i))\} \\
&\text{RedNodes}_i = \{x \in G_N \mid x = g_i(a) \text{ for some } (a, b) \in Red_i\} \\
&\text{LActNodes}_i = g_i(Act_i \cup \{b \mid (a, b) \in Red_i, a \in Act_i, b \in L_i\})
\end{aligned}$$

Suppose

$$g_1(s_1) = v_1 = v_2 = g_2(s_2) \;\Rightarrow\; [\text{ For both } i \in \{1, 2\}, D_i \text{ does not redirect } s_i\ ].$$

Let H_i be obtained by rewriting the redex rooted at f_i in G_N, via the usual phases $g_i' : P_i \rightarrow G_{Ni}'$, $g_i'' : P_i \rightarrow G_{Ni}''$, $h_i : P_i \rightarrow H_i$, and associated i and r maps. Let

$$\begin{aligned}
&Red_{G_{Ni}} = \{(x, y) \in G_N{\times}G_N \mid \text{for some } (a, b) \in Red_i, g_i(a) = x, g_i(b) = y\} \\
&Red_i^{\circ} = \{(x, y) \in G_N{\times}G_N \mid \text{for some } (a, b) \in Red_i, g_i(a) = x, g_i(b) = y, \\
&\qquad\qquad\qquad \text{and } x\ Red_{G_{Ni}}{+}\ x\} \\
&Red_{1\&2}^{\circ} = \{(x, y) \in G_N{\times}G_N \mid \text{for some } (a, b) \in (Red_{G_{N1}} \cup Red_{G_{N2}}), \\
&\qquad\qquad [(g_1(a) = x \text{ and } g_1(b) = y), \text{ or } (g_2(a) = x \text{ and } g_2(b) = y)], \\
&\qquad\qquad \text{and } x\ (Red_{G_{N1}} \cup Red_{G_{N2}}){+}\ x\}
\end{aligned}$$

Let

$$\text{NN}_{i.1} = \text{LActNodes}_i \cap \text{MapNodes}_j$$

Then

(1) $Not_{i.1} = r_{G_\mathrm{N},H_i}(\mathrm{NN}_{i.1}) = i_{G_\mathrm{N},H_i}(\mathrm{NN}_{i.1})$ contains only active constructors, possibly combined with an active stateholder.

Let M_i be the result of performing notifications from all nodes in $Not_{i.1}$.

Then

(2) $r_{G_\mathrm{N},M_i}(f_j) = i_{G_\mathrm{N},M_i}(f_j)$ is an active function node of M_i, and

$$m_i = r_{G_\mathrm{N},M_i} \circ g_j : L_j \to M_i$$

is a redex for D_j, such that all the $(\mathrm{Map}(\sigma(r_{G_\mathrm{N},M_i}(f_j))) - \mathrm{State}(\sigma(r_{G_\mathrm{N},M_i}(f_j))))$ arguments of $r_{G_\mathrm{N},M_i}(f_j)$ are idle constructors, and any $\mathrm{State}(\sigma(r_{G_\mathrm{N},M_i}(f_j)))$ argument is an idle constructor or stateholder, hence is the redex of a potential rewrite in M_i.

Let N_i be obtained from M_i by rewriting the redex rooted at $r_{G_\mathrm{N},M_i}(f_j)$ in M_i, via the usual phases $m_i' : P_j \to M_i'$, $m_i'' : P_j \to M_i''$, $n_i : P_j \to N_i$, and associated i and r maps.

Let

$$\begin{aligned}\mathrm{NN}_3 \;=\; & (\mathrm{LActNodes}_1 \cup \mathrm{LActNodes}_2) \cap \\ & ((\mathrm{MapNodes}_1 - \mathrm{RedNodes}_1) \cup (\mathrm{MapNodes}_2 - \mathrm{RedNodes}_2))\end{aligned}$$

Then

(3) $Not_{i.3} = r_{G_\mathrm{N},N_i}(\mathrm{NN}_3) = i_{G_\mathrm{N},N_i}(\mathrm{NN}_3)$ contains only constructors and stateholders, each in either the active or idle state.

Let K_i be the result of performing notifications from all active nodes in $Not_{i.3}$.

Then

(4) K_1 and K_2 are marking preserving isomorphic via a map $\psi : K_1 \to K_2$; apart from the exceptional cases where

(a) D_j shares and activates an unredirected unactivated stateholder of D_i which has a notification contractum in-arc. In symbols if: for some $p_i \in P_i$, $0 < z = |\{l \mid \alpha(p_i)[l] = s_i, \nu(p_i)[l] = \wedge, v_i \notin \mathrm{RedNodes}_i, v_i \notin \mathrm{LActNodes}_i, v_i \in \mathrm{LActNodes}_j, v_i \in \mathrm{MapNodes}_j\}|$. In such a case, if $\mu(i_{N_j,K_j}(n_j(p_i))) = \#^q$, then $\mu(i_{H_i,K_i}(h_i(p_i))) = \#^{(q-z)}$, and each relevant arc $(i_{N_j,K_j}(n_j(p_i))_l, r_{G_\mathrm{N},K_j}(v_i))$ is a notification arc, whereas $(i_{H_i,K_i}(h_i(p_i))_l, r_{G_\mathrm{N},K_i}(v_i))$ is a normal arc. (And these respective pairs of nodes and arcs correspond via ψ or ψ^{-1} according to the subscripts.)

(b) D_j activates a redirected stateholder of D_i which has a notification in-arc in G_N (and similarly for j). In symbols if: for some $u \in G_\mathrm{N}$, $z_i = |\{l \mid \alpha(u)[l] = v_i, \nu(u)[l] = \wedge, v_i \in \mathrm{RedNodes}_i, v_i \in \mathrm{LActNodes}_j\}|$, and $z_j = |\{l \mid \alpha(u)[l] = v_j, \nu(u)[l] = \wedge, v_j \in \mathrm{RedNodes}_j, v_j \in \mathrm{LActNodes}_i\}|$, and $0 < z_i + z_j$. In such a case, if $\mu_{G_\mathrm{N}}(u) = \#^q$, then we have $\mu(i_{G_\mathrm{N},K_i}(u)) = \#^{(q-z_j)}$ and

$\mu(i_{G_\mathrm{N},K_j}(u)) = \#^{(q-z_i)}$; and each relevant arc $(i_{G_\mathrm{N},K_i}(u))_l, r_{G_\mathrm{N},K_i}(v_j))$ is a notification arc, whereas $(i_{G_\mathrm{N},K_j}(u))_l, r_{G_\mathrm{N},K_j}(v_j))$ is a normal arc, and each relevant arc $(i_{G_\mathrm{N},K_j}(u))_l, r_{G_\mathrm{N},K_j}(v_i))$ is a notification arc, whereas $(i_{G_\mathrm{N},K_i}(u))_l, r_{G_\mathrm{N},K_i}(v_i))$ is a normal arc. (And these respective pairs of nodes and arcs correspond via ψ or ψ^{-1} according to the subscripts.)

Even in the exceptional cases, K_1 and K_2 are marking preserving isomorphic via ψ apart from the stated details.

Proof. Since $f_i \neq f_j$, and both are active, and because all (Map – State) arguments of both f's are idle constructors and any State argument must be an idle constructor or stateholder, the only overlap between $(\{f_i\} \cup \mathrm{MapNodes}_i)$ and $(\{f_j\} \cup \mathrm{MapNodes}_j)$ is on common idle constructors, or a single shared but unredirected idle stateholder. (Of course implicitly matched nodes of either redex may match arbitrary nodes, including arbitrary nodes of the other redex). Noting that an activation from the first rewrite does not affect the root of the redex of the second, clauses (1), (2) and (3) are clear.

It remains to establish the marking preserving isomorphism claimed in (4), which we do in five stages.

Stage 1. First we define a bijection between the nodes of K_1 and K_2. Images of G_N in K_1 and K_2 are made to correspond, as are corresponding images of contractum nodes, and any $\perp$-nodes introduced during redirection. Thus

$$\theta : N_{K_1} \to N_{K_2}$$

where

$$\begin{aligned}
&\theta(i_{G_\mathrm{N},K_1}(x)) = i_{G_\mathrm{N},K_2}(x) \;\text{ for }\; x \in G_\mathrm{N} \\
&\theta(i_{H_1,K_1}(h_1(p_1))) = i_{N_2,K_2}(n_2(p_1)) \;\text{ for }\; p_1 \in N_{P_1} - N_{L_1} \\
&\theta(i_{N_1,K_1}(n_1(p_2))) = i_{H_2,K_2}(h_2(p_2)) \;\text{ for }\; p_2 \in N_{P_2} - N_{L_2} \\
&\theta(i_{H_1,K_1}(q_{1.1})) = i_{N_2,K_2}(q_{1.2}) \;\text{ for }\; \langle q_{1.1}, q_{1.2}\rangle \in \\
&\qquad \{\langle i_{G_{\mathrm{N}1}'',H_1} \circ i_{G_\mathrm{N},G_{\mathrm{N}1}''}([-]^\circ), i_{M_2'',N_2} \circ i_{G_\mathrm{N},M_2''}([-]^\circ)\rangle \mid \\
&\qquad\qquad [-]^\circ \subseteq Red_1{}^\circ{*}\}, \\
&\theta(i_{N_1,K_1}(q_{2.1})) = i_{H_2,K_2}(q_{2.2}) \;\text{ for }\; \langle q_{2.1}, q_{2.2}\rangle \in \\
&\qquad \{\langle i_{M_1'',H_1} \circ i_{G_\mathrm{N},M_1''}([-]^\circ), i_{G_{\mathrm{N}2}'',H_2} \circ i_{G_\mathrm{N},G_{\mathrm{N}2}''}([-]^\circ)\rangle \mid \\
&\qquad\qquad [-]^\circ \subseteq Red_2{}^\circ{*}\}, \\
&\theta(i_{N_1,K_1}(q_{3.1})) = i_{N_2,K_2}(q_{3.2}) \;\text{ for }\; \langle q_{3.1}, q_{3.2}\rangle \in \\
&\qquad \{\langle i_{G_{\mathrm{N}1}'',N_1} \circ i_{G_\mathrm{N},G_{\mathrm{N}1}''}([-]^\circ), i_{G_{\mathrm{N}2}'',N_2} \circ i_{G_\mathrm{N},G_{\mathrm{N}2}''}([-]^\circ)\rangle \mid \\
&\qquad\qquad [-]^\circ \subseteq (Red_{1\&2}{}^\circ{*} - (Red_1{}^\circ{*} \cup Red_2{}^\circ{*}))\}
\end{aligned}$$

We can see that this is a well defined bijection, provided we note some things. First, we assume a sufficiently fussy construction for disjoint union during contractum building has ensured all introduced nodes are distinct. Second, the composition symbols $\circ$ which occur, hide a slight abuse of notation. The earlier map in the composition refers to an equivalence class of nodes (i.e. a set of nodes), before they have been formed into a $\perp$-node; the latter one, to the $\perp$-node itself (i.e. an individual node). Thirdly, the last three cases are genuinely disjoint since the cycle of nodes that forms a $\perp$-node during redirection has the property that each member is both the left node and the right node of a redirection. As we are dealing with redexes which already exist in G_{N}, witnesses to the cycles that comprise the $\perp$-nodes created, also exist already. Thus in G_{N}, since for every left node of a redirection there is a unique right node, cycles entirely contained in the first redex are disjoint from those contained in the second redex. These in turn are disjoint from cycles spanning both redexes, since the latter consist of (an even number of) chains contained entirely in one or other redex, for which the first node is not the right node of a redirection, and the last node is not the left node of a redirection, but such that the chains glue together suitably to form the cycle; and for which the first rewrite (in whichever order), shorts out the pieces relevant to that rewrite, creating a genuine cycle for the second rewrite. (We will see all this in a little more detail below.)

