MONSTR II — Suspending Semantics and Independence

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

Abstract: The suspending semantic model for the execution of the MONSTR generalised term graph rewriting language is defined. This is the canonical operational semantic model for the MONSTR language. Its correctness with respect to DACTL semantics is discussed, and a number of general theorems on the soundness of suspending executions with respect to DACTL semantics are proved. General theorems are proved about the independence of suspending primitive actions, which are useful in the verification of MONSTR systems.

Key Words: Intermediate Languages, Term Graph Rewriting, MONSTR, Semantic Models.

Category: D.1.3, D.3.1, F.3.2, F.4.2

1 INTRODUCTION

In the first paper in this series, [Banach (1996)], we introduced the MONSTR generalised term graph rewriting language, a sublanguage of the DACTL language, 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 DACTL semantics is desired.

In this paper we examine one of these alternative semantic models, the suspending model, in detail. It is important to state clearly that: **When one is speaking of the MONSTR language, without specifically mentioning any particular semantic model for it, then the suspending model is to be understood**. Thus suspending semantics is the canonical semantics for the language, and all correctness issues refer, more or less directly, to it.

The suspending model differs from the conventional DACTL model insofar as pattern matching of redexes is sensitive to the node and arc markings in the redex. When a redex with non-idle markings on the root symbol's matched arguments is encountered, the rewrite suspends until these arguments become idle. Obviously the suspension phenomenon creates dependencies between rewrites beyond those present in the conventional DACTL semantic model. The correctness problem (with respect to DACTL semantics) thus reduces to the analysis of these dependencies, and of the potential deadlocks that they might create. This is a major concern of this paper. A further topic of interest that we deal with, this time regarding correctness with respect to higher level specifications, is when two execution steps of the model may be interchanged. This is needed when discussing serialisability properties of arbitrary executions of systems.

The rest of this paper is structured as follows. [Section 2] reviews the salient material from **M-I**, the first paper in this series, [Banach (1996)]. Although the present paper is self contained, a reasonable working knowledge of **M-I** will be of benefit to the reader as we do not repeat every useful detail from **M-I** in order to prevent the size of these papers from growing in arithmetic progression. Throughout the remainder of the paper, notation such as **M-I**.11.4 refers to the fourth listed item of Section 11 of **M-I**.

[Section 3] defines suspending semantics precisely. [Section 4] discusses correctness of suspending executions compared with DACTL executions and proves a selection of theorems that relate to correctness for terminating executions. [Section 5] uses some inspiration from [Section 4] to construct an appropriate notion of correctness for non-terminating suspending executions, and proves a correctness theorem. [Section 6] examines the independence problem, and proves a number of results that enable adjacent events in arbitrary executions to be interchanged. These are useful when a standard form for executions of some specific system is desired. They are also useful were one to construct an abstract semantics for suspending MONSTR systems; particularly noteworthy is the way that suspensions introduce a lot of asymmetric conflict into the semantics. [Section 7] concludes.

2 KEY IDEAS FROM MONSTR I

We recall that we deal with term graphs, consisting of nodes x, each labelled with a symbol $\sigma(x)$ of fixed arity $A(\sigma(x))$, and with each symbol $\sigma(x)$ coming from an alphabet $\mathbf{S} = \mathbf{F} \cup \mathbf{C} \cup \mathbf{V}$, being the disjoint union of functions in $\mathbf{F}$, constructors in $\mathbf{C}$, or state-holders in $\mathbf{V}$. Nodes x also have a sequence of out-arcs $\alpha(x)$ to their children where for each $k \in A(x) = A(\sigma(x))$, $\alpha(x)[k]$ is a child node. Likewise nodes carry a node marking $\mu(x) \in \{\varepsilon, *, \#, \#\# \dots \#^n\}$ (idle, active, once twice … n times suspended), and each out-arc is marked by $\nu(x)[k] \in \{\varepsilon, \hat{}\}$ (normal or idle arc, notification arc) so $\nu(x)$ is the sequence of these pertaining to x. These markings influence reduction strategy. A pattern is like a graph but where some leaf nodes may be labelled with Any, where Any $\notin$ $\mathbf{S}$. Such a node is called implicit; others are called explicit. A homomorphism or matching is a symbol/arity/child-respecting node map, but the constraints are imposed only on explicit nodes, so “Any nodes can match anything”. The markings are normally disregarded by homomorphisms (calling them graph structure homomorphisms), unless we say otherwise (calling them marking preserving homomorphisms). [Fig. 1] shows a rather simple graph. We note the concrete syntax v:S in which v is the node while S is the symbol, and that eg. α(q) is the sequence [s, v] so that α(q)[2] = v. Likewise ν(q) = [ε, ε] as we generally do not visibly write the idle node or arc markings in pictures. For the sequel we note that we can suppress the mention of nodes when we draw pictures of graphs and patterns, expressing the sharing directly, but this not possible when we use a linear concrete syntax as we will do in examples later.

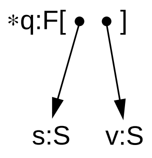

Fig. 1. A graph.

A rule is a quadruple $D = (P, root, Red, Act)$. P is a pattern and *root* is a node of P with $\sigma(root) \in$ **F**. (We say that D is a rule for $\sigma(root)$) The left subpattern L of the rule, is that subpattern accessible from *root*. It must contain all implicit nodes of P and all node and arc markings in L are idle. *Red* is a set of pairs of nodes from $L \times P$, eg. (a, b), with each LHS node a an explicit node of L. The pairs in *Red* form a many–1 relation (i.e. a function) such that distinct LHSs $(a \neq a')$ are labelled differently $(\sigma(a) \neq \sigma(a'))$. This makes the operational semantics of redirection unambiguous. *Act* is also a set of nodes of L. [Fig. 2] shows a picture of a rule, with *root* (and hence L) indicated, *Red* indicated by dashed arrows, and *Act* indicated by $*$-marking the relevant node of L.

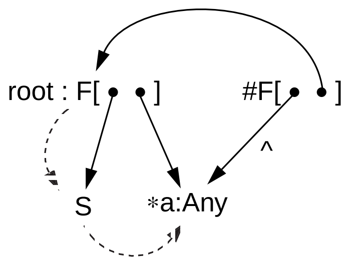

Fig. 2. A rule.

A DACTL execution step proceeds by first nondeterministically choosing a node $t \in G$ which is active in the graph G (i.e. $\mu(t) = *$). The remainder of the step depends on the symbol $\sigma(t)$.

If $\sigma(t) \in$ **C** $\cup$ **V**, then t notifies any of its parents p to which it is connected via a notification (^-marked) in-arc. If they are suspended ($\mu(p) = \#^n$ for $n \geq 1$), their suspension markings are decremented once for each such arc ($\mu(p)$ becomes $\#^{n-m}$ where m is the number of such arcs, with $\#^0 = *$, $\#^{-j} = \varepsilon$), the notification markings are removed from all the relevant arcs, and t becomes idle ($\mu(t)$ becomes ε). If the result of these alterations is called H then there is an obvious injection on nodes $i_{G,H} : G \to H$. [Fig. 3] illustrates such a step assuming 3 is a constructor.

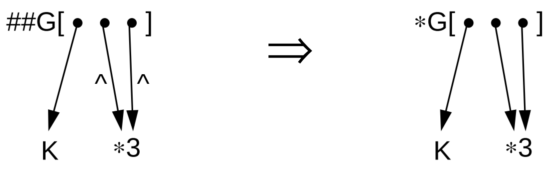

Fig. 3. A notification step.

If $\sigma(t) \in$ **F**, then for MONSTR systems, a normal rule for $\sigma(t)$ is chosen if there is one that will match, otherwise a default rule is chosen. (See below for normal and default rules). A rule D matches a graph G at t iff there is a graph structure homomorphism $g : L \to G$, also known as a redex. For future reference, a redex $g : L \to G$ is said to be standard, iff all the arcs and all the explicitly matched nodes in $g(L)$ are idle.

Once a redex has been located, a rewrite takes place. During a rewrite, a copy of $P - L$ (including the markings) is glued to the redex $g : L \to G$, yielding a graph G', in such a way that there exists an extended matching $g' : P \to G'$, and injective homomorphism $i_{G,G'} : G \to G'$. This phase is called contractum building and defines G' uniquely up to the identity of the nodes in G'. In the next phase, redirection, the G' images $g'(a)$ of LHS nodes a of *Red* pairs (a, b) are located, and all their in-arcs are redirected to the G' images $g'(b)$ of the corresponding RHS nodes b, giving graph G''. Regarded as a node map $P \to G''$, g' no longer extends to a homomorphism and we thus write it as $g'' : P \to G''$. Likewise we have the node injection map $i_{G',G''} : G' \to G''$, also no longer a homomorphism. The redirections themselves are captured by the node redirection map $r_{G',G''} : G' \to G''$ which maps each $g'(a)$ in G' to the corresponding $g'(b)$ in G'' and is otherwise identical to $i_{G',G''} : G' \to G''$. Finally in the root quiescence and activation phase, G'' images of *Act* nodes are located via g'' and made active if they were idle, and t itself is made idle unless it was one of these nodes. This gives the result graph H and injective node maps $h : P \to H$, and $i_{G'',H} : G'' \to H$.

Note that $i_{G,G'}$, $i_{G',G''}$, $i_{G'',H}$ are actually names for the appropriate identities on nodes, while g', g'', h are in fact equal as node maps. This is because of the way we chose what the nodes of the graphs G', G'', H were to be. Had we decided on a different construction, or to define these graphs only up to isomorphism (see [Section 6]), these particular properties of these maps would not have held.

[Fig. 4] shows the rewriting process in action. There is clearly a matching of the left subpattern of the rule pictured in [Fig. 2] to the graph of [Fig. 1] at the active F-labelled node. The double arrow shows the overall effect of the rewrite, while the single arrows show the effects of the three phases discussed above. For added clarity we have indicated the images of the redirection pairs under the extended matching in the second graph. In this example, the $i_{G,H}$ image of the root node $*$F of the rewrite in G is the idle F-labelled node of the result H, and the $r_{G,H}$ image of the same node is the idle S-labelled node of H. Similarly the $i_{G,H}$ image of the first S-labelled child of the root node $*$F is the idle S-labelled node of H, and the $r_{G,H}$ image of the same node is the active S-labelled node of H. For the second S-labelled child of the root node $*$F, both $i_{G,H}$ and $r_{G,H}$ take it to the active S-labelled node of H.

All executions start from a (graph consisting of a) single node labelled with the symbol Initial, and as a matter of notation, if $[X, A, B, C, \ldots, W, Y]$ is a portion of an execution sequence, we define $i_{X,Y}$ and $r_{X,Y}$ to be compositions of the elementary injection and redirection maps $i_{\_,\_}$ and $r_{\_,\_}$ discussed above over the portion $[X \ldots Y]$, where $r_{\_,\_}$ is interpreted as another name for $i_{\_,\_}$ whenever the step or substep in question is not the redirection phase of a rewrite. Thus $i_{X,Y}(x) = (i_{W,Y} \circ \ldots \circ i_{B,C} \circ i_{A,B} \circ i_{X,A})(x)$ is the copy in the graph Y of the node x in graph X, while $r_{X,Y}(x) = (r_{W,Y} \circ \ldots \circ r_{B,C} \circ r_{A,B} \circ r_{X,A})(x)$ refers to the fate of x under whatever redirections $[X \ldots Y]$ contains. In particular, the way that arc redirection works means that the arc (p_k, c) in X becomes the arc $(i_{X,Y}(p)_k, r_{X,Y}(c))$ in Y. We will use the maps $i_{X,Y}$ and $r_{X,Y}$ quite extensively below. Graphs that occur in some execution are called execution graphs.

The theory of MONSTR falls apart unless two crucial invariants hold. We define these now. We say that a node x is balanced iff [$\mu(x) = \#^n$ (for $n \geq 1$) $\Leftrightarrow$ $|\{k \mid \nu(x)[k] = \hat{}\}|$

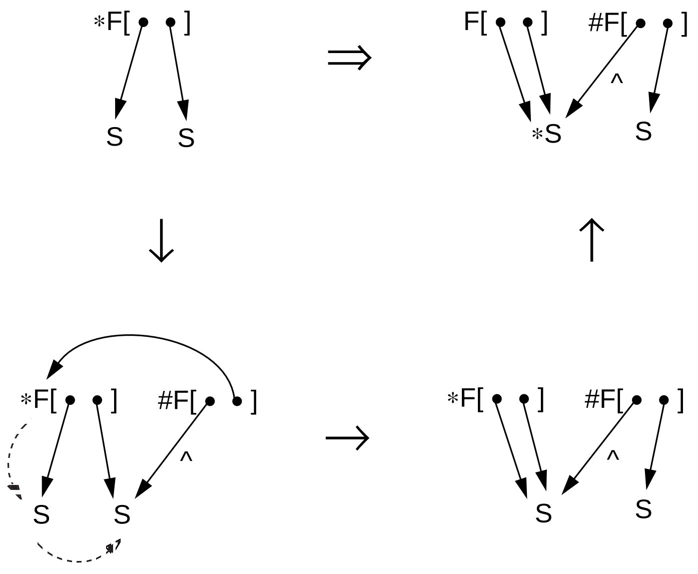

Fig. 4. A rewite execution step.

= *n*]. A graph or pattern is balanced iff the former holds for all its nodes. We say that an arc (p_k, c) is state saturated iff [$\nu(p)[k]$ = ^ and $\mu(c) = \varepsilon \;\Rightarrow\; \sigma(c) \in$ **V**]. A node is state saturated iff the former holds for all its in-arcs; and a graph or pattern is state saturated iff the former holds for all its nodes.

In order that MONSTR executions enjoy desirable properties, MONSTR symbols, rules, systems etc. must conform to a suite of restrictions. Here we tersely quote the relevant parts of **M-I** for reference.

Restriction M-I.11.1 (Alphabets) The alphabet of symbols **S**, is the disjoint union of three subalphabets **S** = **F** ∪ **C** ∪ **V** where: **F** is the alphabet of function symbols which may label the root of the left subpattern *L* of a rule, but not any subroot node of *L*, and which may be the LHS of a redirection. **C** is the alphabet of constructor symbols which may label a subroot node of the left subpattern of a rule, but not the root, and which may not be the LHS of a redirection. **V** is the alphabet of stateholders, or variables. A stateholder symbol may label a subroot node of the left subpattern of a rule, but not the root. Stateholders may label the LHS of a redirection.

Restriction M-I.11.2 (Symbols) For each $S \in$ **S**, there is a set of natural numbers $A(S)$, in every case an initial segment of the naturals from 1, or empty. For each $F \in$ **F**, there are subsets State(F) ⊆ Map(F) ⊆ $A(F)$, with State(F) either a singleton or empty. Root ∈ **C**.

Definition M-I.11.3 (Normal and Default Rules) Let $F \in$ **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.

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

(1) Each node has the arity dictated by its symbol, i.e. 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 $\alpha(root)[k]$ is explicit $\Leftrightarrow$ $k \in \mathrm{Map}(F)$.

(3) A rule for a function may match at most one stateholder, and then only in a fixed position; all other explicit arguments must be constructors, i.e. if $\sigma(root) = F$, and D is a normal rule then $\sigma(\alpha(root)[k]) \in \mathbf{V} \Rightarrow k \in \mathrm{State}(F)$.

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

(5) Implicit nodes of the left subpattern have only one parent in the left subpattern, 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. $\mu(x) = \#^n$ (for $n \geq 1$) $\Leftrightarrow$ $|\{k \mid \nu(x)[k] = \text{^}\}| = n$.

(7) Every arc (p_k, c) of P is either state saturated or activated, i.e. $\nu(p)[k] = \text{^}$ and $\mu(c) = \varepsilon \Rightarrow \sigma(c) \in \mathbf{V}$ 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. $(a, b) \in Red$ and $\mu(b) = \varepsilon \Rightarrow \sigma(b) \in \mathbf{V}$ 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. $(b, c) \in Red$ and $b \in Act \Rightarrow$ there is a $b \neq a \in L$ such that $(a, b) \in Red$.

Theorem M-I.11.5 (Desirable Properties) When all rules used, conform to Restriction **M-I**.11.4, induction over executions yields many desirable properties. Namely:

• All execution graph nodes respect the arities of their symbols. • The pattern matching requirements of each redex, depend solely on the symbol at the root. • No pointer equivalence is required for matching any redex node, that is not evident from Map($\sigma(root)$). • All execution graphs are balanced and state saturated. • When all redexes that are rewritten are standard redexes, the overwriting lemma **M-I**.5.10, applies to most redirections, in practice enabling the convenient representation of rewriting by packet store manipulations.

Restriction M-I.11.6 (Systems, Rule Selection) 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 default rules for F. In an execution, when a chosen root t is identified and it is labelled by $F \in \mathbf{F}$, rule selection is performed according to the following procedure:

```
**If** some rule from $\mathcal{N}_F$ matches the chosen root $t$
**Then** $Sel = \{D \in \mathcal{N}_F \mid D \text{ matches at } t\}$
**Else** $Sel = \mathcal{D}_F$
```

Choice of rule from *Sel* is nondeterministic.

From now on, we always assume that the systems we deal with conform to the above restrictions, i.e. are MONSTR systems.

The last point in theorem **M-I**.11.5 raises the issue of liveness and garbage which is absent from the discussion thus far, since if we are to overwrite some part of (the packet store representation of) an execution graph, we need to be sure it will never be required in the future, (we will say a little more about packet stores below). We say that a node x is live iff one or more of the following hold: (1) $\sigma(x) = \mathsf{Root}$; (2) x is active i.e. $\mu(x) = *$; (3) x is accessible from a node p via a normal arc (p_k, x) where p is already live; (4) x can access a node c via a notification arc (x_k, c) where c is already live. If a node x cannot be proved live on the basis of the preceding four conditions, we say it is garbage. And an arc is live iff both its parent and child nodes are live, being garbage otherwise. Given this, we can define the live subgraph (LSG) of a graph G as consisting of those nodes and arcs of G which are live. In **M-I** we showed that this definition is sound. Note particularly that the LSG does not necessarily satisfy all the properties required for it to be a term graph in the sense we have assumed thus far, as a live suspended node might well have garbage notification out-arcs, so that a node's presence in the LSG does not imply the presence of all its out-arcs in the LSG.

Below, we will need to refer to the following results concerning garbage.

Lemma M-I.5.10 (Overwriting lemma) Let G be a balanced graph, and let $g : L \to G$ be a standard redex for a rule $D = (P, root, Red, Act)$. Denoting the result of contractum building using a prime, as usual, let

$$Red_g = \{(g'(a), g'(b)) \mid (a, b) \in Red, g'(a) \neq g'(b)\}$$

Let $(g'(a), g'(b)) \in Red_g$ and suppose

(1) $\sigma(a) \neq \mathsf{Root}$.

(2) There is no $(g'(c), g'(d)) \in Red_g$ with $g'(d) = g'(a)$.

(3) $g'(a) \notin g'(Act)$.

Then $h(a)$ is garbage in the graph H produced by the rewrite.

Lemma M-I.5.11 (Moving lemma) Let G be a balanced graph, and let $g : L \to G$ be a standard redex for a rule $D = (P, root, Red, Act)$. Denoting the result of contractum building using a prime, as usual, let Red_g be as in lemma **M-I**.5.10 and let $(g'(a), g'(b)) \in Red_g$ satisfy

(2) There is no $(g'(c), g'(d)) \in Red_g$ with $g'(d) = g'(a)$.

Then $h(a)$ has no in-arcs in H.

3 SUSPENDING SEMANTICS

We return once more to the last point in theorem **M-I**.11.5. It contains the caveat that all redexes that are rewritten are standard redexes. This is in fact more important than it appears, since the soundness of the definition of garbage breaks down in its absence. Now from the point of view of an abstract rewriting system, there is nothing to prevent us from simply defining the rewriting relation to be restricted to standard redexes. However the MONSTR rewriting model is intended to capture a much more operationally based machine model at a suitably abstract level and so simply restricting the rewriting relation to a convenient subset of pairs of related graphs will not do. The way to achieve this restriction is instead to introduce a new atomic action.

Suspending executions consist of three kinds of atomic action: notifications, rewrites and suspensions. These are defined below. Whereas the first two are defined for arbitrary DACTL systems (i.e. systems that do not necessarily conform to the suite of restrictions we listed above), the last of these, suspensions, only make sense (in the form to be described), for MONSTR systems. The next definition states the circumstances under which each kind of action is performed.

Definition 3.1 (Execution Steps) Let G be a graph and t an active node of G, the chosen root. For suspending 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 $k \in \mathrm{Map}(\sigma(t))$, $\mu(\alpha(t)[k]) = \varepsilon$  (and $\nu(t)[k] = \varepsilon$)
      Then  Perform a rewrite using a rule chosen
            nondeterministically from $Sel$ where