This completes stage 1.

Now we extend θ to a graph structure isomorphism by checking out the arcs. This occupies three stages since we argue separately about arc tails and arc heads (so each arc is covered by one of the head cases and one of the tail cases), and then bring the two together in a third stage.

Stage 2. We first check the arc tails, which are easy since tails of arcs never move during redirection. So the cases above for nodes extend θ immediately to a bijection on tails of arcs as θ-related nodes have the same arity.

Stage 3. Since arc heads follow the redirection functions under rewriting, we next calculate the r_{W,K_i} functions of all nodes, where W is as appropriate for the node in question. Then we check that θ expresses the right relationship between the various possibilities. There are seven cases, following the breakdown of cases for θ above (the last case splits into two): (a) nodes of G_{N} where W is G_{N}; (b$_i$) contractum nodes introduced in the D_i rewrite where W is either H_i or N_i depending on order of rewriting; (c_i) $\perp$-nodes properly belonging to one or other rewrite where W is either H_i or N_i depending on order of rewriting; (d_i) $\perp$-nodes properly belonging to both rewrites where W is N_i.

We note that $\mathrm{RedNodes}_i \cap \mathrm{RedNodes}_j = \varnothing$ follows easily from the hypotheses; and from this we conclude

$$r_{G_{\mathrm{N}},H_i}(y) = i_{G_{\mathrm{N}},H_i}(y) \text{ for all } y \in \mathrm{RedNodes}_j$$
$$r_{G_{\mathrm{N}},H_j}(y) = i_{G_{\mathrm{N}},H_j}(y) \text{ for all } y \in \mathrm{RedNodes}_i$$

Now for case (b$_i$), for an instantiated contractum node x, say $x = w(c_i)$, for either version of W where $w : P_i \to W$, we have $r_{W,K_i}(x) = i_{W,K_i}(x)$. This is because the first rewrite, of D_i say, only redirects nodes in $\mathrm{RedNodes}_i$, and the second rewrite, perforce of D_j, only redirects nodes in $r_{G_{\mathrm{N}},M_i}(\mathrm{RedNodes}_j) = i_{G_{\mathrm{N}},M_i}(\mathrm{RedNodes}_j)$. Neither of these includes any instantiated contractum nodes. It is clear that θ expresses the right relationship between the $r_{W,K_i}(x)$ images of such nodes.

A similar argument works for cases (c_i) and (d_i), because $\perp$-nodes are not redirected either. Thus $r_{W,K_i}(x) = i_{W,K_i}(x)$ for such nodes, and again θ gives what is required.

For a case (a) node x there are three subcases: (a.1) $x \notin (\mathrm{RedNodes}_i \cup \mathrm{RedNodes}_j)$; (a.2$_i$) $x \in \mathrm{RedNodes}_i$.

For subcase (a.1), suppose D_i rewrites first. We have $r_{G_{\mathrm{N}},H_i}(x) = i_{G_{\mathrm{N}},H_i}(x)$. Subsequently $i_{G_{\mathrm{N}},M_i}(x)$ is not redirected in the D_j rewrite either, and $r_{G_{\mathrm{N}},K_i}(x) = i_{G_{\mathrm{N}},K_i}(x)$. By symmetry we get $r_{G_{\mathrm{N}},K_j}(x) = i_{G_{\mathrm{N}},K_j}(x)$ if D_j rewrites first. Therefore by the first clause for θ, we find $\theta(r_{G_{\mathrm{N}},K_1}(x)) = r_{G_{\mathrm{N}},K_2}(x)$ as required.

For the subcases (a.2_i), there are a large number of sub…subcases, depending on how the redirections combine together. Consider $x \in (\mathrm{RedNodes}_i \cup \mathrm{RedNodes}_j)$. So $x = g_{i!j}(a)$ (where $i!j$ stands for either i or j), for some redirection $(a, b) \in (Red_i \cup Red_j)$. Consider b. It is either a contractum node (case C) and the analysis stops, or not. If not, then $y = g(b)$ is either not in $(\mathrm{RedNodes}_i \cup \mathrm{RedNodes}_j)$ (case I) and the analysis stops, or it is. In the latter case, either y is a node we have analysed already (case $\mathrm{R}_\perp$) and the analysis stops having found a cycle of redirections, or not and we continue the analysis with y (case R). And so on. For any given $x \in (\mathrm{RedNodes}_i \cup \mathrm{RedNodes}_j)$, the whole process continues for at most four steps, since that is the maximum number of distinct redirections.

Each possible combination for a node $x \in (\mathrm{RedNodes}_i \cup \mathrm{RedNodes}_j)$ corresponds to a connected directed graph of redirections, with a single source vertex, with each vertex having at most one out-edge, whose directed edges are given by the $(a, b) \in (Red_i \cup Red_j)$, with vertices b and c identified iff $g_{i!j}(b) = g_{i!j}(c)$, and with each edge (a, b) coloured i or j according to whether (a, b) is in Red_i or in Red_j. Fig. 10 shows the cases of length 4, supressing the i and j colouring (which serves to multiply the number of possibilities by a factor of 6), and for the $\mathrm{R}_\perp$ cases indicating the previously encountered vertex with a blob.

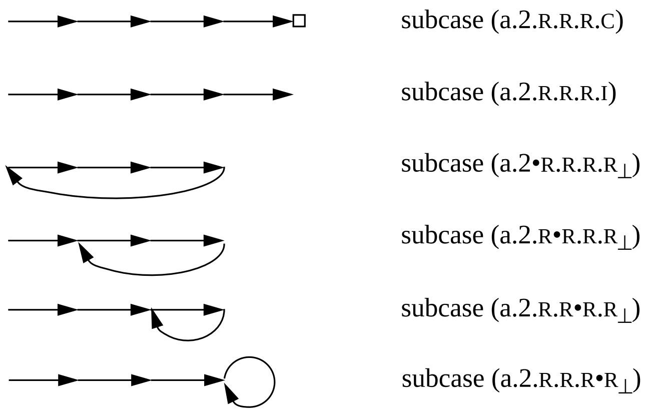
subcase (a.2.R.R.R.C)
subcase (a.2.R.R.R.I)
subcase (a.2•R.R.R.R$_\perp$)
subcase (a.2.R•R.R.R$_\perp$)
subcase (a.2.R.R•R.R$_\perp$)
subcase (a.2.R.R.R•R$_\perp$)

Fig. 10 Cases for redirection chains of length 4.

We can see that the complete set of possibilities for a node $x \in (\mathrm{RedNodes}_i \cup \mathrm{RedNodes}_j)$ is given by an expression of the form .R…R.X, where: the chain .R…R. contains at least none, and at most three R's; X is C or I or $\mathrm{R}_\perp$, and if $\mathrm{R}_\perp$ then one of the preceding dots is a blob; and where each R or X is coloured i or j with at most two of any one colour occurring.

Where both i and j colours occur for some particular case, both rewrites play a role in determining the ultimate redirection target for x, otherwise only one of them does.

Suppose only one colour occurs, so the chain is of length at most 2. If the case ends in C, the $i_{\_,\_}$ image of the contractum node instance provides the final redirection target for x regardless of the order of rewriting. If the case ends in I, the $i_{\_,\_}$ image of the root node of the relevant $[-]^{-}$ class (which is certainly not redirected because only one colour occurs), provides the final redirection target regardless of the order of rewriting. If the case ends in $\mathrm{R}_{\perp}$, the $i_{\_,\_}$ image of the $\perp$-node for the relevant $[-]^{\circ}$ class provides the final redirection target regardless of the order of rewriting. All these possibilities are correctly related by θ.

Suppose both colours occur. Then any maximal single coloured segment of the chain which is not the last segment of the whole chain, must relate to part of a $[-]^{-}$ class of the appropriate rewrite (say the D_i rewrite), and its final vertex (corresponding to the root node of the $[-]^{-}$ class) is necessarily implicitly matched by D_i to a node that is explicitly matched by D_j to the left node of a redirection of D_j.

Suppose D_i rewrites first and that the last segment belongs to D_j. Then it is clear that the redirections of the D_i rewrite serve to short-circuit the i-coloured segments, by redirecting all non-root nodes of each relevant $[-]^{-}$ class to its root. As the root also corresponds to the initial vertex of the following j-coloured segment, the whole chain is transformed into one of the single coloured cases dealt with above, for the D_j rewrite.

Now suppose D_j rewrites first. All j-coloured segments except the last act as short-circuits as previously, but now the final segment participates in the rewrite too. If the last edge of the last segment ends in the C or I cases, then the redirections of the D_j rewrite short circuit this segment, and transform the immediately preceding i-coloured segment of the original chain from a pure short-circuit, to a C or I -ended case respectively, as this preceding segment now becomes the last part of the chain for the D_i rewrite. If the last edge of the last segment ends in $\mathrm{R}_{\perp}$, there are two subcases, depending on whether the cycle consists purely of edges of a single colour (necessarily j), or whether both colours are involved. If the former, then the last segment acts like the $\mathrm{R}_{\perp}$-ended case for the single coloured situation, creating a $\perp$-node for the relevant $[-]^{\circ}$ class, which puts the preceding i-coloured segment into the I-ended case, as the $\perp$-node does not get redirected. If the latter, then the last segment just acts like another of the previously discussed short-circuits, as its final vertex necessarily corresponds to a node which is also the left node of a redirection (of either the D_i or the D_j rewrite). In this case, the D_j rewrite short-circuits part of the cycle, creating a smaller cycle for the D_i rewrite, which now falls into an $\mathrm{R}_{\perp}$-ended case for the single coloured situation.

Tedious detailed calculations for all the possibilities, along the lines of that for subcase (a.1), confirm that the results for either order of rewriting are correctly related by θ.

This completes stage 3.

Stage 4. We now utilise the results of stage 3 to show that all arcs of K_1 and K_2 are related as required. There are three cases: (a) arcs of G_{N}; (b$_i$) instantiations of contractum arcs of D_i.

Let (p_k, c) be an arc of G_{N}. Then $\theta(i_{G_{\mathrm{N}},K_1}(p)) = i_{G_{\mathrm{N}},K_2}(p)$ by stage 2, and $\theta(r_{G_{\mathrm{N}},K_1}(c)) = r_{G_{\mathrm{N}},K_2}(c)$ by stage 3, so

$$\theta((i_{G_\mathrm{N},K_1}(p)_k, r_{G_\mathrm{N},K_1}(c))) = (i_{G_\mathrm{N},K_2}(p)_k, r_{G_\mathrm{N},K_2}(c))$$

and we have what we need for case (a) arcs.

For case (b_i) arcs there are two subcases: (b_i.C) where the child node is an instantiation of a contractum node; (b_i.I) where the child node is a matching image of a left pattern node.

For case (b_i.C), let (p_k, c) be an arc between two contractum nodes of D_i. Case (b_i) of stage 3 assures us that the instantiations of neither p nor c get redirected. The homomorphic nature of the contractum building phase, the second and third clauses for θ, and symmetry, then assure us that

$$\theta((i_{H_1,K_1}(h_1(p))_k, r_{H_1,K_1}(h_1(c)))) = ((i_{N_2,K_2}(n_2(p)))_k, r_{N_2,K_2}(n_2(c)))$$

if $D_i = D_1$, and

$$\theta(((i_{N_1,K_1}(n_1(p)))_k, r_{N_1,K_1}(n_1(c)))) = (i_{H_2,K_2}(h_2(p))_k, r_{H_2,K_2}(h_2(c)))$$

otherwise.

For case (b_i.I), let (p_k, c) be an arc from a contractum node p to a left pattern node c of D_i. We first note that due to the homomorphic nature of the contractum building phase, the homomorphism $g_i' : P_i \rightarrow G_{\mathrm{N}i}'$ guarantees that $G_{\mathrm{N}i}'$ has a copy, $g_i'((p_k, c))$ of (p_k, c) if D_i rewrites first, and the homomorphism $m_j' : P_i \rightarrow M_j'$ guarantees that M_j' has a copy, $m_j'((p_k, c))$ of (p_k, c) if D_i rewrites second. Case (b_i) of stage 3 assures us that the instantiations of p do not get redirected. So for p, noting that $i_{G_{\mathrm{N}i}',H_i}(g_i'(p)) = h_i(p)$ in the first case, and $i_{M_i',N_i}(m_i'(p)) = n_i(p)$ in the second case, we can use the second and third clauses for θ as above.

For c, there will be a node $x \in G_\mathrm{N}$ such that $i_{G_\mathrm{N},G_{\mathrm{N}i}'}(x) = r_{G_\mathrm{N},G_{\mathrm{N}i}'}(x) = g_i'(c)$ in the first case, and $r_{G_\mathrm{N},M_j'}(x) = m_j'(c)$ in the second case, given that we have clause (2) of the theorem. Now using stage 3 for $x \in G_\mathrm{N}$, (which allows us to factorise the $r_{G_\mathrm{N},K_1}(x)$ and $r_{G_\mathrm{N},K_2}(x)$ maps at $G_{\mathrm{N}i}'$ and at M_j'), and symmetry, allows us to conclude that

$$\theta((i_{G_{\mathrm{N}1}',K_1}(g_1'(p))_k, r_{G_{\mathrm{N}1}',K_1}(g_1'(c)))) = (i_{M_2',K_2}(m_2'(p))_k, r_{M_2',K_2}(m_2'(c)))$$

if $D_i = D_1$, and

$$\theta((i_{M_1',K_1}(m_1'(p))_k, r_{M_1',K_1}(m_1'(c)))) = (i_{G_{\mathrm{N}2}',K_2}(g_2'(p))_k, r_{G_{\mathrm{N}2}',K_2}(g_2'(c)))$$

otherwise.

At this point θ is a graph structure isomorphism.

Stage 5. Finally we turn our attention to the markings, starting with the node markings.

There are eleven disjoint cases: (a_i) the root f_i; (b_i) nodes in $\mathrm{RedNodes}_i$ other than f_i (if any); (c_i) nodes in $\mathrm{MapNodes}_i$ – $\mathrm{Rednodes}_i$; (d_i) instantiations of contractum nodes of D_i; (e) ⊥-nodes created by either rewrite; (f) non-idle nodes of G_N not previously mentioned; (g) idle nodes of G_N not previously mentioned.

For case (a_i), f_i is quiesced in the D_i rewrite. If it occurred in $\mathrm{ActNodes}_j$ then if D_i rewrites first, the redirection of f_i to a non-idle, or activated, or ⊥- node ensures that the node $i_{G_\mathrm{N},M_i}(f_i)$ is not accessible to the D_j rewrite later; if D_j rewrites first, it is unaffected by any activation, being active already. So $\mu(i_{G_\mathrm{N},K_1}(f_i)) = \mu(i_{G_\mathrm{N},K_2}(f_i)) = \varepsilon$.

For case (b$_i$), the redirected stateholder v_i starts off idle. If D_i rewrites first, it is redirected to a non-idle, or activated, or $\perp$- node, and hence $i_{G_{\mathrm{N}},M_i}(v_i)$ is idle and remains immune to any activation from D_j. If D_j rewrites first, it may be activated, but then notifies, ready for the D_i rewrite, which is as before. Either way we have, $\mu(i_{G_{\mathrm{N}},K_1}(v_i)) = \mu(i_{G_{\mathrm{N}},K_2}(v_i))$.

For case (c$_i$), the relevant nodes are idle constructors and possibly a stateholder which is not redirected. Such a node may be activated by the first rewrite, and if it is matched by the second rewrite, notifies immediately beforehand. It may be activated again, but in any event, it notifies in the last step if it is active by then. Either way, for such a node c_i, $\mu(i_{G_{\mathrm{N}},K_1}(c_i)) = \mu(i_{G_{\mathrm{N}},K_2}(c_i)) = \varepsilon$.

For case (d$_i$), a contractum node d_i, an instantiation of node p_i of D_i, is first created with the marking specified by the rule, regardless of the order of rewriting. The only way that this can change, is if it has a notification out-arc to a child node which is one of the notifying nodes, or it receives an activation via a redirection. These two possibilities are mutually exclusive. If an activation changes the marking, d_i must have been created idle, ends up active regardless of rewriting order, and cannot receive notifications by balancedness.

If it receives a notification, d_i must have been created suspended. In that case any relevant child node is either among the MapNodes nodes (of either rewrite) and is activated by D_i, or is the unredirected and unactivated stateholder node of the D_i rewrite, (by **M-I.11.4**.(7),(9) and the hypotheses for redirections of D_i and for which nodes notify). In the former case, the notification takes place regardless of the order of rewriting, and regardless of whether D_j also activates the node, because any arc can only be notified once, and the hypotheses guarantee that this happens. The same applies in the latter case if D_j does not activate the node. In these cases therefore, we have $\mu(i_{H_i,K_i}(h_i(p_i))) = \mu(i_{N_j,K_j}(n_j(p_i)))$.

However in the latter case, if D_j does activate the node, it ends up with different markings depending on rewriting order. Thus if D_j rewrites first, it activates the stateholder node of the D_i rewrite which promptly notifies, before d_i has been instantiated, and thus $i_{N_j,K_j}(n_j(p_i))$ has not received a notification. If D_i rewrites first, the instantiation of p_i is now earlier, and $i_{H_i,M_i}(h_i(p_i))$ is around and able to receive the notification following the activation from the D_j rewrite. For this case therefore, $\mu(i_{H_i,K_i}(h_i(p_i)))$ has as many fewer suspension markings compared to $\mu(i_{N_j,K_j}(n_j(p_i)))$, as there are notification out-arcs from p_i to s_i in D_i.

For case (e), we note that a $\perp$-node is immune to activations, and cannot receive notifications because it has no out-arcs, so $\mu_{K_i}(\perp) = \mu_{K_j}(\perp) = \varepsilon$, for any θ-related $\perp$-nodes indicated informally by $\perp$.

For case (f), the marking of an otherwise not discussed non-idle node u of G_{N} can only be affected if it is suspended and receives a notification. In this case the child node in question must belong to (MapNodes$_i$ $\cup$ MapNodes$_j$). If the child is never redirected, or never activated (or both), for either order of rewriting it either does or does not notify, consistently. So if u has only such children at worst, we get $\mu(i_{G_{\mathrm{N}},K_1}(u)) = \mu(i_{G_{\mathrm{N}},K_2}(u))$. The same holds if the child is redirected (and/or perhaps activated) by one rewrite and not activated by the other.

However, if the child is redirected by D_i and activated by D_j (which means the child must be v_i), if D_i rewrites first, the child is redirected to a non-idle, or activated, or $\perp$-

node, which means that it is immune to the subsequent activation from D_j, and hence does not notify. On the other hand, if D_j rewrites first, v_i is activated and immediately notifies, so a notification goes to u. For this case therefore, $\mu(i_{G_{\mathrm{N}},K_i}(u))$ has as many more suspension markings compared to $\mu(i_{G_{\mathrm{N}},K_j}(u))$, as there are notification out-arcs from u to v_i in G_{N}. (N.B. For simplicity, we have assumed no out-arcs of a similar kind from u to v_j; if there are some, the required difference in number of suspensions is the difference of two such calculations.)

For case (g), an idle node w of G_{N} not otherwise discussed, can only have its marking changed by virtue of receiving an activation. Since w is not redirected, it is clear that w does or does not receive such an activation consistently, regardless of the order of rewriting. So $\mu(i_{G_{\mathrm{N}},K_1}(w)) = \mu(i_{G_{\mathrm{N}},K_2}(w))$.

This completes the discussion of nodes. The argument for arcs follows that for nodes. All arcs of K_i are either (i_{G_{N},K_i}, r_{G_{N},K_i}) copies of arcs of G_{N}, or (i_{H_i,K_i}, r_{H_i,K_i}) or (i_{N_i,K_i}, r_{N_i,K_i}) copies of instantiations of contractum arcs of D_i and D_j. Thus corresponding arcs in K_1 and K_2 either are or are not notification arcs consistently, except for the cases where differing numbers of notifications are received by the parent node, in which cases the arcs to the relevant stateholders are or are not notification arcs. These cases can be infered easily from the discussion of nodes, so we will not comment further. We are done. ☺

Lemma 7.7 Let $\mathcal{G}_{\mathrm{N}} = [G_0, \ldots, G_{\mathrm{N}}]$ be a transitive coercing preexecution of a MONSTR system $\mathcal{R}$. Suppose G_{N} contains two active function nodes $f_r \neq f_s$. Suppose for f_r, all of the (Map($\sigma(f_r)$) – State($\sigma(f_r)$)) arguments of f_r are idle constructors, that State($\sigma(f_r)$) = $\{k_r\}$, and the k_r'th argument of f_r is an idle stateholder, and suppose therefore that f_r is the root of a redex $g_r : L_r \to G_{\mathrm{N}}$ for some rule $D_r = (P_r, root_r, Red_r, Act_r)$. Suppose the stateholder argument of D_r is s_r, and that D_r redirects s_r. Suppose for each redirection $(a, b) \in Red_r$, either b is non-idle or b is in Act_r.