            If  some rule from $N_{\sigma(t)}$ matches the chosen root $t$
            Then  $Sel = \{D \in N_{\sigma(t)} \mid D$ matches at $t\}$
            Else  $Sel = D_{\sigma(t)}$
      Else  Perform a suspension at $t$
```

Note that the part in the highlighted box is just the usual MONSTR rule selection procedure, so that the only difference between suspending semantics and DACTL semantics is when the explicit nodes that a normal rule for $\sigma(t)$ would need to match are non-idle, whereupon a suspension occurs. Note that this is subtly different from requiring that only rules having standard redexes at t are eligible for selection. Even when some $\mathrm{Map}(\sigma(t))$ arguments of a chosen root t are non-idle, a default rule for $\sigma(t)$ would still always have a standard redex, so could govern a rewrite at t. But suspending semantics indicates a suspension instead. The rationale behind this is that when a distributed packet store rewriting implementation (**M-I**.10) commences a rewrite, it begins by sending out Constructor_Request messages, and these suspend on any non-idle $\mathrm{Map}(\sigma(t))$ arguments before anything else is known about those arguments. Of course when a suspending rewrite does occur, it will turn out that it is a rewrite of a standard redex for sure.

Now we define the three kinds of suspending atomic action.

Definition 3.2 (Suspending Notification) These are defined exactly as for DACTL semantics, reviewed above. See **M-I**.3.11.

Definition 3.3 (Suspending Rewrites) These are defined exactly as for DACTL semantics, reviewed above. See **M-I**.3.10.

Definition 3.4 (Suspending Suspensions) Suppose t is a chosen root in a graph G, $\sigma(t) \in \mathbf{F}$ and there is at least one $k \in \mathrm{Map}(\sigma(t))$ such that $\alpha(t)[k]$ is non-idle. Let

Susp(t) = {$k \in$ Map($\sigma(t)$) | $\alpha(t)[k]$ is non-idle in G}
n = | 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 $\mu_G(x)$.

(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.

As above, we define the maps $i_{G,H} = r_{G,H}$ as the identity on nodes also for suspension steps, in order to be able to track the fate of nodes through executions using a uniform notation.

This completes the definition of the repertoire of actions available to suspending executions. [Fig. 5] shows a suspension step for a fragment of term graph rooted at an F-labelled chosen root.

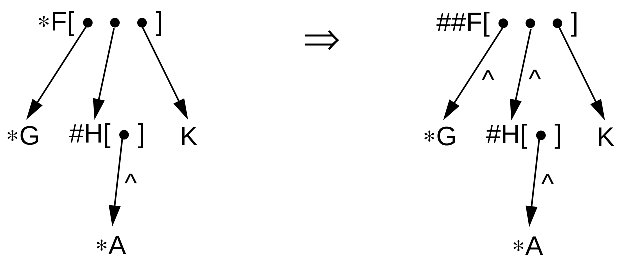

Fig. 5. A suspension step.

Now we must consider the effect of the new suspension steps on properties of executions. First, the preservation of garbage. (Note that the next theorem is actually more general than we need.)

Theorem 3.5 Let G be a balanced graph and t be an active node of t. Let Susp(t) be defined as in definition 3.4 and let $0 \neq n =$ | Susp(t) |. Suppose a suspension step is performed at t yielding graph H. 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 recall that a node is live iff, (1) it is Root-labelled, or (2) it is active, or (3) it is accessible from a live node via a normal arc, or (4) it can access a live node via a notification arc. Otherwise it is garbage.

For a suspension step from t, the suspension redex consists of all arcs (t_l, z) of G, with $l \in \mathrm{Susp}(t)$, and their constituent nodes. Since t is active and all suspension redex arcs are normal by balancedness, the suspension redex is live in G.

Consider a garbage node x in G. There is no proof of liveness of x in G so x is not in the suspension redex. Thus we consider the redex-emergent arcs. These are notification arcs (q_m, t); normal arcs (t_m, q) for $m \notin \mathrm{Susp}(t)$; normal arcs (z_m, q) where $z = \alpha(t)[l]$ for $l \in \mathrm{Susp}(t)$; and finally notification arcs (q_m, z). Before the suspension step, t and all of these arcs are live.

After the suspension, the $i_{G,H}$ image of t in H will be suspended, thus will not be live unless there is an alternative "nonlocal" proof of t's liveness in G not involving t's active marking, or one of the z's is active in G, or there is a "nonlocal" proof of such a z's liveness in G that did not involve the arc (t_l, z). (The latter two cases allow us to deduce $i_{G,H}(t)$ is live in H.) Thus in H, the $i_{G,H}$ images of the redex-emergent arcs (q_m, t) and (t_m, q) will not necessarily be live. Similarly the $i_{G,H}$ image of a z node will not be live in H unless z is active in G, or there is an independent "nonlocal" proof of the liveness of z in G not relying on the suspension redex. Thus in H, the $i_{G,H}$ images of redex-emergent arcs (z_m, q) and (q_m, z) need not be live.

Altogether, this means that for nodes and arcs outside the redex, the suspension step can destroy proofs of liveness, but not create new ones. We conclude that the $i_{G,H}$ image of 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 suspension 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 suspensions, $r_{G,H}(c)$ is garbage in H, giving the conclusion. We are done. ☺

Now we come to the first high point in the theory of the suspending model.

Theorem 3.6 (Properties of Suspending Semantics) Let $\mathcal{R}$ be a MONSTR system and let $\mathsf{G} = [G_0, G_1\ldots]$ be an execution of the system according to suspending semantics. Then

(1) Every rewrite in the execution is a rewrite of a standard redex.

(2) Every execution step preserves garbage.

(3) Every graph G_i is balanced and state saturated.

Proof. By induction on executions. Clearly (3) holds for the initial graph, and (1) and (2) hold for the initial rewrite. Suppose then (3) holds for all execution graphs up to and including G_{i-1}, and (1) and (2) hold for all execution steps that led to G_{i-1}. By (3) we know that G_{i-1} is balanced, hence if G_i is obtained from G_{i-1} by a rewrite, since all explicit non-root arguments of the redex are idle, the redex is standard, giving (1).

We know that rewrites of standard redexes preserve garbage (**M-I**.5.7), that notifications preserve garbage (**M-I**.5.8), and that suspensions preserve garbage (by theorem 3.5), so we have (2). Finally we know that rewrites and notifications preserve balancedness and state saturatedness (**M-I**.4.2 and **M-I**.4.4); so to get the result we need to check suspensions. But it is easy to see that suspensions preserved balancedness; and since all the new notification arcs created in a suspension have non-idle child nodes, state saturatedness is preserved also. So we have (3) for G_i. We are done. ☺

Thus changing DACTL semantics by the introduction of suspension steps, forces all executions to behave in the way that we would wish. The suspensions in fact enforce the firewall principle (**M-I**.7) in the suspending model, and this is the source of the pleasing properties we have derived.

As the introduction of suspension steps is intended to work smoothly with the lower level aspects of the MONSTR model, we should check that suspensions translate well into the world of packet store primitives described in **M-I**.6, and we should prove an appropriate adequacy theorem. We will not do this completely formally, mainly because suspension steps are so simple; we content ourselves with a brief overview of packet stores, and then we will examine a typical example.

A packet store representation of a MONSTR graph G is contained within a set of packet store locations. Each distinct node x of G is represented by a distinct packet at a location ~~*x*~~. The location contains: (1) x's symbol $\sigma(x)$; (2) x's marking $\mu(x)$; (3) a sequence of items each of which is either the unit BLANK, or another location ~~*y*~~ (a forward pointer); (4) a return address set $\rho(x)$ containing a set of pairs such as ~~*m*~~.k where ~~*m*~~ is a packet location and k is an index into the sequence of items at ~~*m*~~ (a reversed pointer). The invariant that binds the packet store to the represented graph G asserts that: (a) normal arcs (p_k, c) of G are in bijective correspondence with entries ~~*c*~~ in k'th position of the sequence of items at ~~*p*~~, where locations ~~*p*~~ and ~~*c*~~ hold the representatives of nodes p and c respectively; (b) notification arcs (p_k, c) of G are in bijective correspondence with entries ~~*p*~~.k in the return address set of ~~*c*~~, where locations ~~*p*~~ and ~~*c*~~ hold the representatives of nodes p and c respectively and the k'th position of the sequence of items at ~~*p*~~ is BLANK.

In such an environment, the representation of suspension steps is simple enough. All we need do, is to change the active marking in the root packet to a suitable suspension, remove the root packet's forward pointers to Map(σ(*root*)) non-idle child packets, replacing them by BLANK, and insert a reversed pointer to the relevant root packet item, into the return address set of each such child packet.

This simple picture becomes slightly more complicated when we optimise the representation to allow garbage not to be represented, and permit the presence of Ind(irection) packets as intermediaries in the representation of arcs to allow the local implementation of redirection (see **M-I**.6 for a detailed discussion). Nevertheless the complications are rather elementary and we merely give an example.

[Fig. 6] shows the packet store transformation for the suspension of [Fig. 5]. [Fig. 6.(a)] shows the pre-state while [Fig. 6.(b)] shows the post-state. Note that the first child of the root packet is assumed to have been exported to some other location in the packet store, leaving behind a suspended Ind which causes the suspension on that argument of the root (as the argument itself is inaccessible), while the second child argument *is* itself accessible because it can be found by traversing an idle indirection chain (all the point-

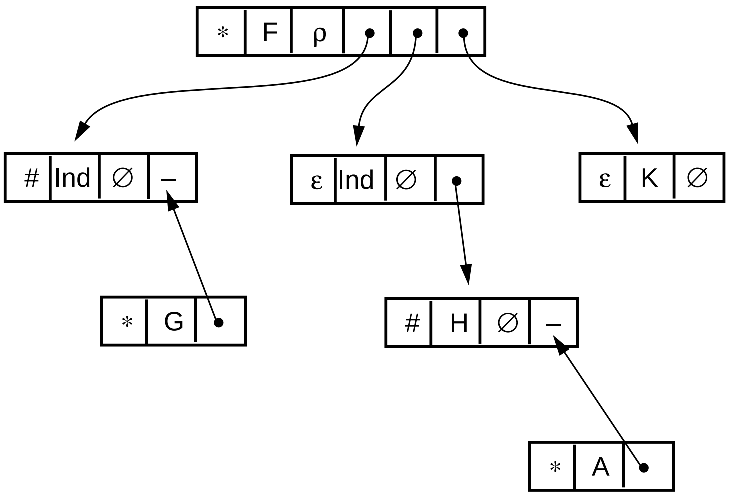

Fig. 6.(a) Pre-state of the packet store for the suspension of [Fig. 5].

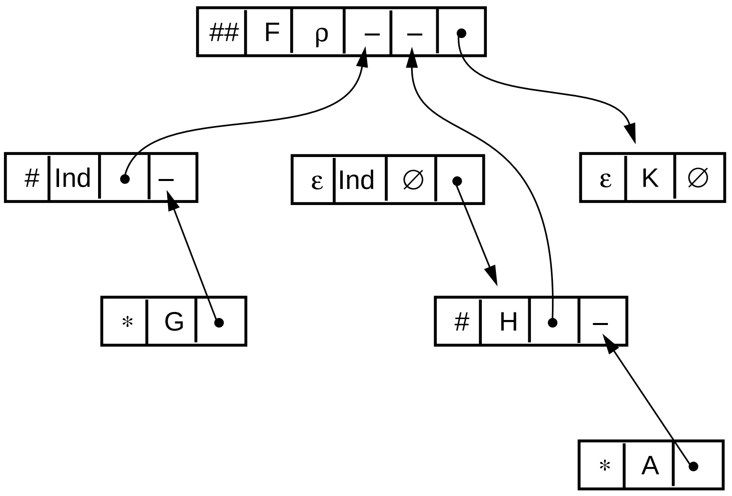

Fig. 6.(b) Post-state of the packet store for the suspension of [Fig. 5].

ers that represent the arc in question point in the necessary direction), allowing the destination packet to be retrieved, at which point the suspension is caused because the argument itself is non-idle.

4 CORRECTNESS OF SUSPENDING EXECUTIONS

In this section we study the effect on correctness (with respect to DACTL semantics) that the introduction of suspension steps generates. The first important result is below.

Theorem 4.1 Let R be a MONSTR system. Let $\mathsf{G} = [G_0, G_1 \ldots]$ be an execution of R according to suspending semantics. Let the domain of G (i.e. its set of indices) be $\mathbf{N}_G$. Then there is an execution of R, $\mathsf{G}^{\mathrm{D}} = [G^{\mathrm{D}}{}_0, G^{\mathrm{D}}{}_1 \ldots]$ according to DACTL semantics, with domain $\mathbf{N}_{G^{\mathrm{D}}}$, such that there is a non-decreasing map $\delta : \mathbf{N}_G \to \mathbf{N}_{G^{\mathrm{D}}}$ such that for all $i \in \mathbf{N}_G$,

(1) There is a graph structure isomorphism $\theta_i : G_i \to G^{\mathrm{D}}{}_{\delta(i)}$.

(2) $x \in G_i$ is non-idle $\Leftrightarrow$ $\theta_i(x) \in G^{\mathrm{D}}{}_{\delta(i)}$ is non-idle.

(3) $x \in G_i$ and $\mu(x) = \#^n \Rightarrow \mu(\theta_i(x)) = \#^m$ with $0 \le m \le n$ (and $\#^0 = *$), so that

(4) $x \in G_i$ is active $\Rightarrow$ $\theta_i(x) \in G^{\mathrm{D}}{}_{\delta(i)}$ is active.

(5) $(p_k, c) \in G_i$ is a normal arc $\Rightarrow$ $(\theta_i(p)_k, \theta_i(c)) \in G^{\mathrm{D}}{}_{\delta(i)}$ is a normal arc.

(6) (p_k, c) is a notification arc in G_i and $(\theta_i(p)_k, \theta_i(c))$ is a normal arc in $G^{\mathrm{D}}{}_{\delta(i)}$ $\Rightarrow$ $\theta_i(p)$ is active.

And

(7) Whenever execution step $G_i \to G_{i+1}$ is a rewrite with root t governed by rule D, then execution step $G^{\mathrm{D}}{}_{\delta(i)} \to G^{\mathrm{D}}{}_{\delta(i)+1}$ is a rewrite of a standard redex with root $\theta_i(t)$ governed by rule D; and whenever execution step $G_i \to G_{i+1}$ is a notification from t, then execution step $G^{\mathrm{D}}{}_{\delta(i)} \to G^{\mathrm{D}}{}_{\delta(i)+1}$ is a notification from $\theta_i(t)$.

Proof. We construct G^{D} and $\delta : \mathbf{N}_G \to \mathbf{N}_{G^{\mathrm{D}}}$ as follows.

Step 0: Let $\delta(0) = 0$ and $G^{\mathrm{D}}{}_0 = G_0$. Clearly (1) – (6) hold for $i = 0$.

Step $i + 1$: Suppose we have δ for $j \in [0 \ldots i]$ and the corresponding $G^{\mathrm{D}}{}_{\delta(j)}$. There are three cases depending on whether G_{i+1} is obtained from G_i by notification, by rewriting, or by a suspension.

Notification case. Let $t \in G_i$ be the chosen root for the notification so $\sigma(t) \in \mathbf{C} \cup \mathbf{V}$. By hypothesis $\theta_i(t)$ is active, and $\sigma(\theta_i(t)) \in \mathbf{C} \cup \mathbf{V}$ so that node $\theta_i(t)$ is a possible root for a notification in $G^{\mathrm{D}}{}_{\delta(j)}$. Let $G^{\mathrm{D}}{}_{\delta(j)+1}$ be the result of performing a notification at $\theta_i(t)$ in $G^{\mathrm{D}}{}_{\delta(j)}$, and let $\delta(i + 1) = \delta(i) + 1$. Since there is a graph structure isomorphism $\theta_i : G_i \to G^{\mathrm{D}}{}_{\delta(i)}$ (note that this does not require the node and arc markings to be faithfully mapped by θ_i; see **M-I**.3.3), and notifications merely manipulate the markings, there is obviously a graph structure isomorphism $\theta_{i+1} : G_{i+1} \to G^{\mathrm{D}}{}_{\delta(i+1)}$ giving (1).

We note that notifications only manipulate the markings in the notification redex. In G_i (resp. $G^{\mathrm{D}}{}_{\delta(i)}$), t (resp. $\theta_i(t)$) is active and has its active marking replaced by the idle marking in G_{i+1} (resp. $G^{\mathrm{D}}{}_{\delta(i+1)}$). Noting also that non-idle parents of t (resp. $\theta_i(t)$) merely have one non-idle marking changed to another when their out-arcs to t (resp. $\theta_i(t)$) are notification arcs, we conclude that (2) is preserved by notifications. Since (3) holds for the notification redexes in G_i and $G^{\mathrm{D}}{}_{\delta(i)}$ and notifications decrement non-zero suspension marks, (3) holds for the graphs G_{i+1} and $G^{\mathrm{D}}{}_{\delta(i+1)}$, whence we quickly get (4) for G_{i+1} and $G^{\mathrm{D}}{}_{\delta(i+1)}$. It is easy to see that (5) is preserved as notifications normalise all notification in-arcs of t. Likewise for (6).

Rewrite case. By hypothesis G_i and $G^{\mathrm{D}}{}_{\delta(i)}$ are graph structure isomorphic, whence if t is the chosen root in G_i, $\sigma(t) = \sigma(\theta_i(t)) \in \mathbf{F}$. By (2) and (4) and the fact that G_i has a standard redex for some rule $D = (P, root, Red, Act)$ for $\sigma(t)$ at t, $G^{\mathrm{D}}{}_{\delta(i)}$ has a standard redex for the same rule at $\theta_i(t)$. And because D was eligible for selection to govern the step $G_i \to G_{i+1}$, so is D to govern a step $G^{\mathrm{D}}{}_{\delta(i)} \to G^{\mathrm{D}}{}_{\delta(i+1)}$. Therefore we perform a rewrite at $\theta_i(t)$ similar to the one at t.

Let us compare the rewriting processes that create G_{i+1} from G_i and $G^{\mathrm{D}}{}_{\delta(i+1)}$ from $G^{\mathrm{D}}{}_{\delta(i)}$ using the rule D. Let $g_i : L \to G_i$ and $g^{\mathrm{D}}{}_{\delta(i)} : L \to G^{\mathrm{D}}{}_{\delta(i)}$ be the redexes.

The two contractum building phases clearly allow the extension of $\theta_i : G_i \to G^{\mathrm{D}}{}_{\delta(i)}$ to a graph structure isomorphism

$$\theta_i' : G_i' \to G^{\mathrm{D}}{}_{\delta(i)}'$$

such that the obvious triangle involving $g_i' : P \to G_i'$ and $g^{\mathrm{D}}{}_{\delta(i)}' : P \to G^{\mathrm{D}}{}_{\delta(i)}'$ commutes. Since identically labelled and marked nodes and edges are added to corresponding places, it is easy to see that (1) – (6) hold for G_i' and $G^{\mathrm{D}}{}_{\delta(i)}'$ via θ_i'.

Also the redirection phase admits a further extension to a graph structure isomorphism

$$\theta_i'' : G_i'' \to G^{\mathrm{D}}{}_{\delta(i)}''$$

such that the triangle involving the node maps $g_i'' : P \to G_i''$ and $g^{\mathrm{D}}{}_{\delta(i)}'' : P \to G^{\mathrm{D}}{}_{\delta(i)}''$ commutes too. Since similar modifications are made to both graphs, it is again easy to see that (1) – (6) hold for G_i'' and $G^{\mathrm{D}}{}_{\delta(i)}''$ via θ_i''.

Likewise the activation phase finally yields the graph structure isomorphism

$$\theta_{i+1} : G_{i+1} \to G^{\mathrm{D}}{}_{\delta(i+1)}$$

such that the triangle involving $h_{i+1} : P \to G_{i+1}$ and $h^{\mathrm{D}}{}_{\delta(i+1)} : P \to G^{\mathrm{D}}{}_{\delta(i+1)}$ commutes, and we easily see that (1) – (6) hold for G_{i+1} and $G^{\mathrm{D}}{}_{\delta(i+1)}$ via θ_{i+1}.

Suspension case. Here we do not construct a new graph at all. We set $\delta(i+1) = \delta(i)$, $G^{\mathrm{D}}{}_{\delta(i+1)} = G^{\mathrm{D}}{}_{\delta(i)}$, and $\theta_{i+1} = \theta_i$, noting in the latter case that markings are ignored by the θ maps; giving (1). As for notifications, we observe that if $t \in G_i$ is the chosen root, the marking on t changes from $*$ to $\#^n$, both of which are non-idle markings; otherwise all node markings remain the same. This quickly yields (2) – (4); (5) is also easy to see. The basic nature of suspensions ensures that (6) is preserved.

The above yields a sequence of execution steps for which property (7) is obvious by construction.

If G is non-terminating, it is a DACTL execution G^{D}. (We do not contemplate transfinite executions in this paper). Otherwise it is a prefix of a DACTL execution, possibly a proper one since there may be active nodes left over in the final G^{D} graph arising from suspensions in G not all of whose arguments subsequently notified. In such a case we can complete the DACTL execution by performing rewrites and notifications according to DACTL semantics until we either reach a final graph, or the execution does not terminate. The result in either case is G^{D}. ☺

The close relationship between G and G^{D} constructed above, extends to liveness and garbage as follows.

Theorem 4.2 Assume the notation of theorem 4.1. Then for all $i \in \mathbf{N}_G$,

(1) $x \in G_i$ is live $\Rightarrow$ $\theta_i(x) \in G^{\mathrm{D}}_{\delta(i)}$ is live.

(2) $(p_k, c) \in G_i$ is a live arc $\Rightarrow$ $(\theta_i(p)_k, \theta_i(c)) \in G^{\mathrm{D}}_{\delta(i)}$ is a live arc.

Proof. We start with the nodes. We use a form of induction on the size of a proof of the liveness of $x \in G_i$. (By the size of a proof presented in the sequent style of **M-I**.Fig.10, we mean the sum of the number of occurrences of formulae and of the number of occurrences of inference steps in the proof, in an obvious way.) The precise induction hypothesis is

If $x \in G_i$ has a liveness proof $\Pi_{x,i}$ of size n, then
$\theta_i(x) \in G^{\mathrm{D}}_{\delta(i)}$ has a liveness proof $\Pi^{\mathrm{D}}_{\theta_i(x),\delta(i)}$ of size $m \le n$.

If x is live because it is Root-labelled or active, then by theorem 4.1.(1) or 4.1.(4) respectively, $\theta_i(x)$ is live in $G^{\mathrm{D}}_{\delta(i)}$.

If x is live because there is a normal arc (p_k, x) with p live in G_i, then p has a smaller liveness proof than x, and by theorem 4.1.(5), $(\theta_i(p)_k, \theta_i(x))$ is a normal arc. By hypothesis $\theta_i(p)$ is live, whence $\theta_i(x)$ is live, both with proofs of requisite size.

If x is live because there is a notification arc (x_k, c) with c live in G_i, then c has a smaller liveness proof than x, and $(\theta_i(x)_k, \theta_i(c))$ is either a notification arc or a normal arc. If the former, then $\theta_i(c)$ is live by hypothesis, whence $\theta_i(x)$ is live, both with proofs of requisite size. If the latter, then by theorem 4.1.(6), $\theta_i(x)$ is active anyway, whence live, by a very small proof.

Since an arc is live iff both the parent and child nodes are live, we immediately deduce the result for arcs. ☺

Thus suspending executions map fairly straightforwardly into at least prefixes of DACTL executions of the same system. If things go well, the suspensions of the suspending execution subsequently receive enough notifications to reactivate their roots and let a rewrite take place. When all suspensions are thus released, we get the same final graph (assuming the execution terminates). We now study this further.

Definition 4.3 Let G_i be an execution graph (for either suspending or DACTL semantics). The execution dependency graph (EDG) of G_i consists of

(1) all notification arcs and their parent and child nodes,

(2) all non-idle nodes.

Note that like the live subgraph (LSG), the EDG need not satisfy all the invariants for being a term graph, not least because not all out-arcs of a suspended node need be notification arcs.

Theorem 4.4 Let G_f be an execution graph of a suspending execution G of a MONSTR system R. Then

(1) (a) All nodes of G_f are idle $\Rightarrow$
(b) The EDG of G_f is empty.

(2) (a) The EDG of G_f is empty $\Rightarrow$ [
(b) G_f is the final graph of the suspending MONSTR execution G of R, and
(c) G_f is the final graph of the associated DACTL execution G^{D} of R.]

Proof. (1) is clearly true by balancedness. For (2), (a) $\Rightarrow$ (b) since all nodes of G_f are idle, whence there is no candidate chosen root for another execution step. Also (a) $\Rightarrow$ (c) by theorem 4.1.(2). ☺

Corollary 4.5 If G is a suspending execution of a MONSTR system R, with final graph G_f, and the EDG of G_f is empty, then there is a DACTL execution of R producing the same final graph.

N.B. In the following theorem, the two chains of implications are to be understood as shorthand for {[(a) $\Rightarrow$ (b)] $\wedge$ [(b) $\Rightarrow$ (c)]}.

Theorem 4.6 Let G_f be a suspending semantics execution graph of a MONSTR system R. Then

(1) (a) G_f contains no active nodes $\Rightarrow$
(b) The EDG of G_f contains no active nodes $\Rightarrow$
(c) The EDG of G_f is non-empty $\Leftrightarrow$ the EDG of G_f contains a suspended node, and [there is a cycle of deadlocked suspended nodes in G_f, or there is an idle stateholder at the head of a notification arc of G_f, (or both)].

(2) (a) The EDG of G_f contains no active nodes $\Rightarrow$
(b) G_f is the final graph of the suspending execution G of R $\Rightarrow$
(c) G_f is graph-structure isomorphic to a graph of the associated DACTL execution G^{D} of R.

Proof. Consider a node in the EDG of an execution graph G_f of R. If it is active, or an idle stateholder at the head of a notification arc, it is a leaf of the EDG. If it is suspended, it has at least one child node. These remarks are true by balancedness and state saturatedness. Moreover there are no other possibilities.

Consider a maximal path of the EDG containing at least one arc. Perforce it must start at a suspended node, and must encounter more suspended nodes as long as it does not find a leaf of the EDG, again by balancedness.

Now for (1), (a) $\Rightarrow$ (b) is obvious. Also (b) $\Rightarrow$ (c) because if the EDG of G_f is non-empty and it has no active nodes, then any leaf of the EDG must be an idle stateholder, implying the presence of a suspended parent. If there is no leaf in the EDG, then since

G_f is finite, its EDG is finite, and so must consist of suspended nodes knotted together in cycles. Part (2) is obvious. ☺

Corollary 4.7 Let G_i be a suspending semantics execution graph of a MONSTR system R. If the EDG of G_i contains either of the following:

(a) A cycle consisting of suspended nodes and notification arcs. Or:

(b) A garbage idle stateholder at the head of a notification arc.

Then no subsequent execution graph will have an empty EDG.

Proof. A suspended node which is either in a cycle of suspended nodes and notification arcs, or is the ancestor (via a path of suspended nodes and notification arcs) of a garbaged idle stateholder, will remain permanently suspended by balancedness. In the former case there is a deadlocked cycle of suspensions, in the latter case, a needed notification from the idle stateholder will never materialise. So the EDG will remain non-empty in each subsequent execution step. ☺

Assuming we define correctness of a terminating suspending execution of a MONSTR system R, by the property that its final graph is marking-preserving isomorphic to the final graph produced by a DACTL execution, then corollary 4.7 identifies some bad things, which once they have occurred in an execution graph, preclude correctness subsequently. The negations of these bad things lead to some good things which we can use to assert correctness. The next result is perhaps the most important one in this paper.

Theorem 4.8 Let R be a MONSTR system, and let $\mathsf{G} = [G_0, G_1 \ldots]$ be a suspending execution of R. Suppose the following hold.

(1) For all $i \in \mathbf{N}_G$ the EDG of G_i is acyclic.

(2) For all $i \in \mathbf{N}_G$ if $x \in G_i$ is an idle stateholder child of a notification arc, then there is a $i < j \in \mathbf{N}_G$.

(3) G terminates.

Then G is correct, i.e. its final graph G_f can be obtained from a DACTL execution of R.

Proof. Consider the EDG of G_f. It is acyclic by (1), (thus in particular does not contain any deadlocked cycles of suspensions). It contains no active nodes since G_f is final by (3), and it contains no idle stateholder node which is a child node of a notification arc since if it did, there would have to be an $f < j \in \mathbf{N}_G$ and G_f would not be final. Thus the EDG of G_f is a finite directed acyclic graph without leaf nodes, hence empty. Thus G is correct by theorem 4.4. ☺

Theorem 4.8 reduces correctness to three sub-problems. The first is to establish the acyclicity of the EDG of any execution graph (the initial graph obviously has an acyclic EDG). The second is to show that idle stateholder children of notification arcs ultimately always have the opportunity to participate in a suitable rewrite. The third is the problem of termination. All of these subproblems give reasonable prospects for static analysis. Properties of rule systems R can be formulated, that guarantee the relevant properties of execution graphs. However, these topics are beyond the scope of this paper and will be treated in depth elsewhere.

We continue this section by considering what happens when we take the preceding notions modulo garbage, i.e. we ask the question whether we can establish the correctness of a suspending execution with respect to a DACTL execution of the same system such that the two final graphs differ only in garbage. Unfortunately this precludes use of our most powerful tool, the emptiness of the EDG. Consequently the results below are fairly weak. They are quoted without proof, being easy adaptations of preceding material.

Definition 4.9 Let G_i be an execution graph (for either suspending or DACTL semantics). The live execution dependency graph (LEDG) of G_i consists of

(1) all live notification arcs and their parent and child nodes,

(2) all live non-idle nodes.

As in theorem 4.6, the chains of implications and equivalences in the next two theorems, are to be read as conjunctions of binary implications or equivalences.

Theorem 4.10 Let G_f be a suspending execution graph of a MONSTR system R. Then

(1) All live nodes of G_f are idle $\Leftrightarrow$

(2) G_f is the final graph of the suspending execution G of R $\Leftrightarrow$

(3) The LEDG of G_f is empty $\Rightarrow$

(4) The EDG of G_f is non-empty $\Leftrightarrow$ there is a (garbage) deadlocked cycle of suspended nodes, or a notification arc whose child node is an idle stateholder in the EDG of G_f.

Theorem 4.11 Let G_f be a suspending execution graph of a MONSTR system R. Then

(1) G_f contains no active nodes $\Leftrightarrow$

(2) G_f is the final graph of the suspending execution G of R $\Leftrightarrow$

(3) The LEDG of G_f contains no active nodes $\Rightarrow$

(4) The LEDG is non-empty $\Leftrightarrow$ there is a deadlocked cycle of suspended nodes, or a notification arc whose child node is an idle stateholder in the LEDG of G_f.

The previous two results are very similar. Theorem 4.10 deals with emptiness of the LEDG, which implies that nodes in the EDG must be non-active garbage. Theorem 4.11 deals with non-activeness of the LEDG, which permits live deadlocked cycles and suspensions on stateholders, unlike theorem 4.10. Note that neither result is able to say much about any associated DACTL execution, since the DACTL execution may well manifest live parts of a graph which correspond to garbage in their suspending MONSTR counterparts. Only if all such live parts commit garbage-theoretic suicide later in the DACTL execution, may we anticipate a equivalence of executions up to garbage. Clearly theorems of a general nature such as those above, will not be able to assert whether this happens or not.

Finally for this section, we examine an example of a terminating system.

Example 4.12 We revisit the factorial example of **M-I**.

S consists of

  F = {Fac, Mul, Sub, Initial}
  C = {0, 1, 2, 3, …} i.e. the naturals
  V = $\emptyset$

Rules:

```
Sub[ 0 0 ] => *0  |
Sub[ 1 0 ] => *1  |
… etc. … i.e. the normal delta rules for subtraction.  ;

Sub[ x y ] => ##Sub[ ^*x ^*y ]  ;

Mul[ 0 0 ] => *0  |
Mul[ 1 0 ] => *0  |
… etc.  …  i.e. the normal delta rules for multiplication.  ;

Mul[ x y ] => ##Mul[ ^*x ^*y ]  ;

Fac[ 0 ] => *1  ;

Fac[ n ] => #Mul[ n ^#Fac[ ^*Sub[ n 1 ] ] ]  ;

Initial => *Fac[ 5 ]  ;
```

This is a fairly uncomplicated example. It is easy to show by a straightforward induction over suspending executions that any instance of an active function symbol in any execution graph of the system has all of its matched arguments already idle. So the only execution (there is only one as the system is completely deterministic) is a DACTL execution and is automatically correct. Note how the default rules for Sub and Mul are effectively just programmed up versions of suspensions, and the fact that all matched arguments turn out to be in constructor form is reflected in the fact that these rules are never actually used.

However considering suspending semantics allows us to increase the concurrency in the system, rewriting three of the default rules as follows.