Suppose for f_s, all of the (Map($\sigma(f_s)$) – State($\sigma(f_s)$)) arguments of f_s are idle constructors, that State($\sigma(f_s)$) = $\{k_s\}$, and the k_s'th argument of f_s is an idle stateholder, and suppose that f_s is the root of a redex $g_s : L_s \to G_{\mathrm{N}}$ for a resuspending rule $D_s = (P_s, root_s, Red_s, \varnothing)$. Suppose for the root redirection $(root_s, b_s) \in Red_r$, b_s is non-idle (as is the case for a resuspending rule).

Suppose $g_r(s_r) = v = g_s(s_s)$.

Let

$$\begin{aligned}
&\mathrm{MapNodes}_r = \{x \in G_{\mathrm{N}} \mid x = \alpha(f_r)[k] \text{ for some } k \in \mathrm{Map}(\sigma(f_r))\}\\
&\mathrm{MapNodes}_s = \{x \in G_{\mathrm{N}} \mid x = \alpha(f_s)[k] \text{ for some } k \in \mathrm{Map}(\sigma(f_s))\}\\
&\mathrm{RedNodes}_r = \{x \in G_{\mathrm{N}} \mid x = g_r(a) \text{ for some } (a, b) \in Red_r\}\\
&\mathrm{RedNodes}_s = \{x \in G_{\mathrm{N}} \mid x = g_s(root_s) \text{ for } (root_s, b_s) \in Red_r\}\\
&\mathrm{LActNodes}_r = g_r(Act_r \cup \{b \mid (a, b) \in Red_r, a \in Act_r, b \in L_r\})
\end{aligned}$$

Let H_r be obtained by rewriting the redex rooted at f_r in G_{N}, via the usual phases $g_r' : P_r \to G_{\mathrm{N}r}'$, $g_r'' : P_r \to G_{\mathrm{N}r}''$, $g_r''' : P_r \to H_r$, and associated i and r maps. Let H_s be obtained by rewriting the redex rooted at f_s in G_{N}, via the usual phases $g_s' : P_s \to G_{\mathrm{N}s}'$, $g_s'' : P_s \to G_{\mathrm{N}s}''$, $g_s''' : P_s \to H_s$, and associated i and r maps.

Let

$$\mathrm{NN}_r = (\mathrm{LActNodes}_r \cap \mathrm{MapNodes}_s) - \mathrm{RedNodes}_r$$

Then

(1) $\mathit{Not}_r = r_{G_\mathrm{N},H_r}(\mathrm{NN}_r) = i_{G_\mathrm{N},H_r}(\mathrm{NN}_r)$ contains only active constructors.

Let M_r be the result of performing notifications from all nodes in Not_r.

Then

(2) (a) $r_{G_\mathrm{N},M_r}(f_s) = i_{G_\mathrm{N},M_r}(f_s)$ is an active function node of M_r, such that all the $\mathrm{Map}(\sigma(r_{G_\mathrm{N},M_r}(f_s))) - \mathrm{State}(\sigma(r_{G_\mathrm{N},M_r}(f_s)))$ arguments of $r_{G_\mathrm{N},M_r}(f_s)$ are idle constructors, and the $\mathrm{State}(\sigma(r_{G_\mathrm{N},M_r}(f_s)))$ argument of $r_{G_\mathrm{N},M_r}(f_s)$ is a non-idle or $\perp$- node, hence is the redex of a potential suspension step in M_r.

(b) $r_{G_\mathrm{N},H_s}(f_r) = i_{G_\mathrm{N},H_s}(f_r)$ is an active function node of H_s, and

$$h_s = r_{G_\mathrm{N},H_s} \circ g_r : L_r \to H_s$$

is a redex for D_r, such that all the $(\mathrm{Map}(\sigma(r_{G_\mathrm{N},H_s}(f_r))) - \mathrm{State}(\sigma(r_{G_\mathrm{N},H_s}(f_r))))$ arguments of $r_{G_\mathrm{N},H_s}(f_r)$ are idle constructors, and the $\mathrm{State}(\sigma(r_{G_\mathrm{N},H_s}(f_r)))$ argument of $r_{G_\mathrm{N},H_s}(f_r)$ is an idle stateholder, hence is the redex of a potential rewrite in H_s.

Let K_r be obtained from M_r by performing a suspension step rooted at $r_{G_\mathrm{N},M_r}(f_s)$ in M_r. Let M_s be obtained from H_s by rewriting the redex rooted at $r_{G_\mathrm{N},H_s}(f_r)$ in H_s, via the usual phases $h_s' : P_r \to H_s'$, $h_s'' : P_r \to H_s''$, $h_s''' : P_r \to M_s$, and associated i and r maps.

Then

(3) $\mathit{Not}_s = r_{G_\mathrm{N},M_s}(\mathrm{NN}_r) = i_{G_\mathrm{N},M_s}(\mathrm{NN}_r)$ contains only active constructors.

Let K_s be the result of performing notifications from all nodes in Not_s.

Then

(4) K_r and $\psi(K_r) \subseteq K_s$ are marking preserving isomorphic via a map $\psi : K_r \to K_s$; and the only things in $K_s - \psi(K_r)$ are the extra idle garbage node $i_{G_\mathrm{N},K_s}(f_s)$ and extra normal garbage arcs $(i_{G_\mathrm{N},K_s}(f_s)_l,\ r_{G_\mathrm{N},K_s}(\alpha(f_s)[l]))$ for $l \in A(\sigma(f_s))$, which have no counterparts in K_r.

Proof. Mercifully, we can reuse much of the proofs of previous lemmas with minor alterations, so we will be fairly brief. Consider performing the resuspension rewrite to produce H_s. Aside from technical details of disjoint unions etc., the only difference between G_N and H_s is that the node f_s and out-arcs $((f_s)_l,\ \alpha(f_s)[l])$ for $l \in A(\sigma(f_s))$, which are active and normal respectively and all live in G_N, become idle and normal respectively and all garbage in H_s; and in H_s, there is a new node $r_{G_\mathrm{N},H_s}(f_s)$ once suspended with the same symbol as f_s, and new arcs $(r_{G_\mathrm{N},H_s}(f_s)_l,\ r_{G_\mathrm{N},H_s}(\alpha(f_s)[l]))$ for $l \in A(\sigma(f_s))$, all normal arcs except for $(r_{G_\mathrm{N},H_s}(f_s)_{k_s},\ r_{G_\mathrm{N},H_s}(v))$ which is a notification arc. Under the circumstances, it is clear that the map r_{G_N,H_s} extends to an injective homomorphism, which is marking preserving, except on f_s and its stateholder out-arc. (2).(b) is now clear.

Let us call r_{G_{N},H_s}, θ for short. As we did in lemmas 7.4 and 7.5, we can perform the rewrites according to D_r of the redexes rooted at $f_r \in G_{\mathrm{N}}$ and $r_{G_{\mathrm{N}},H_s}(f_r) \in H_s$ in parallel, constructing injective "almost" marking preserving homomorphisms phase by phase. So M_s and H_r are related by an injective mapping $\theta''' : H_r \to M_s$, which fails to be a marking preserving isomorphism by a node and arc marking and in M_s an extra node and its out-arcs which are garbage.

The rest is relatively straightforward. As in lemma 7.5, the D_r rewrite (whatever the order of rewriting), may activate some $\mathrm{MapNodes}_s$ nodes, but these can only be previously idle constructors as the stateholders are assumed shared, so the awkward cases treated in lemma 7.5 do not arise, and (1), (2).(a) and (3) are now clear.

When the notifications $H_r \to M_r$ and $M_s \to K_s$ are performed, a similar relationship, let us call it $\theta'''' : M_r \to K_s$, holds as for H_r and M_s. Finally when the suspension step $M_r \to K_r$ is performed, there results the map $\psi : K_r \to K_s$, which is a marking preserving injective homomorphism, such that the only things in K_s falling outside the image of ψ are the previously noted garbage node $i_{G_{\mathrm{N}},K_s}(f_s)$ and its out-arcs. We are done. ☺

8 The Church-Rosser Theorem

The general idea for proving the theorem is to fill in the Church-Rosser diamond between two preexecutions that diverge from G_0, by continually adding instances of subcommuting squares, until one reaches the diagonally opposite point. See Fig. 11. Two things prevent this from being entirely straightforward. The first is that MONSTR is not a free rewriting system, but one whose strategy is programmed by the markings, so we have to be sure that the squares we fill in are permitted by the strategy. The second is that some of the subcommuting squares do not subcommute quite "on the nose", which for our purposes means up to marking preserving isomorphism. This causes the construction to potentially flake into separate sheets at various points[1] — some of these sheets may recombine later in the construction, as later activations and notifications cause differing markings to resynchronise; see Fig. 11 again. Given the flaking, we need to do two things. Firstly we need to be sure that despite it we can still get to the diagonally opposite corner, up to some markings and some garbage. What we call the handrail construction is crucial here. Secondly, we need to relate the graphs occurring at the same coordinates of the various flakes, since there is no guarantee that all the flakes will have recombined by the time we finish filling in all the sheets.

Lemma 8.1 Let $\mathcal{G}_{\mathrm{N}} = [G_0 \ldots G_{\mathrm{N}}]$ be a transitive coercing preexecution of a MONSTR system $\mathcal{R}$. Suppose the hypotheses of one of lemmas 7.1 – 7.7 apply. Then with the notation used in that lemma (or the obviously analogous notation in the subscripts), we have: $\mathcal{K}_1 = [G_0 \ldots G_{\mathrm{N}} \ldots K_1]$ is a transitive coercing preexecution of $\mathcal{R}$ iff $\mathcal{K}_2 = [G_0 \ldots G_{\mathrm{N}} \ldots K_2]$ is a transitive coercing preexecution of $\mathcal{R}$.

[1] Readers familiar with complex analysis may amuse themselves by contemplating an analogy with Riemann surfaces. It is as though the graph structure corresponded to the magnitude of an analytic function and the markings and garbage corresponded to the phase; and there was a singularity in the "bad cell" of Fig. 11.

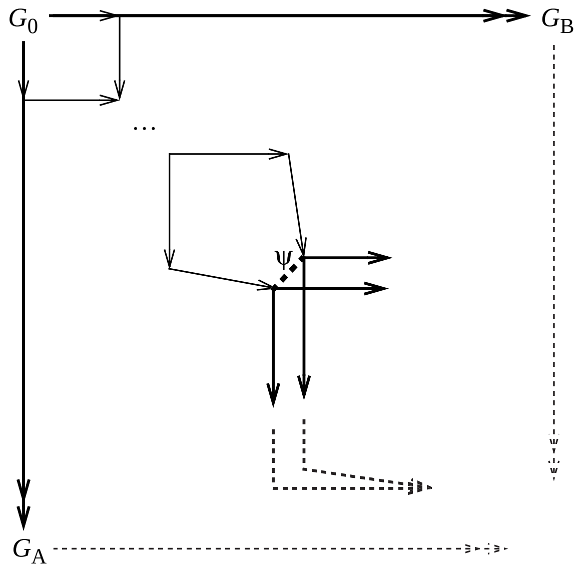

Fig. 11 Filling in the Church-Rosser diamond.