```
Sub[ x y ] => *Sub[ *x *y ]

Mul[ x y ] => *Mul[ *x *y ]

Fac[ n ] => *Mul[ n *Fac[ *Sub[ n 1 ] ] ]
```

In this version, the three functions instantiated on the RHS of the Fac default rule can all attempt to rewrite as soon as an instance of that rule has completed its rewrite. Unfortunately all these functions are strict and need their arguments in constructor form before they can progress the computation. So the only thing that happens if any of them attempt to rewrite at any moment other than the point at which they become scheduled to rewrite in the DACTL execution, is that they become suspended, only to be notified later. Increasing the concurrency thus wastes work here. This is not untypical. If one can deduce the dependencies in a computation well enough statically to program in the requisite suspensions at contractum building time, it is always preferable to do so. However this is not always possible, and then the suspension mechanism guarantees that the computation does things in the right order nevertheless. However this is not the only or even the main virtue of suspending semantics. The most important aspect of suspensions is that they prevent, in an implementable manner, the collapse of the mutually reinforcing system of properties characterised by balancedness, state saturatedness, and the soundness of garbage.

5 CORRECTNESS FOR NON-TERMINATING EXECUTIONS

Since theorem 4.1 worked equally well for terminating and for non-terminating executions, it is desirable to extend the correctness results generated from it to non-terminating executions. Unfortunately, the main criterion available to us, the emptiness of the EDG, is not applicable when the execution doesn't terminate; by definition. We must therefore be content with some sort of approximation to emptiness of the EDG, and different approximations will yield different notions of correctness. Pursuing such different approximations can rapidly transform the study of correctness into a branch of point set topology, a diversion which we do not wish to take in this paper, so we content ourselves with the simplest feasable approximation.

Definition 5.1 Let $\mathcal{G} = [G_0, G_1\ldots]$ be a (non-terminating) suspending execution of a MONSTR system $\mathcal{R}$. Suppose for all $i \in \mathbf{N}_{\mathcal{G}}$, for all $x \in G_i$, there is an $\mathrm{N}_{i,x}$ such that for all $j \in \mathbf{N}_{\mathcal{G}}$ with $j \geq \mathrm{N}_{i,x}$, $i_{G_i,G_j}(x)$ is not in the EDG of G_j. Then we say that $\mathcal{G}$ is pointwise correct with respect to DACTL semantics.

Note that this definition provides a very weak notion of correctness: that of pointwise approach to non-membership of the EDG by all nodes. Further, the fact that all nodes eventually vacate the EDG, may not be useful; eg. the steadily increasing set of henceforth idle nodes that the definition promises, may in fact consist predominantly of garbage.

Usually, showing that an execution satisfies a property like definition 5.1, will involve some assumptions of fairness. This can be a complex subject [Francez (1986)]. Normally we will just make whatever assumptions are necessary to make progress. For definition 5.1 specifically, it is generally enough to assume that every active node in an execution graph, becomes a chosen root eventually.

The next definition gives us the ammunition to track the life history of a node through an execution. A node may engage in a limited number of events depending on which atomic action is next in the execution, and what part if any the node has to play in that action. Thus a node may play no part at all, being an innocent bystander in the atomic action; it may merely have its marking changed as the result of notification, activation or suspension; or it may undergo redirection. There are no other possibilities. More formally we have this definition.

Definition 5.2 Let $\mathcal{G} = [G_0, G_1\ldots]$ be a suspending execution of a MONSTR system $\mathcal{R}$. A node event chain E_x for a node x in a graph G_i of $\mathcal{G}$, is a maximal sequence of the form

$$E_x \equiv [\ (\mu_x\ x : \sigma_x)_k\ ,\ (\mu_y\ y : \sigma_y)_{k+1}\ ,\ (\mu_z\ z : \sigma_z)_{k+2}\ ,\ \ldots\ ]$$

such that the following are true.

(1) If $(\mu_z\ z : \sigma_z)_m$ is an element of E_x then $z \in G_m$, and μ_z and σ_z are the marking and symbol of z in G_m respectively. (I.e. $\mu(z) = \mu_z$ and $\sigma(z) = \sigma_z$.)

(2) If the first element in the sequence is $(\mu_x\ x : \sigma_x)_k$, then either $k = 0$ and x is the initial node in G_0, or x is not in the image of i_{G_{k-1},G_k}. (I.e. the atomic action that creates G_k is a rewrite, and x is introduced during the contractum building phase.)

(3) If $(\mu_y\ y : \sigma_y)_m$, $(\mu_z\ z : \sigma_z)_{m+1}$ are an adjacent pair of elements in E_x, then

EITHER $z = i_{G_m,G_{m+1}}(y) = r_{G_m,G_{m+1}}(y)$, $\sigma(z) = \sigma(y)$, and one of the following holds:

(a) The pair describe an identity event for y, and $\mu_y = \mu_z$.

(b) The pair describe a notification event for y, and $\mu_y = *$ and $\mu_z = \varepsilon$.

(c, d) The pair describe an activation event for y, and $\mu_y \in \{\varepsilon, \#^n\ (n \geq 1)\}$ and $\mu_z = *$.

(e) The pair describe a decrement suspension event for y, and $\mu_y = \#^n$ $(n \geq 2)$ and $\mu_z = \#^{n-m}$ with $n - m \geq 1$.

(f) The pair describe a suspension event for y, and $\mu_y = *$ and $\mu_z = \#^n$ $(n \geq 1)$.

OR $r_{G_m,G_{m+1}}(y) \neq i_{G_m,G_{m+1}}(y)$ and one of the following holds:

(g, h) The pair describe an activation event for y, and $z = i_{G_m,G_{m+1}}(y)$, and $\mu_y \in \{\varepsilon, *\}$ and $\mu_z = *$.

(i) The pair describe a redirection event for y, and $z = r_{G_m,G_{m+1}}(y)$.

It is clear from definition 5.2.(2) and the fact that G is an execution, that identity events apply to all nodes not affected by an execution step (a); that notification events apply only to roots of notifications (b); that activation events apply only to nodes activated but not redirected by a rewrite (c), or activated by a notification (d), or activated and redirected by a rewrite in non-root position (g), or activated and redirected by a rewrite in root position (h); that decrement suspension events apply only to non-root members of a notification redex (e); that suspension events apply only to roots of suspensions (f); and that redirection events apply only to nodes that are being redirected during a rewrite (i). Thus a node event chain starting at some node records what happens to that node from a computational point of view. It is clear that by examining an adjacent pair of elements in a node event chain, we can tell what sort of event they depict.

We observe that node event chains can share common suffixes when redirections redirect one node to another, or several nodes to the same destination. They can share common prefixes too, but only when there is a node that is both activated and redirected during some rewrite, participating in event types (g) and (i), or (h) and (i) simultaneously. We also point out that they share some of the characteristics of sequences of snapshots of a single location in the packet store representations of **M-I**.6, albeit at a higher level of abstraction — not requiring indirections, and ignoring arcs. (Evidently, two node event chains share a common prefix exactly when the conditions of the overwriting lemma **M-I**.5.10 do not hold at some location in the packet store, and one has to be content with the weaker provisions of the moving lemma **M-I**.5.11.) [Fig. 7] gives a state transition diagram for nodes classified by marking and symbol, and for the kinds of permitted events according to suspending semantics. Thick lines represent redirections, where perforce the LHS and RHS nodes are different, while thin lines, suitably labelled, represent the other kinds of events, where only one node (and its image under $i_{G_m,G_{m+1}}$) is involved. Identity events are not shown, for clarity. Note that there is a pleasing (though not exact) degree of symmetry about the diagram.

Lemma 5.3 Let G be a suspending execution of a MONSTR system R. Let x be a live node of graph G_k of G. Then $(\mu(x)\ x : \sigma(x))_k$ is an element of some node event chain E_w of G.

Proof. By induction over executions. In G_0, the one and only initial node $\mathsf{initial}$ is in the common first element $(*\ \mathsf{initial} : \mathsf{Initial})_0$ in all node event chains that have a 0-subscripted element. The result follows for subscript 0.

Suppose the result holds up to G_m. If the next execution step $G_m \rightarrow G_{m+1}$ is not a rewrite then it is easy to see that the $i_{Gm+1,Gm}$ image of x occurs in element $(\ldots)_{m+1}$ in E_w iff x occurs in the preceding element $(\ldots)_m$ in E_w. So if all the nodes of Live(G_m) occur in m-subscripted elements of node event chains, then all their $i_{Gm+1,Gm}$ images occur in m+1-subscripted elements of node event chains. Consequently we get the required result as Live(G_{m+1}) $\subseteq i_{Gm,Gm+1}$(Live(G_m)).

Otherwise the next execution step $G_m \rightarrow G_{m+1}$ is a rewrite. All contractum nodes introduced during the rewrite are in the first (m+1-subscripted) elements of all corresponding node event chains by definition 5.2.(2), so the conclusion holds for them regardless of whether they are live in G_{m+1} or not. So now consider Live(G_m). By theorem 3.6.(2), we know that Live($i_{Gm,Gm+1}(G_m)$) $\subseteq i_{Gm,Gm+1}$(Live(G_m)), so the conclusion holds for all live nodes of G_m subjected to an event of type other than (i) in definition 5.2. For a node $x \in G_m$ subjected to an event of type (i), i.e. x is the LHS of a redirection, we know that x is balanced and $\sigma(x) \neq \mathsf{Root} \in \mathbf{C}$. Thus by the overwriting lemma **M-I**.5.10, the $i_{Gm,Gm+1}$ image of x will not be live unless, either x is also activated, or x is also the RHS of some other redirection (or both). But in these cases there is an m+1-subscripted element $(\ldots\ i_{Gm,Gm+1}(x)\ \ldots)_{m+1}$ in some node event chain that records these possibilities. ☺

Lemma 5.4 Let G be a suspending execution of a MONSTR system R. Let $y = i_{G^*,G_\mathrm{A}}(y^*)$ in graph G_A, be the i_{G^*,G_A} image of a node y^* first introduced in graph G^* of G, and let A be the smallest subscript k such that for each element $(\mu_z\ z : \sigma_z)_k$ in every node event chain E_w of G that has a k-subscripted element, $i_{G^*,Gk}(y^*) \neq z$. Then the execution step $G_{\mathrm{A}-1} \rightarrow G_\mathrm{A}$ was a rewrite, $y = i_{G\mathrm{A}-1,G\mathrm{A}}(x)$ for some $x \in G_{\mathrm{A}-1}$, and in this rewrite, x was the LHS but not the RHS of a redirection, was not activated, and was garbaged by the rewrite. And conversely.

Proof. A straightforward extension of lemma 5.3. ☺

Lemma 5.5 Let G be a suspending execution of a MONSTR system R. Let y be a node of graph G_k of G, and let k be such that for each element $(\mu_z\ z : \sigma_z)_k$ in every node event chain E_w of G that has a k-subscripted element, $y \neq z$. Then for any $q > k$, there is no q-subscripted element $(\ldots\ i_{Gk,Gq}(y)\ \ldots)_q$ in any node event chain of G.

Proof. Suppose not. We know that a node $y \in G_m$ of G first fails to be mentioned in m+1-subscripted elements of node event chains of G by being garbaged as the unactivated LHS but not RHS of redirections by lemma 5.4. To be mentioned once more at

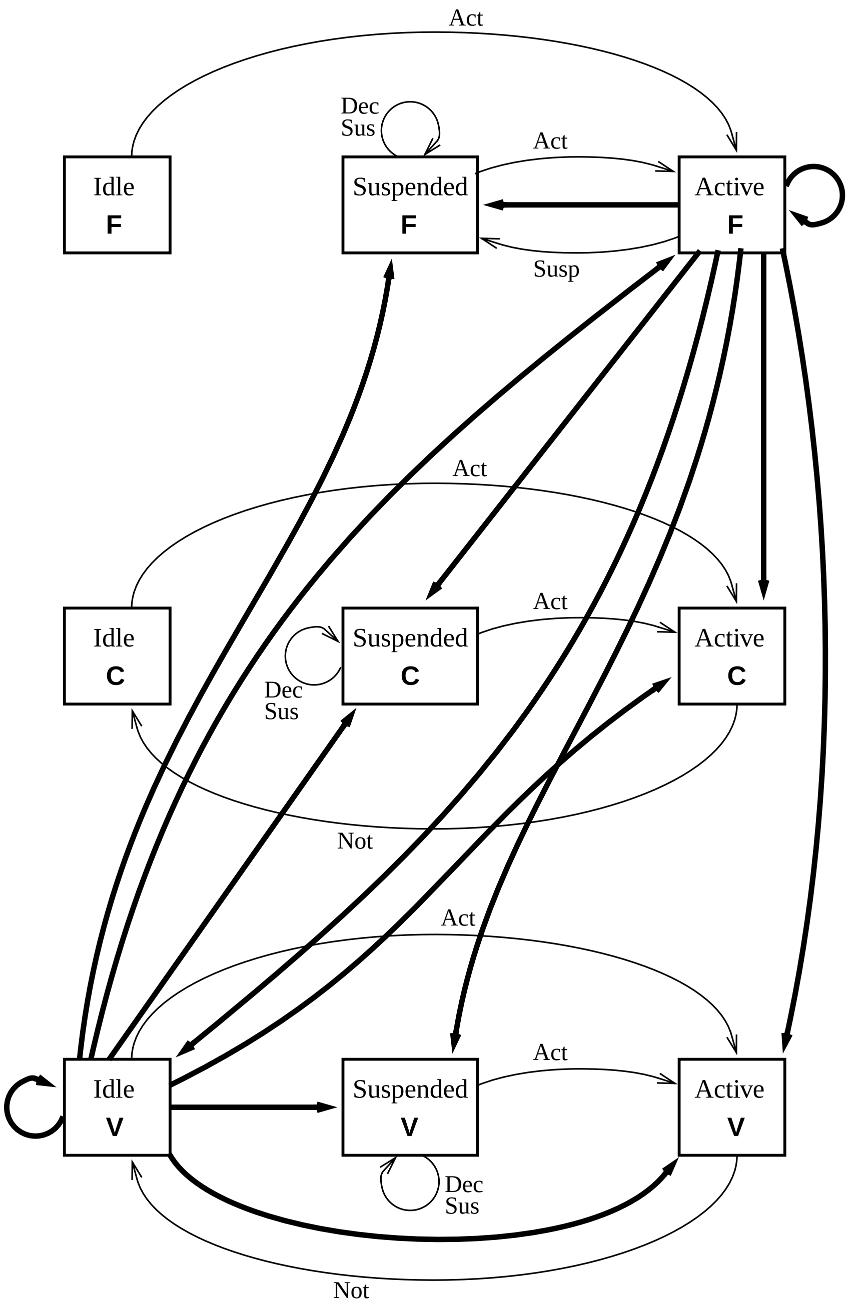

Fig. 7. State transition diagram for events in suspending semantics.

some later execution step, it would have to at least become the RHS of a redirection, as all other events relate to the $i_{Gm,Gm+1}$ images of nodes. To achieve this in turn, requires it to become a member of a standard redex, as the only alternative for a redirection RHS, a contractum node, can only occur in the rewrite that creates it. But all standard redexes are live, and garbage is persistent, so the proposal is impossible. We are done. ☺

Continuing the analogy with the packet store of **M-I**, we see that nodes like y in lemma 5.4, are those which in the packet store representation of rewriting in **M-I**.6, have their representing packets overwritten, without their packet contents simultaneously being moved to some other location; in other words, those packets which get "discarded without trace".

Definition 5.6 Let x be a node occurring in graph G_i in a suspending execution G of a MONSTR system R. We say that "x is eventually Π" (or similar), iff there is an $\mathrm{N} \geq i$ such that for all $j \geq \mathrm{N}$, property Π is true of the i_{G_i,G_j} image of x in G_j. Similarly let E_x be a node event chain for x in G. We say that "E_x is eventually Π", iff there is an N such that for all $j \geq \mathrm{N}$, property Π is true for all elements $(\ldots)_j$ in E_x.

Theorem 5.7 Let G be a non-terminating suspending MONSTR execution of a MONSTR system R. Suppose for all x occuring in graphs G_i of G, and for all node event chains E_y in G, the following hold.

(1) x is eventually not both the LHS of a redirection, and also either activated or the RHS of a redirection of a rewrite of G.

(2) E_y is eventually idle.

Then G is a pointwise correct execution of R with respect to DACTL semantics.

Proof. Consider a node x of the execution, first introduced in a graph G_A of G. Thus either $\mathrm{A} = 0$ and x is the initial node, or $\mathrm{A} > 0$ and x was introduced during contractum building in the rewrite $G_{\mathrm{A}-1} \rightarrow G_\mathrm{A}$. We must show that there is an N such that for $k \geq \mathrm{N}$, $i_{G_\mathrm{A},G_k}(x)$ is not in the EDG of G_k. There are three cases.

Case 1: There is an N such that for $k \geq \mathrm{N}$, $i_{G_\mathrm{A},G_k}(x)$ is not in the k'th element of any node event chain of G. Assume N is the smallest possible. Then by lemma 5.4 $i_{G_\mathrm{A},G_\mathrm{N}}(x)$ is garbage, by being the unactivated LHS of a redirection without being the RHS of a redirection, in the rewrite $G_{\mathrm{N}-1} \rightarrow G_\mathrm{N}$. In such a case $i_{G_\mathrm{A},G_\mathrm{N}}(x)$ is necessarily idle and balanced. By balancedness, all its out-arcs are also idle and garbage. Since it was not the RHS of a redirection, it has no in-arcs, thus no non-idle ones. So it is not in the EDG of G_N. For $k \geq \mathrm{N}$, this situation persists (for $i_{G_\mathrm{A},G_k}(x)$), as $i_{G_\mathrm{A},G_k}(x)$ continues to remain as idle garbage, to have only idle garbage out-arcs, and no in-arcs. This follows because nodes can only alter an idle marking or acquire in-arcs as the result of being in a standard redex, which requires them to be live.

Case 2: There is a B such that for $k \geq \mathrm{B}$, $i_{G_\mathrm{A},G_k}(x)$ is in some node event chain E_w whose suffix of elements with subscripts $\mathrm{B}+i$ for i non-negative, contains only events of types (a) – (h) (i.e. anything except redirections). So for each $\mathrm{B}+i$, the $\mathrm{B}+i$'th element of E_w contains $i_{G_\mathrm{A},G_{\mathrm{B}+i}}(x)$. By (2), there is an $\mathrm{N} \geq \mathrm{B}$ such that for i non-negative, the $\mathrm{N}+i$'th element of E_w, containing $i_{G_\mathrm{A},G_{\mathrm{N}+i}}(x)$, is idle. By balancedness, if $i_{G_\mathrm{A},G_{\mathrm{N}+i}}(x)$ was in the EDG of $G_{\mathrm{N}+i}$, it would have to be a stateholder with a notification in-arc from some suspended $p \in G_{\mathrm{N}+i}$. Such a p would have to remain suspended forever, since receiving the requisite notification entails $i_{G_\mathrm{A},G_{\mathrm{N}+i+j}}(x)$ being active for some non-negative j, an impossibility. Thus there is a node event chain, E_u say, containing the non-idle node $i_{G_{\mathrm{N}+i},G_{\mathrm{N}+i+j}}(p)$ in each N+i+j'th element of a suffix, which it is easy to see contains only identity events. But this contradicts (2), which says that E_u is eventually idle. So we conclude that $i_{G_{\mathrm{A}},G_{\mathrm{N}+i}}(x)$ was not in the EDG of $G_{\mathrm{N}+i}$ after all.

Case 3: Everything else. In this case, for all B, for all node event chains E_w say, that have $i_{G_{\mathrm{A}},G_{\mathrm{B}}}(x)$ in their B'th element, there is a non-negative i such that elements $(\ldots)_{\mathrm{B}+i}$ and $(\ldots)_{\mathrm{B}+i+1}$ of E_w depict a redirection event of rewrite $G_{\mathrm{B}+i} \to G_{\mathrm{B}+i+1}$. So $i_{G_{\mathrm{A}},G_{\mathrm{B}+i}}(x)$ is the LHS of a redirection. If $i_{G_{\mathrm{A}},G_{\mathrm{B}+i}}(x)$ were neither activated nor the RHS of some other redirection of the rewrite $G_{\mathrm{B}+i} \to G_{\mathrm{B}+i+1}$, we would be in case 1, as the rewrite would garbage $i_{G_{\mathrm{A}},G_{\mathrm{B}+i}}(x)$. So either $i_{G_{\mathrm{A}},G_{\mathrm{B}+i}}(x)$ is activated or is the RHS of a redirection. Since B was arbitrary, there must be infinitely many indices of G, $k_1 < k_2 < \ldots < k_i < \ldots$ at which the node $i_{G_{\mathrm{A}},G_{k_i}}(x)$ is both the LHS of a redirection, and also either activated or the RHS of a redirection. But this is precluded by (1), so case 3 is empty. We are done. ☺

Theorem 5.7 shows that under reasonable conditions, we can deduce correctness even for non-terminating suspending executions. The formulation of sufficient conditions in terms of node event chains and redirections, is frequently more convenient than definition 5.1, because the events in node event chains conform more closely to the way we naturally think of the destiny of nodes in an execution as evolving.

We examine a couple of examples of non-terminating systems.

Example 5.8 We consider a version of the producer consumer example of **M-I**.

S consists of

> **F** = {Producer, Consumer, Reader, Writer, Initial}
> **C** = {Item, Cons, Nil}
> **V** = {Empty, Full}

Rules:

```
Producer => *Cons[ Item *Producer ]  ;

Consumer[ Cons[ h t ] ] => #Consumer[ ^*t ]  ;

Consumer[ x ] => #Consumer[ ^*x ]  ;

Reader[ s:Full[ x ] ] => *Cons[ x #Reader[ ^y:Empty ] ] , s := *y   |
Reader[ s:Empty ] => #Reader[ ^s ]  ;

Reader[ x ] => #Reader[ ^*x ]  ;

Writer[ Cons[ h t ] s:Empty ] => #Writer[ ^*t u:Full[ h ] ] , s := *u   |
Writer[ x:Cons[ h t ] s:Full ] => #Writer[ x ^s ]  ;

Writer[ x y ] => ##Writer[ ^*x ^*y ]  ;

Initial =>  #Consumer[ ^#Reader[ ^s:Empty ] ] ,
            #Consumer[ ^#Reader[ ^s ] ] ,
            x:#Writer[ ^*Producer s ] ,
            y:#Writer[ ^*Producer s ]  ;
```

In this example, suspending semantics is actually needed for a smooth machine implementation as non-standard redexes can arise. For consider the first rules for both Reader and Writer; both of which redirect their stateholder to an active node. Because of the presence of multiple Readers and Writers, schedules exist in which Readers and Writers attempt to pattern match their stateholder argument while it is still active. Suspensions thus provide the natural way to handle this situation.

Another area in which suspending semantics can be exploited is in the rules that cause the lists of Items to be devoured by Consumers and Writers. If a Consumer wishes to consume an item which has not yet been read by the Reader it must wait, and likewise when a Writer wishes to write an item not yet produced by the Producer. This is achieved by having each freshly created Consumer or Writer node activate and suspend on its matched argument in the rules

```
Consumer[ Cons[ h t ] ] => #Consumer[ ^∗t ]

Writer[ Cons[ h t ] s:Empty ] => #Writer[ ^∗t u:Full[ h ] ] , s := ∗u
```

The programmed suspension ensures that such Consumers and Writers only wake once the relevant argument is in Cons form. However there are races between Consumers and their Readers and Writers and their Producers. If the latter usually stay ahead of the former so that the Consumers and Writers usually have their argument in Cons form anyway, the activation and suspension of the above rules waste work and the runtime suspension mechanism provides a more efficient synchronisation allowing the rules to be replaced by

```
Consumer[ Cons[ h t ] ] => ∗Consumer[ t ]

Writer[ Cons[ h t ] s:Empty ] => ∗Writer[ t u:Full[ h ] ] , s := ∗u
```

We can verify that (either version of) the above system is pointwise correct if we make the conventional fairness assumption that all active nodes eventually become chosen roots. The key is to note that the live part of any execution graph is a semitree, i.e. there is a unique non-backtracking unoriented live semipath[1] between any two live nodes of any execution graph. This is trivially true for the initial graph and it is easy to check that it is preserved by all the rules, being trivially preserved by notifications and suspensions. This allows us to quickly see that no LHS of a redirection is ever activated. Furthermore it is clear from a superficial examination of the rules that the RHS of every redirection is a contractum node. Thus no redex node can be the LHS of a redirection and either the RHS of a redirection or activated, even once let alone infinitely often. So theorem 5.7.(1) holds. To establish theorem 5.7.(2) we need a rather more intricate inductive argument which we will not describe in detail. Assuming fairness, this argument needs to prove that: (1) each Item node is created idle and remains so forever; (2) each Cons node is created active, later notifies, possibly repeats an activation/notification cycle and eventually becomes idle garbage; (3) each Empty node and each Full node is created active, later notifies, and eventually is redirected, becoming idle garbage; (4) each function node is created suspended and eventually receives the requisite number of notifications and becomes active or is created active already, and rewrites, eventually becoming idle garbage. [Fig. 8] which shows an execution graph of the system, makes all of these statements at least plausible. These in turn are enough for theorem 5.7.(2), and hence for pointwise correctness.

1. We recall that a semipath in a directed graph just disregards the orientation of the arcs.

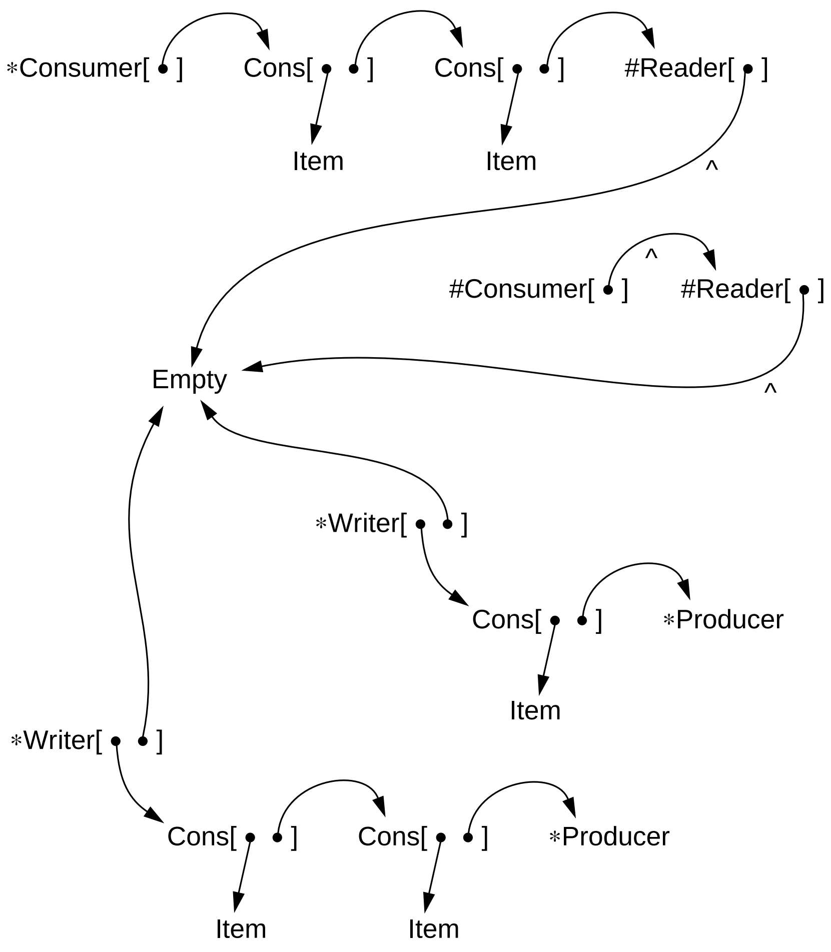

Fig. 8. An execution graph of Producers and Consumers.

Example 5.9 In this example we display a MONSTR system that does not satisfy the conditions of theorem 5.7. All of the rewrites of this system are in fact instances of the example that we used to illustrate rewriting in [Section 2], [Fig. 4]. The one possible execution of the system, has two stateholder nodes that are simultaneously the LHS and RHS of redirections infinitely often. This property is used to ensure that neither of them is eventually in every subsequent element of some node event chain (though they occur in some node event chains infinitely many times), and this helps to ensure that they are repeatedly in the EDG. Ironically, though the execution of this system is not pointwise correct according to definition 5.1, we would much prefer to regard it as correct in some wider sense, as it is in fact a DACTL execution; containing no suspension steps at all. This merely illustrates the obvious fact that non-terminating executions can in general be trickier to deal with than terminating ones.

S consists of

F = {F, Initial}
C = ∅
V = {S}

Rules:

```
root:F[ s:S a ] => s , s := ∗a , #F[ ^a root ]  ;

F[ a b ] => #F[ ^∗a b ]  ;