Proof. Beyond the facts established already in the lemmas mentioned, all we need to check, is that for the execution steps discussed in each particular case, conformance to the execution strategy of suspending MONSTR semantics in definition 3.1 holds for one order of execution steps iff it holds for the other; but it is rather obvious that this is so. (Note in particular that the lemmas involving rewrite steps are insensitive to whether or not a default rule is being used when a normal rule is demanded by definition 3.1.) ☺

Lemma 8.2 Let $\mathcal{G}_{\mathrm{N}} = [G_0 \dots G_{\mathrm{N}}]$ be a transitive coercing preexecution of a MONSTR system $\mathcal{R}$. Suppose the hypotheses of one of lemmas 7.1 – 7.7 apply. With the notation used in that lemma (or the obviously analogous notation in the subscripts), let $\mathcal{K}_1 = [G_0 \dots G_{\mathrm{N}} \dots H_1 \dots K_1]$ and $\mathcal{K}_2 = [G_0 \dots G_{\mathrm{N}} \dots H_2 \dots K_2]$ be the transitive coercing pre-executions of $\mathcal{R}$ constructed in the relevant lemma, and let $\psi : K_1 \to K_2$ be the marking preserving isomorphism (or *not quite* marking preserving isomorphism), constructed in that lemma. Then

(1) An active node t of H_i has $r_{H_i,K_i}(t)$ active in K_i iff for no graph M_i in $H_i \dots K_i$ is $r_{H_i,M_i}(t)$ the chosen root. For such a t, $r_{H_i,M_i}(t) = i_{H_i,M_i}(t)$ for every M_i.

(2) For every active node t of H_1 such that $r_{H_1,K_1}(t)$ is active in K_1, $\psi \circ r_{H_1,K_1}(t)$ is active in K_2.

(3) For every active node t of H_2 such that $r_{H_2,K_2}(t)$ is active in K_2, $r_{H_2,K_2}(t)$ is in the range of ψ, and $\psi^{-1} \circ r_{H_2,K_2}(t)$ is active in K_1.

(4) A node $x \in K_1$ is non-idle iff $\psi(x) \in K_2$ is non-idle.

Proof. This requires an examination of the individual cases in lemmas 7.1 – 7.7. For the notification/notification, suspension/suspension and notification/suspension cases, things are helped by the facts that: (a) $H_i \ldots K_i$ consists of at most one step; (b) the graph structure is not changed by any of the steps involved, and; (c) ψ is a marking preserving isomorphism.

Thus for the notification/notification case, the only active markings removed in $H_i \ldots K_i$ are the ones on the notification roots, hence (1), (2) and (3). Similarly for the suspension/suspension case. Also for the notification/suspension case, except that $H_t \ldots K_t$ might be trivial.

For the rewrite/notification case if the notification is last, its root is the only node whose active marking is removed in $H_f \ldots K_f$; and if the rewrite is last, likewise for $H_t \ldots K_t$; with the observation that if the rewrite reactivated the notification root, a final notification is there to fix things.

For the rewrite/suspension case, ψ may no longer be marking preserving, but in any case the discrepancy amounts to a differing number of suspensions on the suspension root, which does not affect the conclusions. Otherwise the reasoning is much as in the previous case.

For the rewrite/rewrite case, there are two subcases: (A) dealt with in lemma 7.6, covering two rewrites which either do not belong to the same critical cone or if they do, their use of the common stateholder is read-only. In this case, ψ may no longer be marking preserving, and we need to check more carefully that the required conclusions hold. The failure of ψ to be marking preserving concerns two kinds of nodes suspended on the stateholder: either contractum nodes introduced during one or other rewrite, or existing nodes of G_{N}. Both kinds of nodes are suspended in H_i on the shared stateholder (apart from the contractum nodes which are due to be introduced in the D_j rewrite which do not exist yet in H_i), so cannot cause the conclusions to fail. For other nodes we have the conclusions because neither a rewrite nor a notification can remove an active marking, except from its own root.

The final subcase is (B), dealt with in lemma 7.7, covering two rewrites which belong to the same critical cone and one of which is a resuspension. The reasoning here is similar to but simpler than in the previous subcase, as none of the awkward circumstances of that subcase arise because of the simple nature of resuspending rules. It is also the only case in which the direction in which ψ goes matters, as $\psi : K_r \rightarrow K_s$ is not onto. This makes it necessary to have differing clauses (2) and (3) in the lemma.

Clause (4) follows from a trivial inspection of the conclusions of lemmas 7.1 – 7.7 ☺

Lemma 8.2 does little above stating the obvious, namely that performing one execution step does not destroy other potential sites for performing execution steps. Still there is more to it than meets the eye. For instance, the “iff” of the first clause does not hold for other semantic models that have been considered in this series of papers. More important, is the granularity of the implications in the clauses (2) and (3), which directly relate properties of H_i to ones of K_i, without pausing to involve any of the intermediate graphs along the way. Contemplating going only in small steps, i.e. proving a series of finer grained implications that relate the active nodes of each graph in $H_i \dots K_i$ to the active nodes of its successor is perfectly possible, but would cause clauses (2) and (3) to fail. For example, the differences in markings that we were able to ignore in discussing rewrite/rewrite subcase (A), would become visible to the hypotheses of these mini-implications, and we would not be able to cross the gap described by the ψ functions. Not being able to do so would cause our proof strategy for the Church-Rosser theorem to fail in cases where ψ refers to an exceptional case, as we would not be sure that any execution step that we needed to perform on one sheet, could be mimicked on another. On the other hand, we only *need* to do this for execution steps that occurred in the *original* preexecutions (and some necessary notifications), not other ones that merely happen to be permitted as a result of the subcommutativity cells that we construct in the process[1]. This is exploited by the handrail construction in the main theorem, which is vital for propagating the proof in the presence of the flaking, and shows why the correct granularity of the implications is so important here.

Now for the main result.

Theorem 8.3 Let R be a MONSTR system. Suppose

(a) For every two distinct normal rules $D_i \neq D_j$ in R, where $D_i = (L_i \subseteq P_i, root_i, Red_i, Act_i)$ and $D_j = (L_j \subseteq P_j, root_j, Red_j, Act_j)$, we have that L_i and L_j are not graph unifyable, (i.e. there is no G and no $g_i : L_i \to G$, $g_j : L_j \to G$, with $g_i(root_i) = g_j(root_j)$). Similarly for every two distinct default rules.

(b) For every rule of R, every redirection is to an activated node or non-idle node.

(c) Every critical cone that arises in any execution of R is safe.

Let $\mathsf{G}_\mathrm{A} = [G_0, \dots, G_\mathrm{A}]$ and $\mathsf{G}_\mathrm{B} = [G_0, \dots, G_\mathrm{B}]$ be transitive coercing preexecutions of R. Then there are extensions $\mathsf{G}_\mathrm{X} = [G_0, \dots, G_\mathrm{A}, \dots, G_\mathrm{X}]$ and $\mathsf{G}_\mathrm{Y} = [G_0, \dots, G_\mathrm{B}, \dots, G_\mathrm{Y}]$ of G_A and G_B such that

(1) There are subgraphs $G_\mathrm{X}{}^*$ of G_X, and $G_\mathrm{Y}{}^*$ of G_Y, and a graph structure isomorphism $\Theta_\mathrm{XY}{}^* : G_\mathrm{X}{}^* \to G_\mathrm{Y}{}^*$.

(2) All nodes and arcs in $G_\mathrm{X} - G_\mathrm{X}{}^*$, and in $G_\mathrm{Y} - G_\mathrm{Y}{}^*$ are garbaged roots (and their out-arcs) of resuspending rewrites. In particular, $G_\mathrm{X}{}^*$ contains LSG(G_X) and $G_\mathrm{Y}{}^*$ contains LSG(G_Y).

Proof. Let us set the scene informally for the moment. We assume that $\mathsf{G}_\mathrm{A} = [G_0, \dots, G_\mathrm{A}]$ consists of A steps, and $\mathsf{G}_\mathrm{B} = [G_0, \dots, G_\mathrm{B}]$ of B steps. So the Church-Rosser diamond divides up into a grid of A.B cells. We will coordinatise the diamond, refering to a cell by its coordinates (a, b), where $1 \le a \le \mathrm{A}$, and $1 \le b \le \mathrm{B}$, so roughly speaking, cell (a, b) produces graph (or graphs, because of the flaking) $G_{(a,\,b)}$, from $G_{(a-1,\,b-1)}$, where the graphs of G_A and G_B are identified with graphs $G_{(a,\,b)}$ such that one or other coordinate is zero.

[1] To put it more brusquely, we do not need K_1 and K_2 to be bisimilar.

The filling in of the Church-Rosser diamond will attempt to work by well founded recursion on a partial order $<$ over the coordinates, where $(a, b) < (a', b')$ iff $a < a'$ and $b \leq b'$, or $a \leq a'$ and $b < b'$. We will call this setup the base coordinate system and base partial order. We can fill a cell, i.e. construct the graph(s) at its highest coordinates by using a subcommutativity lemma, once all the graphs at lower coordinates w.r.t. $<$ have been constructed. This process requires the consideration of two technical points.

The first technical point is that not all the subcommutativity lemmas close their cells (i.e. construct $H_i \ldots K_i$ say) using a single execution step. In one case there is no step at all which is no problem at all, since we just introduce a suitable marking preserving isomorphism to act as a step, but in others several steps are needed. The latter possibility requires the consideration of two points in and of itself.

On the one hand we need to amplify the indexing system in the Church-Rosser diamond to cope with these extra graphs — this we can do for example by introducing a decimal point and naming the first graph on the way from $G_{(a,\, b)}$ to say $G_{(a,\, b+1)}$ as $G_{(a,\, b.1)}$, the second $G_{(a,\, b.2)}$ and so on; introducing a second decimal point if perhaps the step $G_{(a+1,\, b.1)}$ to $G_{(a+1,\, b.2)}$ say, subsequently needs to be subdivided itself, thus $G_{(a+1,\, b.1.1)}$ etc. The partial order $<$ extends pointwise and lexicographically to these pairs of index sequences in the obvious way: the least significant index variable changes fastest in both dimensions. We will call this generalised setup, the refined coordinate system and refined partial order.

On the other hand, and more seriously, this process potentially causes the construction to diverge as the hierarchy of newly introduced steps might potentially have an unbounded number of levels, and might require at least some sort of limit treatment. However we notice that all the additional steps introduced, in any of the subcommutativity lemmas, are always notifications. It takes only a moment to check that notifications subcommute on the nose with all other execution steps, so the phenomenon we feared does not happen. In fact bearing in mind that the worst behaved of the subcommutativity lemmas is 7.6, where up to $2\mathrm{M}$ notifications are introduced in either closing sequence of the cell (where M is the maximum cardinality of the MapNodes set of any function used during either G_{A} or G_{B}), we can calculate that when an execution sequence of length A, is projected over another of length B, we get a closing sequence $[G_{(\mathrm{A},\, 0)}=G_{\mathrm{A}}, \ldots, G_{\mathrm{X}}=G_{(\mathrm{A},\, \mathrm{B})}]$ of length at most $2\mathrm{M}.\mathrm{A}.\mathrm{B}$.