Initial => ∗F[ s:S v:S ]  ;
```

[Fig. 9] illustrates the execution of this system showing only the live nodes in each execution graph, (note that the difference between the top right and bottom left graphs of [Fig. 9] and the result graph of [Fig. 4], is that [Fig. 4] includes the idle F-labelled node which is garbage). In relation to this note also that while the F nodes are constantly being garbaged and replenished, the S nodes, s and v, are the same nodes in each graph (i.e. they are $i_{Gk,Gk+1}$ images rather than contractum instantiations), as a result of the phenomena pointed out above. Consequently, there is a node event chain E_{s} say, that perpetually contains one or other of these s and v nodes alternately, and each picture of the cyclic part of the execution in the figure, indicates which node is in E_{s} by the ♠. As usual, dashed arrows represent redirections to be performed in the next step, and it is clear that each suffix of E_{s} contains an infinite number of redirection events where the redirected node is the RHS as well as the LHS of redirections, because both s and v visit E_{s} infinitely many times.

6 INDEPENDENCE

In this section, we examine the conditions under which the order of adjacent execution steps of suspending executions is irrelevant. This is of interest when we wish to reason about the correctness of specific systems. In general the relatively finegrained nature of MONSTR rules means that larger atomic actions at a higher level of abstraction, may have to be broken down into smaller subactions. Often this can generate a certain amount of concurrency in the execution graph, as independent subactions are in principle able to execute simultaneously — which is modelled by interleaving them in either order in the serial rewriting model. Combining such independence with any parallelism in the original system can yield an astronomical number of execution sequences to consider, especially when executions are infinite. Often large numbers of these differ only in trivial ways, and the interchange theorems of this section help with this, by allowing executions that differ only by sequences of permitted interchanges to be regarded as equivalent. Ultimately one may be able to reduce all the executions to a standard form if the understanding of the system is profound enough. For a concrete example of this process see [Banach et al. (1995)].

Actually, to just say that independent actions may simply be interchanged is too simplistic. In one case, choosing one action first obliterates the other; in another, one choice entails an extra action. And in any case, the results we start with in this section are expressed in a symmetric form which states that given an execution graph for which

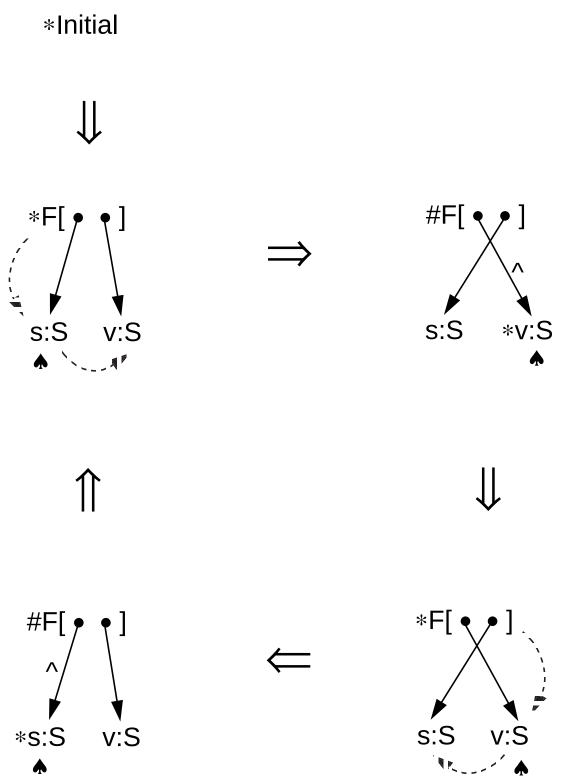

Fig. 9. A non-terminating execution of the system featuring two nodes that are repeatedly in the EDG.

two candidate next execution steps are available, one may perform them in either order with equivalent results (provided suitable conditions hold). We return to these points later.

Theorem 6.1 Let $\mathcal{G}_N = [G_0, G_1, \ldots, G_N]$ be a prefix of a suspending execution of a MONSTR system $\mathcal{R}$. 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) $r_{G_N,H_i}(t_j) = i_{G_N,H_i}(t_j)$ 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 $r_{G_N,H_i}(t_j)$. Then

(3) K_1 and K_2 are marking preserving isomorphic.

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 $r_{G_N,H_i}(t_j)$ 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 $r_{G_N,H_j}(t_i)$; 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 theorem. ☺

Theorem 6.2 Let $\mathcal{G}_N = [G_0, G_1, \ldots, G_N]$ be a prefix of a suspending execution of a MONSTR system $\mathcal{R}$. Suppose G_N contains two active function nodes $s_1 \neq s_2$ with not all Map($\sigma(s_i)$) arguments idle, i.e. such that for $i \in \{1, 2\}$ there is at least one $k_i \in$ Map($\sigma(s_i)$) such that $\alpha(s_i)[k_i]$ is not idle in G_N. 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_N. Then

(1) H_1 and H_2 are graph structure isomorphic.

(2) $r_{G_N,H_i}(s_j) = i_{G_N,H_i}(s_j)$ 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 $r_{G_N,H_i}(s_j)$. Then

(3) K_1 and K_2 are marking preserving isomorphic.

Proof. This is pretty similar to theorem 6.1, 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 marking that is changed in a suspension step is that of the suspension root, and all nodes are non-idle afterwards iff they were non-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_N, we get (3) easily. ☺

Theorem 6.3 Let $\mathcal{G}_N = [G_0, G_1, \ldots, G_N]$ be a prefix of a suspending execution of a MONSTR system $\mathcal{R}$. Suppose G_N contains an active function node s such that there is at least one $k \in$ Map($\sigma(s)$) such that $\alpha(s)[k]$ is not idle in G_N. Suppose G_N also contains an active constructor or stateholder t. Let

Susp = $\{k \in \text{Map}(\sigma(s)) \mid \alpha(s)[k] \text{ is non-idle in } G_N\}$
SuspNodes = $\{x \in G_N \mid x = \alpha(s)[k] \text{ for some } k \in \text{Susp}\}$

$\Pi \equiv \text{SuspNodes} = \{t\}$

Let H_s be obtained by performing a suspension from s in G_N, and let H_t be obtained by performing a notification from t in G_N. Then

(1) H_s and H_t are graph structure isomorphic.

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

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

Then

(3) K_s and K_t are marking preserving isomorphic.

Proof. As in the previous theorems, (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-root nodes of suspension redexes do not change during suspensions, we conclude (2), noting that if s's only non-idle Map($\sigma(s)$) argument was t, 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), if the suspension is done first, the constituent arcs become notification arcs of H_s, and next become normal arcs of K_s after the notification. If the notification is done first, these arcs disappear from the suspension redex in H_t since $r_{G_{\mathrm{N}},H_t}(t)$ is idle, remaining normal in the suspension step that follows if the suspension redex in G_{N} contained other than case (a) 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 (a) arcs. 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 (a) arcs in the following notification. Obviously if the suspension redex consists solely of case (a) arcs and their nodes, then all suspensions 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 non-idle Map($\sigma(s)$) arguments 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). ☺

Theorem 6.4 Let $\mathcal{G}_N = [G_0, G_1, \ldots, G_N]$ be a prefix of a suspending execution 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 $\mathrm{Map}(\sigma(f))$ arguments are idle, and which is thus the root of a standard redex $g : L \to G_N$ for some rule $D = (P, root, Red, Act)$. Let

$$\Pi \;\equiv\; t \in g(Act)$$

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 $g' : P \to G_N'$, $g'' : P \to G_N''$, $h_f : P \to H_f$, and associated i and r maps. Then

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

(b) $r_{G_N,H_t}(f) = i_{G_N,H_t}(f)$ is an active function node, and

$$h_t = r_{G_N,H_t} \circ g : L \to H_t$$

is a standard redex for D, such that all of the $\mathrm{Map}(\sigma(r_{G_N,H_t}(f)))$ arguments of $r_{G_N,H_t}(f)$ are idle, hence is the redex of a potential rewrite in H_t.

Let K_f be obtained from H_f by performing a notification from $r_{G_N,H_f}(t)$. Let J_t be obtained from H_t by rewriting the redex rooted at $r_{G_N,H_t}(f)$ 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** $r_{G_N,J_t}(t)$ 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 $r_{G_N,J_t}(t)$ in J_t

Then

(3) K_f and K_t are marking preserving isomorphic.

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, $r_{G_N,H_t}(f)$ 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

$$\theta : G_N \to H_t$$

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

$$\theta' : G_N' \to H_t'$$

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

$$\theta'' : G_N'' \to H_t''$$

such that the triangle involving the node maps $g'' : P \to G_N''$ 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 of rewriting (**M-I**.3.10) shows that the only active node of the rewritten graph that can possibly end up idle in the result, is the root of the redex (if it is not one of the activated nodes). 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 $t \neq f$, $r_{G_N,H_f}(t)$ 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 four disjoint cases: (a) t; (b) nodes of the notification redex other than t; (c) contractum nodes; (d) 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 $t \notin g(Act)$, or is active in J_t if $t \in g(Act)$, giving us (2). In the latter case, $r_{G_N,J_t}(t)$ is a notification root in J_t, and doing the notification, makes it idle in K_t, as required.

For case (b) 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 (c), 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 (c1) concerns all notification out-arcs of q whose child node is (a node whose g' image, resp. h_t' image, is the $r_{G_N,G_N'}$ image, resp. the $r_{G_N,H_t'}$ image, of) t, or whose child node is the LHS of a redirection $(a, b) \in Red$ where the RHS node is (a node whose g' image, resp. h_t' image, is the $r_{G_N,G_N'}$ image, resp. the $r_{G_N,H_t'}$ image, of) t. If there are such notification out-arcs, then we have Π by **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 (c2) 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 (c1) out-arcs of $h_t'(q)$ is idle in H_t', but active in J_t, whereupon $j_t(q)$ receives notifications along the (c1) out-arcs which decrease its suspension marking in K_t. (N.B. Because of the earlier notification from t, the only suspended parents that $r_{G_N,J_t}(t)$ has, are the parent nodes of these (c1) out-arcs.) If rewriting is done first, the child node of (c1) out-arcs of $g'(q)$ is active in G_N', hence in H_f, whereupon the (c1) out-arcs join the image of the notification redex in H_f. $h_f(q)$ therefore receives notifications along the (c1) 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 (c1) out-arcs, the markings on them in K_t and K_f are the same. The (c2) out-arcs do not affect the node markings of contractum nodes.

Finally for case (d) 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 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 (c1) subcase of case (c) 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. ☺

Theorem 6.5 Let $\mathcal{G}_N = [G_0, G_1, \ldots, G_N]$ be a prefix of a suspending execution of a MONSTR system $\mathcal{R}$. Suppose G_N contains an active function node s such that there is at least one $k \in \mathrm{Map}(\sigma(s))$ such that $\alpha(s)[k]$ is not idle in G_N. Suppose G_N also contains an active function node f, all of whose $\mathrm{Map}(\sigma(f))$ arguments are idle, and which is thus the root of a standard redex $g : L \to G_N$ for some rule $D = (P, root, Red, Act)$. Let

$$\mathrm{Susp} = \{k \in \mathrm{Map}(\sigma(s)) \mid \alpha(s)[k] \text{ is non-idle in } G_N\}$$
$$\mathrm{SuspNodes} = \{x \in G_N \mid x = \alpha(s)[k] \text{ for some } k \in \mathrm{Susp}\}$$

$$\overline{\mathrm{Susp}} = \{k \in \mathrm{Map}(\sigma(s)) \mid \alpha(s)[k] \text{ is idle in } G_N\}$$
$$\overline{\mathrm{SuspNodes}} = \{x \in G_N \mid x = \alpha(s)[k] \text{ for some } k \in \overline{\mathrm{Susp}}\}$$

$$\mathrm{ActNodes} = g(Act)$$

Let $(root, b) \in Red$ be the root redirection of D. Suppose

(A) $f \notin \mathrm{SuspNodes}$ or $\mu(b) \neq \varepsilon$ or $b \in Act$

(B) $\mathrm{ActNodes} \cap \overline{\mathrm{SuspNodes}} = \varnothing$

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

(1) (a) $r_{G_N,H_f}(s) = i_{G_N,H_f}(s)$ is an active function node of H_f such that there is at least one $k \in \mathrm{Map}(\sigma(r_{G_N,H_f}(s)))$ such that $\alpha(r_{G_N,H_f}(s))[k]$ is not idle in H_f. Hence $r_{G_N,H_f}(s)$ the root of a potential suspension step in H_f.

(b) $r_{G_N,H_s}(f) = i_{G_N,H_s}(f)$ is an active function node, and

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

is a standard redex for D, such that all of the Map($\sigma(r_{G_{\mathrm{N}},H_s}(f))$) arguments of $r_{G_{\mathrm{N}},H_s}(f)$ are idle, hence is the redex of a potential rewrite in H_s.

Let K_f be obtained from H_f by performing a suspension from $r_{G_{\mathrm{N}},H_f}(s)$. Let K_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 $h_s' : P \to H_s'$, $h_s'' : P \to H_s''$, $k_s : P \to K_s$, and associated i and r maps. Then

(2) K_f and K_s are marking preserving isomorphic.

Proof. Obviously $f \neq s$ since f's Map($\sigma(f)$) arguments are idle and s's aren't. Equally obviously, f's Map($\sigma(f)$) arguments do not include any suspension redex nodes since the latter are non-idle. However, this does not prevent the suspension redex nodes from occurring as implicitly matched arguments of the root of the rewriting redex. Because 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 node whose marking changes at all is s itself, $r_{G_{\mathrm{N}},H_s}(f)$ 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

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

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

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

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

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

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

$$\theta''' : H_f \to K_s$$

such that the triangle involving $h_f : P \to H_f$ and $k_s : P \to K_s$ commutes.

As in the previous theorem, the definition of rewriting (**M-I**.3.10) shows that the only active node of the rewritten graph that can possibly end up idle in the result, is the root of the redex (if it is not one of the activated nodes). 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 there is at least one Map($\sigma(r_{G_{\mathrm{N}},H_f}(s))$) argument of $r_{G_{\mathrm{N}},H_f}(s)$ in H_f which is non-idle. If $\{f\} \neq$ SuspNodes, then since SuspNodes is nonempty, there is at least one node $f \neq y \in$ SuspNodes. Evidently y is non-idle in G_{N}, so $r_{G_{\mathrm{N}},H_f}(y) = i_{G_{\mathrm{N}},H_f}(y)$ is non-idle in H_f, and we are done. Otherwise by (A), we conclude that $r_{G_{\mathrm{N}},H_f}(f)$ is non-idle in H_f, and therefore H_f has the arc $(i_{G_{\mathrm{N}},H_f}(s)_m, r_{G_{\mathrm{N}},H_f}(f))$ for some $m \in$ Map($\sigma(i_{G_{\mathrm{N}},H_f}(s))$), which witnesses what we need.

To get (2) we must follow what happens to the markings on the nodes and arcs.

For arcs there are four 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; (c) all arcs (s_k, x) not in the suspension redex, but with $k \in$ Map(σ(s)); (d) all remaining arcs, whether already existing in G_{N}, or introduced during rewriting, (this includes all arcs (s_k, x), for any $k \notin$ Map(σ(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 by (A), they become notification arcs after the suspension. If the suspension is done first, they become notification arcs immediately, and remain so during the rewrite.

For case (b) arcs, they are unaffected by rewriting, and become notification arcs after the suspension, regardless of the order of steps. For case (c) arcs, if the rewrite is first, they remain normal, and because of (B), their child nodes are not activated, whence they remain normal after the suspension. If the suspension is first, they remain normal through both the suspension and rewrite. For case (d) arcs, they retain the marking they had in G_{N}, or were given during contractum building, regardless of the order of steps.

For nodes there are five disjoint cases: (a) s; (b) f; (c) nodes in SuspNodes other than f (if applicable); (d) nodes in $\overline{\text{SuspNodes}}$; (e) 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, or perhaps reactivated during the rewrite. This holds regardless of the order of the steps. For the case (c) nodes, they start off non-idle, and remain so, regardless of the order of steps, being unaffected by any activations from the rewrite. For the case (d) nodes, we know by (B) that they are not activated, and this holds regardless of the order of steps, so they remain idle either way. For case (e) 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 arcs to case (c) nodes plus arcs to f if f is a Map(σ(s)) argument of s. The marking remains during the rewriting step. If rewriting is done first, the Map(σ(s)) argument arcs to f (if applicable), become redirected to a non-idle node by (A), and the other Map(σ(s)) argument arcs to case (c) and case (d) nodes remain, as neither case (c) nor case (d) nodes are redirected. Since case (c) nodes are all non-idle before the rewrite and therefore non-idle after the rewrite, and case (d) nodes are all idle before the rewrite and by (B) idle after the rewrite, the the same number of suspensions are generated as for the other order of steps. We are done. ☺

The final result of this section, addressing the interchange of two rewrites, is the most complex. Unlike the previous results which are “optimal” in a fairly clear sense, the following theorem is not, in that some straightforward extensions may easily be imagined. We comment on some of these below.

Theorem 6.6 Let $\mathcal{G}_{\mathrm{N}} = [G_0, G_1, \ldots, G_{\mathrm{N}}]$ be a prefix of a suspending execution 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(σ(f_i)) arguments of f_i are idle, and suppose therefore that f_i is the root of a standard redex $g_i : L_i \rightarrow G_{\mathrm{N}}$ for some rule $D_i = (P_i, root_i, Red_i, Act_i)$.

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

$$\begin{aligned}
&\mathrm{ActNodes}_i = g_i(Act_i)\\
&\mathrm{MapNodes}_i = \{x \in G_\mathrm{N} \mid x = \alpha(f_i)[k] \text{ for some } k \in \mathrm{Map}(\sigma(f_i))\}\\