The second technical point concerns the flaking of the construction into sheets, which we have already mentioned. To keep things reasonable, for cases where the map ψ constructed in any of the subcommutativity lemmas is actually a genuine marking preserving isomorphism, let us agree that the cell in question is closed by a single graph, rather than an isomorphic pair. We call such cells good cells; others will be bad cells. Maximising the number of good cells in the construction requires that we view execution steps as being defined up to marking preserving isomorphism in general, and our notation throughout the paper has been fussy enough to transplant without change to this approach. More specifically, given the specific constructions for the various execution steps defined in Section 3, which prescribe how to obtain from the starting graph G the result graph H and the injection and redirection functions $i_{G,H}$ and $r_{G,H}$, if W is a graph which is marking preserving isomorphic to H, via an isomorphism $\theta : H \to W$, then W is an equally acceptable result of the execution step, provided we equip it with the injection and redirection functions $i_{G,W} = \theta \circ i_{G,H}$ and $r_{G,W} = \theta \circ r_{G,H}$. This allows us to close all the good subcommutativity cells on the nose, and bad cells are consequently ones which cannot be closed on the nose despite marking preserving isomorphisms. Since all cases involving notifications are unproblematic, we see that all the extra steps that get introduced in closing the more complicated cells (whether good or bad), do not introduce additional flaking, and thus we easily estimate that the maximum number of distinct non marking preserving isomorphic graphs we can generate at coordinate position (A, B) is overestimated by flaking once per base coordinate cell on each sheet generated. The total thus generated being the binomial coeficient $(^{\mathrm{A+B}}_{\mathrm{A}})$.

In principle the flaking can interfere rather badly with the coordinate construction, as graphs on different sheets at a given coordinate position might conceivably spawn incompatible subsequent behaviour, leading to ambiguity in the construction of later coordinates. But we will in fact see that such unwelcome behaviour will not arise.

With the above preamble, we turn to the more formal construction. We start with the special case in which no flaking occurs; so all cells are good. This proceeds by a simultaneous recursive construction of, and induction on, the refined coordinate system and partial order.

If both G_{A} and G_{B} are trivial, consisting of G_0 only, then the theorem holds trivially, taking $G_{\mathrm{A}} = G_{\mathrm{B}} = G_{\mathrm{X}} = G_{\mathrm{Y}}$, and $\Theta_{\mathrm{XY}}*$ to be the identity. If only one of G_{A} or G_{B} is trivial, let it be G_{A}; then take $G_{\mathrm{B}} = G_{\mathrm{X}} = G_{\mathrm{Y}}$, with $\Theta_{\mathrm{XY}}*$ an identity as before. There remains the non-trivial case. This involves considering the situation illustrated in Fig. 12.

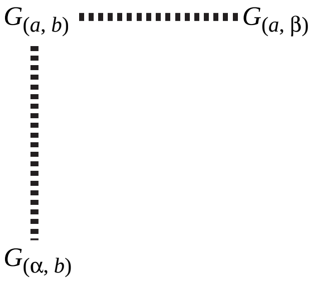

Fig. 12 Lines at the start of an induction step.

The induction hypothesis consists of the following clauses.

(i) The construction thus far, has constructed the refined coordinate system to include all coordinates lower than (a, b), (α, b), (a, β) in the refined partial order.

(ii) In the refined coordinate system thus far, α is the successor of a in the vertical direction, and β is the successor of b in the horizontal direction[1].

[1] That is to say, any cell to the left of [Fig. 12] has its top and bottom at coordinates p and q, where $p \le a$, and $\alpha \le q$ in the refined partial order. Similarly for any cell above Fig. 12.

(iii) The construction thus far is complete up to at least coordinate positions (a, b), (α, b), (a, β), in that it has built the graphs required at these coordinate positions.

(iv) The dashed lines connecting a graph $G_{(a, b)}$ to a graph $G_{(\alpha, b)}$ or a graph $G_{(a, \beta)}$ are of two kinds. Each is either: (a), a marking preserving isomorphism; or (b), an execution step.

The rest is relatively straightforward. We just have to complete the square in Fig. 13, and re-establish the induction hypothesis for the new graph(s) $G_{(\alpha, \beta)}$ (and perhaps others), and arrows $G_{(\alpha, b)} \cdots G_{(\alpha, \beta)}$ and $G_{(a, \beta)} \cdots G_{(\alpha, \beta)}$ illustrated.

$$\begin{tikzcd}
G_{(a, b)} \arrow[r, dash, dashed] \arrow[d, dash, dashed] & G_{(a, \beta)} \arrow[d, dashed] \\
G_{(\alpha, b)} \arrow[r, dashed] & G_{(\alpha, \beta)}
\end{tikzcd}$$

Fig. 13 The objective of an induction step.

The two different kinds of line generate three cases to consider, by symmetry.

If both lines are marking preserving isomorphisms, then we let $G_{(\alpha, \beta)}$ be yet another marking preserving isomorphic copy of $G_{(a, b)}$ and $G_{(\alpha, b)}$ and $G_{(a, \beta)}$, and we let the required arrows be the obvious marking preserving isomorphisms. If only one line is a marking preserving isomorphism, we complete the square with a marking preserving isomorphism and a marking preserving isomorphic copy of whatever execution step the other line was, so that the isomorphisms and execution steps respectively face each other across the square. In these cases, the coordinate system does not require further refinement, so all clauses of the induction hypothesis are easily seen to be preserved.

Now suppose both lines are execution steps. Then either the same active node is the chosen root of both steps in $G_{(a, b)}$ or not. If yes, then both lines are the same step, since if either is a notification or suspension, the other can only be the same; and if either is a rewrite, then hypothesis (a) of the theorem ensures that in MONSTR's prioritised rule selection strategy, only one rule will match at the chosen root so it must be the same one in both cases. At any rate in any of these cases, the cell can be closed with a marking preserving isomorphic copy $G_{(\alpha, \beta)}$, of the already marking preserving isomorphic $G_{(\alpha, b)}$ and $G_{(a, \beta)}$; the required arrows being the obvious isomorphisms. As previously, all clauses of the induction hypothesis are easily seen to be preserved.

Suppose then that the roots are different. In such a case we call on the subcommutativity lemmas to help. Where there are two rewrites involved and they are not in the same critical cone, then lemma 7.6 applies; if both are in the same critical cone, the safety hypothesis (c) of the theorem, assures us that either both uses of the stateholder are read-only so that lemma 7.6 applies again, or that one rewrite is a resuspension rewrite, and then lemma 7.7 applies. Apart from that, the subcommutativity lemmas apply unreservedly. In each possible case, the relevant lemma allows us to complete the square on the nose, by assumption.

In the case of notification/suspension, if the suspension comes second, it may become trivial. In this case, the corresponding arrow of the construction is a marking preserving isomorphism, dealt with rather as above. In other cases, the arrows may contain several execution steps. In this case we refine the coordinate system further, for example on the bottom arrow, making the successors of $G_{(\alpha,\, b)}$ in turn, $G_{(\alpha,\, b.1)}$, $G_{(\alpha,\, b.2)}$, $G_{(\alpha,\, b.3)}$ etc., until the last graph is called $G_{(\alpha,\, \beta)}$ as in previous cases. This establishes that the induction hypothesis is preserved. This completes the treatment of the special case.

For the general case, we start by using the special case as far as we can. We fill in something like the shaded region of Fig. 14. The two not quite enclosed squares are bad cells that do not close on the nose. Extending the construction into the area south east of such cells, requires the handrail construction, a kind of strip lemma, next.

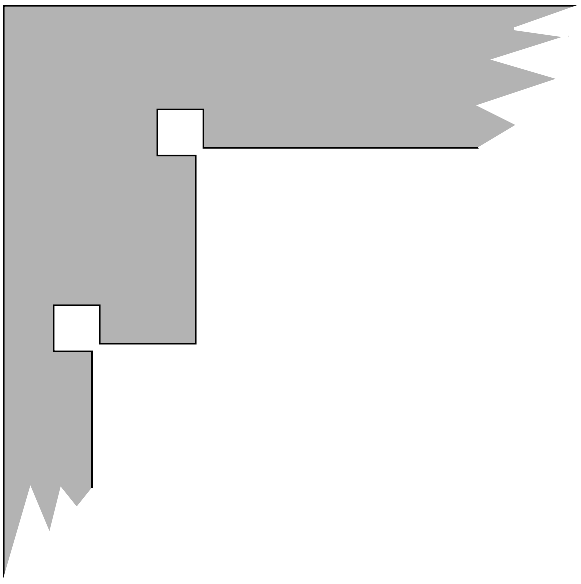

Fig. 14 Bad cells block the Church-Rosser construction.

Consider the situation in Fig. 15. $YXG_{\mathrm{N}}PR$ is an instance of a bad cell, while $PQRS$ is a strip filled entirely by good cells. We claim that there is an extension of $G_0G_{\mathrm{N}}XY$ to $G_0G_{\mathrm{N}}XYZ$, such that the natural analogue of lemma 8.2 holds. Thus let $\psi_{RY} : R \rightarrow Y$ be the ψ map provided by the appropriate subcommutativity lemma. (A completely analogous result holds if $\psi_{YR} : Y \rightarrow R$ is given instead). We will prove that there is a map $\psi_{SZ} : S \rightarrow Z$ (resp. $\psi_{ZS} : Z \rightarrow S$) such that the following hold.

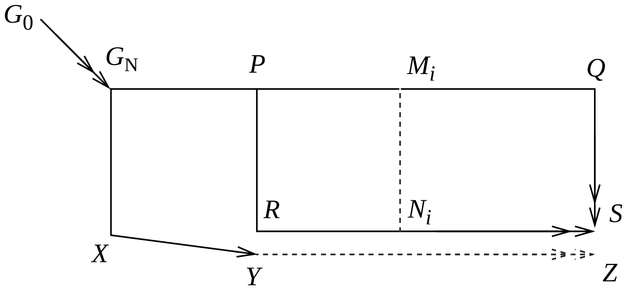

Fig. 15 The handrail construction.

(1) *RS* and *YZ* are of the same length, and contain similar execution steps in the same order; in particular, rewrites occur at the same places in both.

(2) $\psi_{SZ} : S \rightarrow Z$ (resp. $\psi_{ZS} : Z \rightarrow S$) is a graph structure isomomorphism except when YXG_NPR is an instance of lemma 7.7, when there is an extra garbage node and its out-arcs not related by ψ_{SZ} (resp. ψ_{ZS}) to the other graph.

(3) An active node t of Q has $r_{Q,S}(t)$ active in S iff for no graph B_j in $Q \ldots S$ is $r_{Q,B_j}(t)$ the chosen root. For such a t, $r_{Q,B_j}(t) = i_{Q,B_j}(t)$ for every B_j.

(4) For every active node t of Q such that $r_{Q,S}(t)$ is active in S, $\psi_{SZ}(t)$ is active in Z, (resp. $r_{Q,S}(t)$ is in the range of ψ and $\psi^{-1}{}_{ZS}(r_{Q,S}(t))$ is active in Z) .

(5) A node $t \in S$ is non-idle iff $\psi_{SZ}(t)$ (resp. $\psi^{-1}{}_{ZS}(t)$) is non-idle.

Now if YXG_NPR is an instance of lemma 7.7, the only difference between R and Y is some garbage. By the soundness results for garbage in Section 5, we can make *YZ* the same as *RS*, up to this garbage, and the claim holds.

Otherwise the claim is substantiated by induction on the length of *PQ*. If this is trivial then the claim holds trivially by lemma 8.2. Otherwise we go by cases on the next step in *PQ*, say $M_i \rightarrow M_{i+1}$. Let N_i close the span of PM_i and *PR* as in Fig. 15, and let N_{i+1} close PM_{i+1} and *PR*. $N_i \ldots N_{i+1}$ may consist of more than a single step. Let W_i be the graph of *YZ* corresponding to N_i by induction hypothesis. Our job is to construct $W_i \ldots W_{i+1}$ corresponding to $N_i \ldots N_{i+1}$.

If $M_i \rightarrow M_{i+1}$ is an isomorphism step, it subcommutes with everything on the way from $M_i \ldots N_i$, so step $N_i \rightarrow N_{i+1}$ in *RS* is an isomorphism, and we make the corresponding step $W_i \rightarrow W_{i+1}$ in *YZ* an isomorphism too. If the step $M_i \rightarrow M_{i+1}$ is a notification, this commutes with everything, and by the hypotheses, we can make $W_i \rightarrow W_{i+1}$ in *YZ* a notification too. This works, provided nothing in $M_i \ldots N_i$ is the *same* notification, otherwise we have isomorphisms again from a certain point on. In such a case, the corresponding node in W_i is already idle by the induction hypothesis, and we make $W_i \to W_{i+1}$ an isomorphism too. It is easy to see that the claim holds.

If the step $M_i \to M_{i+1}$ is a suspension, then either some step in $M_i \dots N_i$ is a notification that trivialises the suspension, in which case the induction hypothesis assures us that the corresponding node in W_i is already idle and we make $W_i \to W_{i+1}$ an isomorphism, or not. If not, then apart from notifications and isomorphisms, of which there may be several, $M_i \dots N_i$ contains exactly one rewrite or suspension (this is because we are discussing a strip with exactly one bad cell on the left). If it is a suspension, the two of them either subcommute, and the corresponding step $W_i \to W_{i+1}$ becomes a suspension, or they are the same, in which case we get an isomorphism. If it is a rewrite, then the rewrite and suspension subcommute (on the nose, because we assume *PQRS* is covered by good cells), with perhaps the addition of some notifications as per lemma 7.5. The extra notifications make $N_i \dots N_{i+1}$ into a multistep sequence but are easy to deal with. Hence we construct $W_i \dots W_{i+1}$ as a sequence consisting of a suspension, followed by the requisite notifications. Establishing the claim is easy.

Finally the step $M_i \to M_{i+1}$ might be a rewrite. This will subcommute with any isomorphisms or notifications in $M_i \dots N_i$ giving a corresponding step for $W_i \to W_{i+1}$, leaving suspensions and rewrites to consider. If we have a suspension, the situation is as in the previous paragraph with the roles reversed. If we have a rewrite, then either the two of them are the same, and we have the inevitable isomorphisms, or not. If not then by assumption *PQRS* is covered with good cells, so the rewrites subcommute with the addition of the appropriate notifications, and $W_i \dots W_{i+1}$ becomes a sequence consisting of some notifications, a rewrite, and some more notifications. The notifications are again easy to deal with and it is easy to see that all the clauses of the induction hypothesis are properly established. This completes the inductive step for the construction of *YZ*.

So we have the handrail *YZ*. In itself this is not enough to completely fill in the whole Church-Rosser diamond, as a bad cell to the right of *QS* might block further progress. For this let us examine Fig. 16.

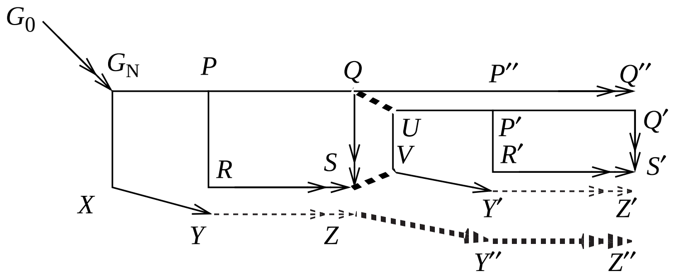

Fig. 16 Two bad cells in the same strip.

There are various possibilities depending on the relationship between QS and UV, where we intend that UV is a subsequence of QS. First of all UV must be “the same” execution step as was $G_{\mathrm{N}}X$. Therefore if QU is non-trivial, it must consist of notifications and isomorphisms, and the same holds if VS is non-trivial. So there are four cases. If both QU and VS are trivial, it is easy to extend the argument used already to extend the handrail YZ to Y'', and then to notice that the same hypothesis applies to P' and Y'' as applies to P' and Y', and thence to continue the extension to Z'', independently of the construction of $Y'Z'$. If QU is trivial but VS is not, we must first construct the handrail $Y'Z'$, then fill in the projection of VS along it (this will contain only good cells), and finally build $ZY''Z''$ which can be done as above. If VS is trivial but QU is not, we must consider the extension of PQ to $P''Q''$ in order that the induction hypothesis for the extension of XYZ to $XYZY''Z''$ does not reveal differing markings on nodes that correspond in U and Z. Since QU consists of isomorphisms and notifications only, only good cells occur between $QP''Q''$ and $UP'Q'$ so this is easy; then the handrail construction proceeds as formerly. If both QU and VS are non-trivial, we must do both things.

Once we can build handrails, we can extend the filling in of the Church-Rosser diamond to the various sheets generated. Roughly speaking, proceeding inwards into the interior from the given preexecutions G_{A} and G_{B}, ensures that we can always make progress even if PQ, $P'Q'$ or $P''Q''$ are themselves partly handrails. (We leave to the interested reader the construction of the partial order, over whose induction the filling in of the Church-Rosser diamond can be more formally described.) Because the relationships between corresponding graphs at the same coordinates on the pairs of sheets we generate are, apart from garbage, graph structure isomorphisms which preserve the idle/non-idle property of nodes, it is clear from definition 3.1 and the hypotheses, that for an arbitrary coordinate position and arbitrary choice of direction of progress east or south, we always perform “the same step” from corresponding roots on different sheets, i.e. either always an isomorphism, or notification, or suspension on the same arguments, or always a rewrite using the same rule. This also ensures that the construction of the refined coordinate system and partial order is unambiguous. Because the original preexecutions are finite, and we flake once per sheet per base cell at most, we complete the whole process in a finite number of iterations. When the construction reaches coordinates (A, B), erasing the isomorphism steps from the extensions to G_{A} and G_{B} generated, gives us the G_{X} and G_{Y} we need.

It remains to construct $G_{\mathrm{X}}*$, $G_{\mathrm{Y}}*$ and $\Theta_{\mathrm{XY}}*$, and show they have the right properties. We construct similar objects for all pairs of graphs on different sheets, for all coordinate positions. Then the ones required will just be a particular case.

If only good cells occur, then everything commutes on the nose, there is only one sheet, and we can take $G_{\mathrm{X}}* = G_{\mathrm{X}}$, $G_{\mathrm{Y}}* = G_{\mathrm{Y}}$ and $\Theta_{\mathrm{XY}}*$ as the obvious identity; the same thing holds for all coordinates.

Otherwise the structure of the sheets and their graphs can be discerned from the collection of bad cells. Each bad cell is identified by the coordinates of its southeast corner, (a, b) say. For each bad cell, one sheet is obtained by going east then south from its northwest corner, call this the + sheet; and the other sheet is given by going south then east, call it the – sheet. Any graph at coordinates (a, b) on any sheet, and the sheet itself, can be named by a function mapping all bad cells into the set $\{+, –\}$, and ignoring the values at all bad cells at coordinates $(a', b') > (a, b)$. The collection of such functions restricted to $(a', b') \leq (a, b)$ names the set of sheets which are distinct at coordinates (a, *b*), so all graphs created can be named $G_{(a,\,b)}{}^{\mathrm{T}}$ where (a, b) gives the coordinates, and T is the function value naming the sheet.

Now we proceed by recursion on the flakings pertaining to bad cells. These are ordered by < which is a finite partial order. Consider a <-maximal bad cell at (α, β) (on some sheet with sheet name T0 which is about to flake for the last time giving sheets T1 and T2, so T1 and T2 are the two possible extensions of the map T0). The construction of the handrails from the two different graphs $G_{(\alpha,\,\beta)}{}^{\mathrm{T1}}$ and $G_{(\alpha,\,\beta)}{}^{\mathrm{T2}}$ involves the construction of homomorphisms ψ as described above. Since similar steps are performed on the handrail and the corresponding normal execution, all the graphs constructed with coordinates $(a, b) > (\alpha, \beta)$, are isomorphic up to markings or garbage. If they are isomorphic up to markings, set $G_{(\alpha,\,\beta)}{}^{\{\mathrm{T0}\},\mathrm{T1}*} = G_{(\alpha,\,\beta)}{}^{\mathrm{T1}}$, $G_{(\alpha,\,\beta)}{}^{\{\mathrm{T0}\},\mathrm{T2}*} = G_{(\alpha,\,\beta)}{}^{\mathrm{T2}}$, and $\Theta_{(\alpha,\,\beta)}{}^{\{\mathrm{T0}\},\mathrm{T1},\mathrm{T2}} : G_{(\alpha,\,\beta)}{}^{\{\mathrm{T0}\},\mathrm{T1}*} \rightarrow G_{(\alpha,\,\beta)}{}^{\{\mathrm{T0}\},\mathrm{T2}*} = \psi$ as constructed by the relevant lemma, and do similarly for coordinates $(a, b) > (\alpha, \beta)$. If they are isomorphic up to extra garbage, colour the relevant garbage node and arcs black (for example) in whichever of $G_{(\alpha,\,\beta)}{}^{\mathrm{T1}}$ or $G_{(\alpha,\,\beta)}{}^{\mathrm{T2}}$ has it, and propagate the colouring via the $(i_{\_,\_}, r_{\_,\_})$ maps to $(a, b) > (\alpha, \beta)$. Set $G_{(\alpha,\,\beta)}{}^{\{\mathrm{T0}\},\mathrm{T1}*}$ = the uncoloured subgraph of $G_{(\alpha,\,\beta)}{}^{\mathrm{T1}}$, $G_{(\alpha,\,\beta)}{}^{\{\mathrm{T0}\},\mathrm{T2}*}$ = the uncoloured subgraph of $G_{(\alpha,\,\beta)}{}^{\mathrm{T2}}$, and $\Theta_{(\alpha,\,\beta)}{}^{\{\mathrm{T0}\},\mathrm{T1},\mathrm{T2}} : G_{(\alpha,\,\beta)}{}^{\{\mathrm{T0}\},\mathrm{T1}*} \rightarrow G_{(\alpha,\,\beta)}{}^{\{\mathrm{T0}\},\mathrm{T2}*} = \psi$ as constructed by lemma 7.7 and restricted to the uncoloured parts of $G_{(\alpha,\,\beta)}{}^{\{\mathrm{T0}\},\mathrm{T1}*}$ and $G_{(\alpha,\,\beta)}{}^{\{\mathrm{T0}\},\mathrm{T2}*}$. Do similarly for coordinates $(a, b) > (\alpha, \beta)$. In this notation {T0} indicates how far the construction has progressed (i.e. the common part of the names of sheets processed), and T1 and T2 refer to the sheets themselves. So much for the base case.