&\mathrm{RedNodes}_i = \{x \in G_\mathrm{N} \mid x = g_i(a) \text{ for } (a, b) \in Red_i \text{ such that } g_i(a) \neq g_i(b)\}
\end{aligned}$$

Suppose

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

(B) For either $i \in \{1, 2\}$, D_i redirects $s_i \;\Rightarrow\;$ [$v_i \notin \mathrm{MapNodes}_j$ and $v_j \notin \mathrm{MapNodes}_i$]

(C) $(\mathrm{ActNodes}_i \cup \mathrm{ActNodes}_j) \cap (\{f_i, f_j\} \cup \mathrm{MapNodes}_i \cup \mathrm{MapNodes}_j) = \varnothing$

(D) For both $i \in \{1, 2\}$,

$$\begin{aligned}
\varnothing = \{(x, y, z) \in {G_\mathrm{N}}^3 \mid\ & x \in \mathrm{RedNodes}_i \text{ and } y \in \mathrm{RedNodes}_j \text{ and } z \in \mathrm{RedNodes}_i\\
\text{and } x \neq z \ \ & \text{and } (x, y) = (g_i(a), g_i(b)) \text{ for } (a, b) \in Red_i\\
& \text{and } (y, z) = (g_j(c), g_j(d)) \text{ for } (c, d) \in Red_j\}
\end{aligned}$$

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

Then

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

$$h_i = r_{G_\mathrm{N},H_i} \circ g_j : L_j \to H_i$$

is a standard redex for D_j, such that all of the $\mathrm{Map}(\sigma(r_{G_\mathrm{N},H_i}(f_j)))$ arguments of $r_{G_\mathrm{N},H_i}(f_j)$ are idle, hence is the redex of a potential rewrite in H_i.

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

Then

(2) K_1 and K_2 are marking preserving isomorphic.

Proof. Since $f_i \neq f_j$, both are active, and both have all $\mathrm{Map}(\sigma(f_i))$ arguments idle in G_N, we have $f_i \notin \mathrm{MapNodes}_j$ and vice versa. If both D_i and D_j are normal rules and both feature a stateholder in L_i and L_j, then, if $v_i \neq v_j$, then $\mathrm{MapNodes}_i \cap \mathrm{MapNodes}_j$ consists only of constructors. If $v_i = v_j$, then $\mathrm{MapNodes}_i \cap \mathrm{MapNodes}_j$ can include $v_i = v_j$ provided neither is redirected. If D_i is a normal rule but D_j is a default rule, then $\mathrm{MapNodes}_i \cap \mathrm{MapNodes}_j$ consists only of constructors, and $\mathrm{MapNodes}_j - \mathrm{MapNodes}_i$ can include idle nodes with arbitrary symbols; and vice versa. If both D_i and D_j are default rules, $\mathrm{MapNodes}_i \cup \mathrm{MapNodes}_j$ can contain arbitrarily labelled idle nodes. And in any event, nothing precludes the nodes in $\{f_i\} \cup \mathrm{MapNodes}_i$ from occuring as implicitly matched non-$\mathrm{MapNodes}_j$ nodes of the D_j redex provided they are not activated by D_j, and vice versa.

Suppose the D_i redex is rewritten first. Since $f_i \neq f_j$ and $v_i \neq f_j$, f_j cannot be redirected by the D_i rewrite. Also since f_j is active, f_j's marking is unaffected by the D_i rewrite. Therefore $r_{G_N,H_i}(f_j) = i_{G_N,H_i}(f_j)$ is active in H_i.

Let $x \in$ MapNodes$_j$ and suppose D_j is a normal rule; consequently $\sigma(x) \in \mathbf{C} \cup \mathbf{V}$. If $\sigma(x) \in \mathbf{C}$, x is not redirectable, thus not redirected in the D_i rewrite. Thus, $r_{G_N,H_i}(x) = i_{G_N,H_i}(x)$, $\sigma(r_{G_N,H_i}(x)) = \sigma(i_{G_N,H_i}(x))$, and so an arc $(root_m, c)$ of L_j can be matched to the arc $(i_{G_N,H_i}(f_j)_m, r_{G_N,H_i}(x)) = (r_{G_N,H_i}(f_j)_m, r_{G_N,H_i}(x))$ in H_i, iff it can be matched to the arc $((f_j)_m, x)$ in G_N. If $\sigma(x) \in \mathbf{V}$, then assuming that both rules feature a stateholder, by (A) either $v_i \neq v_j$ and the same conclusion holds since the D_i rewrite cannot redirect v_j; or alternatively $v_i = v_j$ and neither rewrite redirects $v_i = v_j$, and the conclusion holds also. Clearly if one or other rule does not feature a stateholder, then we draw the same conclusion even more trivially.

So the explicit nodes of L_j and the arcs connecting them can be matched by $r_{G_N,H_i} \circ g_j$. Since by the MONSTR restrictions on patterns (**M-I**.11.4.(4)), no constraints are placed on any implicit nodes of L_j (or their in-arcs) to achieve a match, $r_{G_N,H_i} \circ g_j$ matches all of L_j to H_i. So $r_{G_N,H_i} \circ g_j : L_j \to H_i$ is a redex for D_j in H_i. Since by (C), there is no overlap between the nodes activated in the D_i rewrite and $(r_{G_N,H_i} \circ g_j)(\{f_j\} \cup$ MapNodes$_j)$, $r_{G_N,H_i} \circ g_j : L_j \to H_i$ is a standard redex for D_j in H_i, giving us (1) for this case.

If D_j is a default rule we need no specific graph structure constraints to be satisfied for matching, as a default rule for a function symbol will always have a redex whenever there is a suitably labelled active node. Hence $r_{G_N,H_i} \circ g_j : L_j \to H_i$ is a redex for D_j in H_i; indeed a standard redex. Not that this is adequate by itself, since we still need the Map($\sigma(f_j)$) arguments of the r_{G_N,H_i} image of f_j to be idle in H_i else a suspension would be forced if $r_{G_N,H_i}(f_j)$ attempted to rewrite in H_i. By (B) v_i is only in MapNodes$_j$ if it is not redirected by D_i, so if it is in MapNodes$_j$, it remains idle by (C), so we have (1) by reasoning analogous to that used above. The remainder of (1) is a matter of symmetry.

It remains to establish the marking preserving isomorphism claimed in (2), which we do in 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. Thus

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

where

$$\theta(i_{G_N,K_1}(x)) = i_{G_N,K_2}(x) \quad \text{for} \quad x \in G_N$$
$$\theta(i_{H_1,K_1}(g_1'''(p_1))) = k_2(p_1) \quad \text{for} \quad p_1 \in N_{P_1} - N_{L_1}$$
$$\theta(k_1(p_2)) = i_{H_2,K_2}(g_2'''(p_2)) \quad \text{for} \quad p_2 \in N_{P_2} - N_{L_2}$$

This is a bijection assuming that a sufficiently fussy construction for disjoint union during contractum building has ensured all introduced nodes are distinct.

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, and then bring the two together in a third stage. So each arc is covered by one of the head cases and one of the tail cases.

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 three cases: (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 K_i depending on order of rewriting.

Now we note straight away that for case (b$_i$), for an instantiated contractum node x, say $x = w(c_i)$, for either version of W where $w : P_i \rightarrow W$, 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 RedNodes$_i$, and the second rewrite, perforce of D_j, only redirects nodes in $r_{G_{\mathrm{N}},H_i}(\text{RedNodes}_j) = i_{G_{\mathrm{N}},H_i}(\text{RedNodes}_j)$. Neither of these includes any instantiated contractum nodes. Thus $\theta(r_{H_i,K_i}(g_i'''(c_i))) = \theta(i_{H_i,K_i}(g_i'''(c_i))) = k_j(c_i)$ and symmetrically, as required.

For a case (a) node x there are three subcases: (a.1) $x \notin (\text{RedNodes}_i \cup \text{RedNodes}_j)$; (a.2$_i$) $x \in \text{RedNodes}_i$. (That $\text{RedNodes}_i \cap \text{RedNodes}_j = \varnothing$ follows easily from the hypotheses.)

For subcase (a.1), suppose D_i rewrites first. We have $(\{x\} \cup \text{RedNodes}_j) \cap \text{RedNodes}_i = \varnothing$ so that $r_{G_{\mathrm{N}},H_i}(y) = i_{G_{\mathrm{N}},H_i}(y)$ for all $y \in \{x\} \cup \text{RedNodes}_j$. Consequently $i_{G_{\mathrm{N}},H_i}(x)$ is not redirected in the subsequent D_j rewrite, 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 subcase (a.2$_i$), suppose D_i rewrites first, giving subsubcase (a.2$_i$.i). By assumption, $r_{G_{\mathrm{N}},H_i}(x) \neq i_{G_{\mathrm{N}},H_i}(x)$. There are three sub…cases: (a.2$_i$.i.C) in which $r_{G_{\mathrm{N}},H_i}(x) = g_i'''(c_i)$, where c_i is a contractum node of the rule D_i; (a.2$_i$.i.I) in which $r_{G_{\mathrm{N}},H_i}(x) = i_{G_{\mathrm{N}},H_i}(y_i)$, where $y_i \notin \text{RedNodes}_j$; (a.2$_i$.$i$.R) in which $r_{G_{\mathrm{N}},H_i}(x) = i_{G_{\mathrm{N}},H_i}(y_i)$, where $y_i \in \text{RedNodes}_j$.

For sub…case (a.2$_i$.i.C), we calculate $r_{G_{\mathrm{N}},K_i}(x) = i_{H_i,K_i}(g_i'''(c_i))$, by case (b$_i$). For sub…case (a.2$_i$.$i$.I), we calculate $r_{G_{\mathrm{N}},K_i}(x) = r_{H_i,K_i}(r_{G_{\mathrm{N}},H_i}(x)) = r_{H_i,K_i}(i_{G_{\mathrm{N}},H_i}(y_i)) = i_{H_i,K_i}(i_{G_{\mathrm{N}},H_i}(y_i)) = i_{G_{\mathrm{N}},K_i}(y_i)$, since $y_i \notin \text{RedNodes}_j$.

For sub…case (a.2$_i$.i.R), with $r_{G_{\mathrm{N}},H_i}(x) = i_{G_{\mathrm{N}},H_i}(y_i)$, where $y_i \in \text{RedNodes}_j$, there are two sub…cases: (a.2$_i$.i.R.C) in which $r_{H_i,K_i}(i_{G_{\mathrm{N}},H_i}(y_i)) = k_i(d_i)$ where d_i is a contractum node of the rule D_j; (a.2$_i$.i.R.I) in which $r_{H_i,K_i}(i_{G_{\mathrm{N}},H_i}(y_i)) = i_{G_{\mathrm{N}},K_i}(z_i)$, with $z_i \in G_{\mathrm{N}}$. In the latter case, we know by (D) that the redirection target of y_i in the redex $g_j : L_j \rightarrow G_{\mathrm{N}}$ is not in RedNodes$_i$.

For sub…case (a.2$_i$.i.R.C), we calculate $r_{G_{\mathrm{N}},K_i}(x) = r_{H_i,K_i}(r_{G_{\mathrm{N}},H_i}(x)) = r_{H_i,K_i}(i_{G_{\mathrm{N}},H_i}(y_i)) = k_i(d_i)$, by case (b$_i$). For sub…case (a.2$_i$.$i$.R.I), we calculate $r_{G_{\mathrm{N}},K_i}(x) = r_{H_i,K_i}(r_{G_{\mathrm{N}},H_i}(x)) = r_{H_i,K_i}(i_{G_{\mathrm{N}},H_i}(y_i)) = i_{G_{\mathrm{N}},K_i}(z_i)$ for $z_i \in G_{\mathrm{N}}$.

Now suppose D_j rewrites first, giving subsubcase (a.2$_i$.j). Since $x \in \text{RedNodes}_i$ and $\text{RedNodes}_i \cap \text{RedNodes}_j = \varnothing$, $r_{G_{\mathrm{N}},H_j}(x) = i_{G_{\mathrm{N}},H_j}(x)$, and so $r_{G_{\mathrm{N}},K_j}(x) = r_{H_j,K_j}(i_{G_{\mathrm{N}},H_j}(x))$. There are now three sub…cases: the first is (a.2$_i$.j.C) in which $r_{H_j,K_j}(i_{G_{\mathrm{N}},H_j}(x)) = k_j(c_j)$, where c_j is a contractum node of the rule D_i. Alternatively, if the redirection target of $i_{G_{\mathrm{N}},H_j}(x)$ is not an instantiated contractum node, then it must be the r_{G_{N},H_j} image of a node of G_{N} by (1). So the other two cases are: (a.2$_i$.j.I) in which $r_{H_j,K_j}(i_{G_{\mathrm{N}},H_j}(x)) =$

$i_{H_j,K_j}(r_{G_N,H_j}(y_j))$, where $y_j \notin$ RedNodes$_j$; and (a.2$_i$.j.R) in which $r_{H_j,K_j}(i_{G_N,H_j}(x)) = i_{H_j,K_j}(r_{G_N,H_j}(y_j))$, where $y_j \in$ RedNodes$_j$.

For sub…case (a.2$_i$.j.C), we calculate $r_{G_N,K_j}(x) = k_j(c_j)$, by case (b$_j$). For sub…case (a.2$_i$.$j$.I), we calculate $r_{G_N,K_j}(x) = r_{H_j,K_j}(i_{G_N,H_j}(x)) = i_{H_j,K_j}(r_{G_N,H_j}(y_j)) = i_{H_j,K_j}(i_{G_N,H_j}(y_j)) = i_{G_N,K_j}(y_j)$, since $y_j \notin$ RedNodes$_j$.

For sub…case (a.2$_i$.j.R), with $r_{H_j,K_j}(i_{G_N,H_j}(x)) = i_{H_j,K_j}(r_{G_N,H_j}(y_j))$, where $y_j \in$ RedNodes$_j$, there are two sub…cases: (a.2$_i$.j.R.C) in which $i_{H_j,K_j}(r_{G_N,H_j}(y_j)) = i_{H_j,K_j}(g_j'''(d_j))$ where d_j is a contractum node of the rule D_j; (a.2$_i$.j.R.I) in which $i_{H_j,K_j}(r_{G_N,H_j}(y_j)) = i_{H_j,K_j}(i_{G_N,H_j}(z_j))$ with $z_j \in G_N$. In the latter case, we know by (D) that the redirection target of y_j in the redex $g_j : L_j \to G_N$ is not in RedNodes$_i$.

For sub…case (a.2$_i$.j.R.C), we calculate $r_{G_N,K_j}(x) = r_{H_j,K_j}(i_{G_N,H_j}(x)) = i_{H_j,K_j}(r_{G_N,H_j}(y_j)) = i_{H_j,K_j}(g_j'''(d_j))$, by case (b$_j$). For sub…case (a.2$_i$.$j$.R.I), we calculate $r_{G_N,K_j}(x) = r_{H_j,K_j}(i_{G_N,H_j}(x)) = i_{H_j,K_j}(r_{G_N,H_j}(y_j)) = i_{H_j,K_j}(i_{G_N,H_j}(z_j)) = i_{G_N,K_j}(z_j)$ for $z_j \in G_N$.

The eight possibilities above pair up nicely when we interchange the order of rewriting in a given sequence. The following describes what happens.

When we interchange the order in case (a.2$_i$.i.C), we get an instance of case (a.2$_i$.j.C) and vice versa. The nodes c_i and c_j are identified, and the second and third clauses for θ show that $\theta(r_{G_N,K_1}(x)) = r_{G_N,K_2}(x)$ as required.

When we interchange the order in case (a.2$_i$.i.I), we get an instance of case (a.2$_i$.j.I) and vice versa. The nodes y_i and y_j are identified, and the first clause for θ shows that $\theta(r_{G_N,K_1}(x)) = r_{G_N,K_2}(x)$ as required.

When we interchange the order in case (a.2$_i$.i.R.C), we get an instance of case (a.2$_i$.j.R.C) and vice versa. The nodes y_i and y_j are identified, the nodes d_i and d_j are identified, and the second and third clauses for θ show that $\theta(r_{G_N,K_1}(x)) = r_{G_N,K_2}(x)$ as required.

When we interchange the order in case (a.2$_i$.i.R.I), we get an instance of case (a.2$_i$.j.R.I) and vice versa. The nodes y_i and y_j are identified, and the nodes z_i and z_j are identified, for the latter of which we need to make essential use of hypothesis (D). Then the first clause for θ shows that $\theta(r_{G_N,K_1}(x)) = r_{G_N,K_2}(x)$ as required.

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 $\theta(i_{G_N,K_1}(p)) = i_{G_N,K_2}(p)$ by stage 2, and $\theta(r_{G_N,K_1}(c)) = r_{G_N,K_2}(c)$ by stage 3, so

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

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

For case (b$_i$) arcs there are two subcases: (b$_i$.C) where the head is an instantiation of a contractum node; (b$_i$.I) where the head 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

$$\theta((i_{H_1,K_1}(g_1'''(p))_k, r_{H_1,K_1}(g_1'''(c)))) = (k_2(p)_k, k_2(c))$$

if $D_i = D_1$, and

$$\theta((k_1(p)_k, k_1(c))) = (i_{H_2,K_2}(g_2'''(p))_k, r_{H_2,K_2}(g_2'''(c)))$$

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 $h_j' : P_i \rightarrow H_j'$ guarantees that H_j' has a copy, $h_j'((p_k, c))$ 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 $i_{G_{\mathrm{N}i}',H_i}(g_i'(p)) = g_i'''(p)$ in the first case, and $i_{H_i',K_i}(h_i'(p)) = k_i(p)$ 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 $r_{G_{\mathrm{N}},H_j'}(x) = h_j'(c)$ in the second case, given that we have clause (1) 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 H_j'), and symmetry, allows us to conclude that

$$\theta((i_{G_{\mathrm{N}1}',K_1}(g_1'(p))_k, r_{G_{\mathrm{N}1}',K_1}(g_1'(c)))) = (i_{H_2',K_2}(h_2'(p))_k, r_{H_2',K_2}(h_2'(c)))$$

if $D_i = D_1$, and

$$\theta((i_{H_1',K_1}(h_1'(p))_k, r_{H_1',K_1}(h_1'(c)))) = (i_{G_{\mathrm{N}2}',K_2}(g_2'(p))_k, r_{G_{\mathrm{N}2}',K_2}(g_2'(c)))$$

otherwise.

At this point θ is a graph structure isomorphism.

Stage 5. Finally we turn our attention to the markings. This is easy given our assumptions about the ActNodes sets, since by (C), for any $x \in (\mathrm{ActNodes}_i \cup \mathrm{ActNodes}_j)$, $r_{G_{\mathrm{N}},K_i}(x) = i_{G_{\mathrm{N}},K_i}(x)$.

There are eight disjoint cases: (a$_i$) the root f_i; (b) nodes in $(\mathrm{ActNodes}_i \cap \mathrm{ActNodes}_j)$; (c$_i$) nodes in $(\mathrm{ActNodes}_i - \mathrm{ActNodes}_j)$; (d) nodes of G_{N} not in $(\mathrm{ActNodes}_i \cup \mathrm{ActNodes}_j)$; (e$_i$) instantiations of contractum nodes of D_i.

For case (a$_i$), since $f_i \notin \mathrm{ActNodes}_i$, f_i is quiesced in the D_i rewrite. Since $f_i \notin \mathrm{ActNodes}_j \cup \mathrm{MapNodes}_j$ it is unaffected by the D_j rewrite. This holds regardless of order of rewriting, so $\mu(i_{G_{\mathrm{N}},K_1}(f_i)) = \mu(i_{G_{\mathrm{N}},K_2}(f_i)) = \varepsilon$.

For case (b), a node x in $(\mathrm{ActNodes}_i \cap \mathrm{ActNodes}_j)$ which is idle in G_{N}, is activated in the first rewrite, and remains active through the second rewrite regardless of order. If x is non-idle in G_{N}, the activations do not affect it. So $\mu(i_{G_{\mathrm{N}},K_1}(x)) = \mu(i_{G_{\mathrm{N}},K_2}(x))$.

For case (c$_i$), a node x in $(\mathrm{ActNodes}_i - \mathrm{ActNodes}_j)$ which is idle in G_{N}, is activated in the D_i rewrite and is unaffected by the D_j rewrite, again regardless of order. If x is non-idle in G_{N}, the D_i activation does not affect it. So $\mu(i_{G_{\mathrm{N}},K_1}(x)) = \mu(i_{G_{\mathrm{N}},K_2}(x))$.

For case (d), a node of G_{N} not in ($\mathrm{ActNodes}_i \cup \mathrm{ActNodes}_j$) retains the marking it has in G_{N}, regardless of order. So $\mu(i_{G_{\mathrm{N}},K_1}(x)) = \mu(i_{G_{\mathrm{N}},K_2}(x))$.

For case (e_i), since activated nodes are always left subpattern nodes, the instantiations of contractum nodes are not activated in the first rewrite, regardless of order. By (C), $(\mathrm{ActNodes}_i \cup \mathrm{ActNodes}_j) \cap \{f_i, f_j, v_i, v_j\} = \varnothing$, so the redirections of the first rewrite cannot redirect a node to the instantiation of a contractum node (of the first rewrite), that in the second rewrite, becomes part of the redex and is to be activated. So instantiations of contractum nodes of the first rewrite are not activated in the second rewrite, for either order. Hence

$$\mu(i_{H_1,K_1}(g_1'''(p_1))) = \mu(k_2(p_1)) \quad \text{for} \quad p_1 \in N_{P_1} - N_{L_1}$$
$$\mu(k_1(p_2)) = \mu(i_{H_2,K_2}(g_2'''(p_2))) \quad \text{for} \quad p_2 \in N_{P_2} - N_{L_2}$$

This completes the argument for nodes. The argument for arcs is trivial since no arc markings are changed in a rewrite. We are done. ☺

As mentioned above, some easy extensions to the preceding theorem may be imagined. For instance, instead of (A), we could allow also the trivial redirection of s_i to itself. Or we could allow more complex overlaps between the sets of activated nodes and sets of matched nodes in the two redexes. Where activations make the MapNodes of the redex of the following rewrite non-idle, additional notification steps may be introduced to preclude a suspension in certain cases. Where the overlap of matched nodes includes the stateholders non-trivially (as for instance when a stateholder of D_i is in $\mathrm{MapNodes}_j$ with D_j a default rule), restrictions have to be imposed on the redirections of the stateholders. In general, the interaction between the matched, activated, redirected, and notified nodes within two arbitrary rewrites becomes extremely complex, and the extensions proposed make any resulting “all purpose” theorem and its proof an exceptionally long-winded listing of cases, even by the standards of what we did prove. Given the nature of typical “useful and understandable” MONSTR systems, the theorem above is sufficient for most practical purposes. (Note for instance that trivial redirections serve no useful purpose; neither do activations of constructor MapNodes arguments, since these can neither “do” anything themselves, nor notify anyone else.) Where a more powerful interchange theorem might be needed in a specific situation, it is best established on a case by case basis, by adding to the preceding case analysis.