For the recursive step, we have done all of the above for all bad cells at positions $(\alpha', \beta') > (\alpha, \beta)$ and located on the sheets that flake from the current bad cell on the current sheet whose sheet name is T0 say. Let Tx refer to a typical bad cell *successor* of T0, so we have generated partial isomorphisms $\Theta_{(\alpha',\,\beta')}{}^{\{\mathrm{Tx}\},\mathrm{T}i,\mathrm{T}j} : G_{(\alpha',\,\beta')}{}^{\{\mathrm{Tx}\},\mathrm{T}i*} \rightarrow G_{(\alpha',\,\beta')}{}^{\{\mathrm{Tx}\},\mathrm{T}j*}$ say, between graphs on sheets belonging to all bad cell descendants of Tx. We have done this for all bad cell successors Tx of T0, so for a typical $\Theta_{(\alpha',\,\beta')}{}^{\{\mathrm{Tx}\},\mathrm{T}i,\mathrm{T}j}$, the {Tx} is the common part of the names of a set of sheets already processed, such that all the sheet names in the set agree with T0 on bad cells $\leq (\alpha, \beta)$ and with each other on one further bad cell successor Tx of T0.

If (α, β) is not a lemma 7.7 bad cell, the ψ generated for it is bijective, so we extend the construction in the expected way, noting that all cells involved at any coordinate position are graph structure isomorphic, and building the relevant isomorphisms $\Theta_{(a,\,b)}{}^{\{\mathrm{T0}\},\mathrm{T}i,\mathrm{T}j}$ where the notation {T0} refers the common part of the previous {Tx} sets. If (α, β) is a lemma 7.7 bad cell, the relevant garbage node and arc are coloured black, the colouring is propagated wherever it will reach via $(i_{\_,\_}, r_{\_,\_})$ maps, and the subgraphs built previously have to be further restricted to keep them uncoloured, for half of the sheets in question.

Eventually we complete the construction, getting subgraphs and isomorphisms $\Theta_{(a,\,b)}{}^{\{\},\mathrm{T1},\mathrm{T2}} : G_{(a,\,b)}{}^{\{\},\mathrm{T1}*} \rightarrow G_{(a,\,b)}{}^{\{\},\mathrm{T2}*}$, between subgraphs on arbitrary sheets at any given coordinates (a, b), where {} names the set of all sheets, being the empty function.

Now one of the $G_{(\mathrm{A,\,B})}{}^{\{\},\mathrm{T}i*}$, the one pertaining to the graph G_{X}, is the $G_{\mathrm{X}}{}^*$ we seek, and another is $G_{\mathrm{Y}}{}^*$. The graph structure isomorphism $\Theta_{\mathrm{XY}}{}^* : G_{\mathrm{X}}{}^* \to G_{\mathrm{Y}}{}^*$ is the relevant $\Theta_{(\mathrm{A,\,B})}{}^{\{\},\mathrm{T}i,\mathrm{T}j}$. It clearly has the required properties. We are done. ☺

Corollary 8.4 Let R be a MONSTR system. Assume the notation and hypotheses of theorem 8.3. Then there is a preexecution of R, $\mathsf{G}_{\mathrm{Z}} = [G_0, \ldots, G_{\mathrm{Z}}]$ such that

(1) There are subgraphs $G_{\mathrm{X}}{}^{**}$ of $G_{\mathrm{X}}{}^*$ of G_{X}, and $G_{\mathrm{Y}}{}^{**}$ of $G_{\mathrm{Y}}{}^*$ of G_{Y}, and a graph structure isomorphism $\Theta_{\mathrm{XY}}{}^{**} : G_{\mathrm{X}}{}^{**} \to G_{\mathrm{Y}}{}^{**}$.

(2) All nodes and arcs in $G_{\mathrm{X}} - G_{\mathrm{X}}{}^{**}$, and in $G_{\mathrm{Y}} - G_{\mathrm{Y}}{}^{**}$ are garbaged roots (and their out-arcs) of resuspending rewrites. In particular, $G_{\mathrm{X}}{}^{**}$ contains LSG(G_{X}) and $G_{\mathrm{Y}}{}^{**}$ contains LSG(G_{Y}).

(3) There are graph structure isomorphisms $\Theta_{\mathrm{ZX}} : G_{\mathrm{Z}} \to G_{\mathrm{X}}{}^{**}$, and $\Theta_{\mathrm{ZY}} : G_{\mathrm{Z}} \to G_{\mathrm{Y}}{}^{**}$, such that $\Theta_{\mathrm{ZY}} = \Theta_{\mathrm{XY}}{}^{**} \circ \Theta_{\mathrm{ZX}}$.

(4) $\Theta_{\mathrm{ZX}}(\mathrm{LSG}(G_{\mathrm{Z}})) \supseteq \mathrm{LSG}(G_{\mathrm{X}})$ and $\Theta_{\mathrm{ZY}}(\mathrm{LSG}(G_{\mathrm{Z}})) \supseteq \mathrm{LSG}(G_{\mathrm{Y}})$.

Proof. We describe how to construct G_{Z}. View the edges of the various cells on the various sheets constructed in the preceding theorem as the edges of a directed graph. We start from G_0 and head for the diagonally opposite corner by an arbitrary path traversing edges downwards and to the right only (i.e. in the direction of the execution steps), and avoiding resuspending rewrite and suspension step edges; thereby generating a preexecution (with isomorphism steps) $\mathsf{G}_{\mathrm{Z}}{}^-$. It may be impossible to proceed beyond some particular point without traversing a resuspending rewrite or suspension step edge if both outgoing edges of some graph at (α, β) say, are resuspending rewrite or suspension step edges. But rather than doing such a step, we jump by a marking preserving isomorphism to the diagonally opposite corner of the cell in question (α′, β′) say, without performing any execution step at all. The graph of $\mathsf{G}_{\mathrm{Z}}{}^-$ that we have at (α′, β′), differs from the graph in the construction built in theorem 8.3 at the same coordinates, by the absence of some garbage, and by the fact that the root(s) of the resuspending rewrite(s) or suspension step(s) is (are) still active. This cannot prevent subsequently tracking a path towards (A, B) as the $\mathsf{G}_{\mathrm{Z}}{}^-$ graph is at least as ready to perform any execution step as any compatriot in the Church-Rosser diamond at the same coordinates is, and this property persists. We generate none of the garbage nodes and arcs that lemma 7.7 takes pains to describe or that a suspension step might create, and futhermore, if there is a critical cone within the Church-Rosser construction such that some resuspending rewrite is not forced to subcommute with a non-resuspending rewrite, the relevant garbaged resuspension root and out-arcs survive into $G_{\mathrm{X}}{}^*$ and $G_{\mathrm{Y}}{}^*$ because there is no instance of lemma 7.7 in the construction to throw them out. However the construction of $\mathsf{G}_{\mathrm{Z}}{}^-$ omits the creation of such garbage, hence the subgraph of $G_{\mathrm{X}}{}^*$, $G_{\mathrm{X}}{}^{**}$, that we need to relate G_{Z} to may be a proper subgraph of $G_{\mathrm{X}}{}^*$. We now get G_{Z} from $\mathsf{G}_{\mathrm{Z}}{}^-$ by erasing all the isomorphism steps, and the conclusions follow easily, clause (4) in particular following from the soundness of garbage and induction on the length of the preexecution. ☺

9 Conclusions

In the preceding sections, we have described the structure of MONSTR, studied at length in previous papers, and have given it the most elegant of the semantic models considered in this series. Most elegant that is with respect to its having desirable properties, rather than having the simplest description: in fact its description is the most involved of all the models considered, and also the furthest from the term rewriting origins that inspired the original term graph rewriting model ([Barendregt et al. (1987)]). Nevertheless the more complex collection of primitives offered in the model seems to be more than justified by the strong properties it possesses. From **M-IV** we infer a good serialisability property as regards finegrained implementation, and in this paper we showed that the Church-Rosser property holds, despite the lack of exact subcommutativity which looked more than once as though it might bring down the whole enterprise.

Acknowledgements

Most of the work described in this paper was done while the author was on leave at the Computer Science Department of the University of Cyprus at Nicosia, and the Computer Science Department of the University of Pisa in Italy. The hospitality of those departments and the financial support of the Royal Society, the British Council, the Consiglio Nazionale delle Ricerche, and the ERASMUS Staff Mobility Program of the European Commission are gratefully acknowledged.

References

[Barendregt et al. (1987)] Barendregt H.P., van Eekelen M.C.J.D., Glauert J.R.W., Kennaway J.R, Plasmeijer M.J., Sleep M.R., Term Graph Rewriting. *in*: Proc. PARLE-87, de Bakker, Nijman (eds.), LNCS **259**, 141-158, Springer, (1987).

[Banach (1996a)] Banach. R., Transitive Term Graph Rewriting. Inf. Proc. Lett., **60**, 109-114.

[Banach (1996b)] Banach. R., MONSTR I — Fundamental Issues and the Design of MONSTR. J.UCS, **2**, 164-216, (1996). `http://www.iicm.edu/jucs`

[Banach (1997a)] Banach. R., MONSTR II — Suspending Semantics and Independence. J.UCS, **3**, 755-801, (1997). `http://www.iicm.edu/jucs`

[Banach (1997b)] Banach. R., MONSTR III — Finegrained Semantics and Serialisability. *In preparation*, (1997).

[Banach (1997c)] Banach. R., MONSTR IV — Weakly Coercing Semantics and Serialisability for Resilient Systems. *In preparation*, (1997).

[Banach and Papadopoulos (1997)] Banach R., Papadopoulos G., A Study of Two Graph Rewriting Formalisms: Interaction Nets and MONSTR. Journal of Programming Languages, (1997), *to appear.*