Theorem 6.7 Let $\mathcal{G}_{\mathrm{N}} = [G_0, G_1, \ldots, G_{\mathrm{N}}]$ be a prefix of a suspending execution of a MONSTR system $\mathcal{R}$. Suppose the hypotheses of one of theorems 6.1 – 6.6 apply. Then with the notation used in that theorem (or the obviously analogous notation), $\mathcal{K}_1 = [G_0, G_1, \ldots, G_{\mathrm{N}} \ldots K_1]$ is a prefix of a suspending execution iff $\mathcal{K}_2 = [G_0, G_1, \ldots, G_{\mathrm{N}} \ldots K_2]$ is a prefix of a suspending execution.

Proof. Beyond the facts established already in the theorems mentioned, all we need to check, is that for the execution steps discussed in each particular case, conformance to the execution strategy of suspending semantics in definition 3.1 holds for one order of execution steps iff it holds for the other. But it is relatively obvious that this is so. ☺

There are three points that deserve to be further discussed now. The first, mentioned previously, concerns the style of theorems 6.1 – 6.6. All of them concern a graph in which two potential actions are available, and say that regardless of which is done first, the other is still available afterwards (in a suitable, perhaps even trivial, form). The alternative approach is to postulate that an execution prefix ends with two actions satisfy-ing certain properties, and derive from this, that the order of the steps may be interchanged. In the literature, in [Ehrig (1979)], [Ehrig (1986)] on the one hand, the former is called parallel independence of steps while the latter is called sequential independence; on the other hand, in [Lynch et al. (1994)], the former is called forward commutativity of steps while the latter is called backward commutativity.

For most graph rewriting frameworks, where the structure of the nodes of graphs and the operations on them permitted in a single execution step are relatively simple, the two concepts are equivalent in the sense that a parallel independent pair corresponds to a sequential independent pair, and vice versa; and there is a body of theorems for each such framework establishing the fact. The same is true for suspending semantics up to a point: in some of the cases there is more to it than just swapping the execution steps in question, as the reader will suspect by the nature of some of the preceding theorems.

Below we will list the corresponding sequential independence theorems in a rather abbreviated form, saying just enough to enable us to use the already established parallel independence theorems to complete the interchange of steps. Enthusiastic readers will have no trouble in filling in the details that state the sequential independence theorems as completely standalone results, and the theorems that proclaim how the standalone sequential independence theorems are equivalent to the parallel independence theorems.

A second and related point concerns the closeness of the redexes involved in an interchange theorem. Many graph rewriting frameworks have simpler descriptions than MONSTR, and it is often correspondingly easier to draw up the appropriate independence theorems. It is relatively obvious that if the redexes for two execution steps are "far enough apart" then the two steps can be interchanged. However, for useful MONSTR systems, the redexes of logically independent steps must frequently overlap, and so theorems that merely deal with the independence of steps having disjoint redexes are insufficient. For pairs of execution steps dealt with in theorems 6.1 – 6.5 above, the boundary between steps that are independent can be drawn precisely and fairly easily. For the independence of rewrite steps though, we experience an almost palpable physical force between redexes. The closer we try to push two redexes together, the more excruciating the case analysis of an independence proof becomes. As a result, we contented ourselves with a suboptimal result, which nevertheless is still sufficient for the majority of real applications, since these must not only embody a solution to the problem at hand, but also be relatively comprehensible to the humans who design and use them. The latter aspect limits in practice the degree to which independent redexes actually overlap.

As a final point, we mention that independence theorems provide a route to various formulations of abstract semantics for graph rewriting systems via event structures. (See eg. various papers in [Schneider and Ehrig (1993)], [Cuny et al. (1996)], [Corradini and Montanari (1995)].) We avoid doing the same in this study for two reasons. On the one hand, we are primarily interested in formulating what equivalences we can between different semantic models for MONSTR, and to do this it is sufficient to derive relationships between different concrete executions, without worrying about the prospects for collecting up sets of executions into equivalence classes in a convincing manner. On the other hand, it is clear from theorems 6.3 and 6.4, and from considering simple examples of overlapping redexes for rewrites, that suspending semantics can manifest a number of situations where the notion of conflict between events is more complex than the elegant notion of symmetric conflict to be found in conventional treatments of event structures ([Nielsen et al. (1981)], [Winskel (1986)], [Winskel (1988)]). It is probably fair to say that a completely satisfactory treatment of asymmetric conflict in event structures has not yet appeared.

Here are the abbreviated sequential independence theorems. Note that the asymmetry of the situations involving different types of execution step entails the existence of two different theorems whereas one sufficed in the more symmetric case of parallel independence. Note also the additional hypotheses required to exclude various cases in which the second execution step might conceivably have been caused by the first.

Theorem 6.8 Let $\mathcal{K}_1 = [G_0, G_1, \ldots, G_N, H_1, K_1]$ be a prefix of a suspending execution of a MONSTR system $\mathcal{R}$. Suppose the last two execution steps of $\mathcal{K}_1$ are notifications, from $t_1 \in G_N$ and $t^*_2 \in H_1$ respectively. Then there is a node $t_2 \in G_N$ such that $r_{G_N,H_1}(t_2) = t^*_2$ and such that the hypotheses of theorem 6.1 apply to t_1, t_2 in G_N. Consequently the two notifications may be interchanged, yielding marking preserving isomorphic final graphs for the two resulting execution prefixes.

Theorem 6.9 Let $\mathcal{K}_1 = [G_0, G_1, \ldots, G_N, H_1, K_1]$ be a prefix of a suspending execution of a MONSTR system $\mathcal{R}$. Suppose the last two execution steps of $\mathcal{K}_1$ are suspensions, from $s_1 \in G_N$ and $s^*_2 \in H_1$ respectively. Then there is a node $s_2 \in G_N$ such that $r_{G_N,H_1}(s_2) = s^*_2$ and such that the hypotheses of theorem 6.2 apply to s_1, s_2 in G_N. Consequently the two suspensions may be interchanged, yielding marking preserving isomorphic final graphs for the two resulting execution prefixes.

Theorem 6.10a Let $\mathcal{K}_s = [G_0, G_1, \ldots, G_N, H_s, K_s]$ be a prefix of a suspending execution of a MONSTR system $\mathcal{R}$. Suppose the last two execution steps of $\mathcal{K}_s$ are a suspension, from $s \in G_N$ and a notification from $t^* \in H_s$ respectively. Then there is a node $t \in G_N$ such that $r_{G_N,H_s}(t) = t^*$ and such that the hypotheses of theorem 6.3 apply to s, t in G_N. Consequently the suspension and notification may be interchanged, yielding $\mathcal{K}_t$, which equals $[G_0, G_1, \ldots, G_N, H_t, K_t]$ if (in the notation of theorem 6.3), SuspNodes $\neq \{t\}$, and which equals $[G_0, G_1, \ldots, G_N, K_t]$, omitting the suspension, otherwise; where in either case, K_s and K_t are marking preserving isomorphic.

Theorem 6.10b Let $\mathcal{K}_t = [G_0, G_1, \ldots, G_N, H_t, K_t]$ be a prefix of a suspending execution of a MONSTR system $\mathcal{R}$. Suppose the last two execution steps of $\mathcal{K}_t$ are a notification, from $t \in G_N$ and a suspension from $s^* \in H_t$ respectively. Then there is a node $s \in G_N$ such that $r_{G_N,H_t}(s) = s^*$ and such that the hypotheses of theorem 6.3 apply to s, t in G_N. Consequently the notification and suspension may be interchanged, yielding $\mathcal{K}_s$, which equals $[G_0, G_1, \ldots, G_N, H_s, K_s]$ where K_s and K_t are marking preserving isomorphic. A similar execution prefix may be generated if $\mathcal{K}_t = [G_0, G_1, \ldots, G_N, K_t]$, with the last step a notification from a node $t \in G_N$ which is the only non-idle Map($\sigma(s)$) child of a function node $s \in G_N$; and again K_s and K_t are marking preserving isomorphic.

Theorem 6.11a Let $\mathcal{K}_f = [G_0, G_1, \ldots, G_N, H_f, K_f]$ be a prefix of a suspending execution of a MONSTR system $\mathcal{R}$. Suppose the last two execution steps of $\mathcal{K}_f$ are a rewrite from $f \in G_N$ governed by a rule $D = (P, root, Red, Act)$, and a notification from $t^* \in H_f$ respectively. Suppose $t^* \in H_f$ is not the instantiation of a contractum node of P. Then there is a node $t \in G_N$ such that $r_{G_N,H_f}(t) = t^*$. Suppose $t \in G_N$ is not idle. Then the hypotheses of theorem 6.4 apply to f, t in G_N. Consequently the rewrite and notification may be interchanged, yielding $\mathcal{K}_t$, which equals $[G_0, G_1, \ldots, G_N, H_t, K_t]$ if (in the notation of theorem 6.4), $t \notin g(Act)$, and which equals $[G_0, G_1, \ldots, G_N, H_t, J_t, K_t]$, including an additional notification from $i_{G_N,J_t}(t)$ in J_t, otherwise; where in either case, K_f and K_t are marking preserving isomorphic.

Theorem 6.11b Let $\mathcal{K}_t = [G_0, G_1, \ldots, G_N, H_t, K_t]$ be a prefix of a suspending execution of a MONSTR system $\mathcal{R}$. Suppose the last two execution steps of $\mathcal{K}_t$ are a notification from $t \in G_N$, and a rewrite from $f^* \in H_t$ governed by a rule $D = (P, root, Red, Act)$ respectively. Then there is a node $f \in G_N$ such that $r_{G_N,H_t}(f) = f^*$. Suppose that t is not a Map($\sigma(f)$) argument of $f \in G_N$. Then the hypotheses of theorem 6.4 apply to f, t in G_N. Consequently the notification and rewrite may be interchanged, yielding $\mathcal{K}_f$, which equals $[G_0, G_1, \ldots, G_N, H_f, K_f]$ if (in the notation of theorem 6.4), $t \notin g(Act)$; where K_f and K_t are marking preserving isomorphic. A similar execution prefix may be generated if $\mathcal{K}_t = [G_0, G_1, \ldots, G_N, H_t, J_t, K_t]$, with the last step an additional notification from $r_{G_N,J_t}(t)$ in J_t in case (in the notation of theorem 6.4), $t \in g(Act)$; and again K_f and K_t are marking preserving isomorphic.

Theorem 6.12a Let $\mathcal{K}_f = [G_0, G_1, \ldots, G_N, H_f, K_f]$ be a prefix of a suspending execution of a MONSTR system $\mathcal{R}$. Suppose the last two execution steps of $\mathcal{K}_s$ are a rewrite from $f \in G_N$ governed by a rule $D = (P, root, Red, Act)$, and a suspension from $s^* \in H_f$ respectively. Suppose $s^* \in H_f$ is not the instantiation of a contractum node of P. Then there is a node $s \in G_N$ such that $r_{G_N,H_f}(s) = s^*$. Suppose $s \in G_N$ is not idle. Suppose the hypotheses of theorem 6.5 apply to s, f in G_N; in particular suppose (in the notation of theorem 6.5), that [$f \notin$ SuspNodes or $\mu(b) \neq \varepsilon$ or $b \in Act$], and that ActNodes $\cap$ $\overline{\text{SuspNodes}} = \varnothing$. Consequently the rewrite and suspension may be interchanged, yielding $\mathcal{K}_s = [G_0, G_1, \ldots, G_N, H_s, K_s]$, where K_f and K_s are marking preserving isomorphic.

Theorem 6.12b Let $\mathcal{K}_s = [G_0, G_1, \ldots, G_N, H_s, K_s]$ be a prefix of a suspending execution of a MONSTR system $\mathcal{R}$. Suppose the last two execution steps of $\mathcal{K}_s$ are a suspension, from $s \in G_N$, and a rewrite from $f^* \in H_s$ governed by a rule $D = (P, root, Red, Act)$ respectively. Then there is a node $f \in G_N$ such that $r_{G_N,H_s}(f) = f^*$. Suppose the hypotheses of theorem 6.5 apply to s, f in G_N; in particular suppose (in the notation of theorem 6.5), that [$f \notin$ SuspNodes or $\mu(b) \neq \varepsilon$ or $b \in Act$], and that ActNodes $\cap$ $\overline{\text{SuspNodes}} = \varnothing$. Consequently the suspension and rewrite may be interchanged, yielding $\mathcal{K}_f = [G_0, G_1, \ldots, G_N, H_f, K_f]$, where K_f and K_s are marking preserving isomorphic.

Theorem 6.13 Let $\mathcal{K}_1 = [G_0, G_1, \ldots, G_N, H_1, K_1]$ be a prefix of a suspending execution of a MONSTR system $\mathcal{R}$. Suppose the last two execution steps of $\mathcal{K}_1$ are rewrites, from $f_1 \in G_N$ and $f^*_2 \in H_1$, governed by rules $D_i = (P_i, root_i, Red_i, Act_i)$ for $i = 1$ and 2 respectively. Suppose $f^*_2 \in H_1$ is not the instantiation of a contractum node of P_1. Then there is a node $f_2 \in G_N$ such that $r_{G_N,H_1}(f_2) = f^*_2$. Suppose $f_2 \in G_N$ is not idle. Suppose the hypotheses of theorem 6.6 apply to f_1, f_2 in G_N; in particular suppose (in the notation of theorem 6.6), that

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

(B) For either $i \in \{1, 2\}$, D_i redirects $s_i \;\Rightarrow\;$ [$v_i \notin \text{MapNodes}_j$ and $v_j \notin \text{MapNodes}_i$]

(C) $(\text{ActNodes}_i \cup \text{ActNodes}_j) \cap (\{f_i, f_j\} \cup \text{MapNodes}_i \cup \text{MapNodes}_j) = \varnothing$

(D) For both $i \in \{1, 2\}$,

$$\begin{aligned}\varnothing = \{(x, y, z) \in {G_N}^3 \mid\ & x \in \text{RedNodes}_i \text{ and } y \in \text{RedNodes}_j \text{ and } z \in \text{RedNodes}_i \\ & \text{and } x \neq z \text{ and } (x, y) = (g_i(a), g_i(b)) \text{ for } (a, b) \in Red_i \\ & \text{and } (y, z) = (g_j(c), g_j(d)) \text{ for } (c, d) \in Red_j\}\end{aligned}$$

Consequently the two rewrites may be interchanged, yielding marking preserving isomorphic final graphs for the two resulting execution prefixes.

7 CONCLUSIONS

In the previous sections we have defined suspending semantics for the MONSTR graph rewriting language. We studied notions of correctness for suspending executions of a MONSTR system R in terms of their producing graphs that were in one sense or another equivalent to graphs produced by DACTL executions of the same system. At best, marking preserving isomorphism is the equivalence of choice, and one that fortunately, gives us a powerful handle on correctness via properties of the EDG, especially for terminating executions. In the process of exploring correctness, we were able to describe conditions sufficient for correctness, which reduced the correctness problem to the solution of a number of subproblems.

For terminating executions three subproblems sufficed:

- The acyclicity of the EDG.
- The absence of idle stateholder children of notification arcs in the final graph.
- Termination.

For non-terminating executions we had:

- Nodes are not simultaneously both the LHS and RHS of redirections indefinitely.
- Node event chains are eventually idle.

Readers will note that the subproblems listed above all involve properties of execution graphs, i.e. they involve dynamic, or runtime properties of systems. Typically such properties are undecidable; so to be able to state that there are systems whose suspending executions are correct for the kinds of reason that we have explored, we must look for decidable approximations. Future papers will examine the stated subproblems in depth, and in turn, approximate them by criteria decidable on the basis of a static analysis of rule systems.

A further important issue we examined was independence. We saw that the introduction of suspensions complicated the picture quite a bit; some of the independence results required complexities such as the introduction of extra steps, or the elimination of one of the original steps. In the face of this, it was safer in the short term, to restrict our attention to concrete independence results rather than try to construct an abstract event semantics for suspending systems. The latter is certainly a fascinating problem, given the “closeness to reality” of suspending execution steps. They certainly seem to display much of the contrariness of events in the real world, and this feature may be helpful in the construction of a convincing theory of asymmetric conflict for them. But it also seems likely that any attempt at such a theory might well be controversial to a greater or lesser extent, so it is best left to another place. In the same vein, one can speculate that even more convoluted combinations of execution steps may be proved interchangable (up to isomorphism of the graph produced), than we contemplated above. With regard to abstract event semantics, such possibilities forcefully raise questions about the very notion of what constitutes an event, and confirm that it is very much a “context sensitive decision”, depending on what the creator of a particular theory hopes to achieve. Again such matters are best avoided in this paper.

References

[Banach (1996)] Banach. R., MONSTR I — Fundamental Issues and the Design of MONSTR. J.UCS, **2**, 164-216, (1996).

[Banach et al. (1995)] Banach R., Balazs J., Papadopoulos G., A Translation of the Pi-Calculus into MONSTR. J.UCS **1**, 335-394, (1995).

[Corradini and Montanari (1995)] Corradini A., Montanari U., (eds.) Proc. SEGRAGRA-95. E.N.T.C.S. **2** 9-16, Elsevier, (1995).

[Ehrig (1979)] Ehrig H., Introduction to the Algebraic Theory of Graph Grammars (A survey). *in*: L.N.C.S. **73**, 1-69, Springer, Berlin, (1979).

[Ehrig (1986)] Ehrig H., A Tutorial Introduction to the Algebraic Approach of Graph Grammars. *in*: Third International Workshop on Graph Grammars, L.N.C.S. **291**, 3-14, Springer, Berlin, (1986).

[Francez (1986)] Francez N., Fairness. Springer, Berlin, (1986).

[Lynch et al. (1994)] Lynch N., Merritt M., Weihl W., Fekete A., Atomic Transactions. Morgan Kaufmann, (1994).

[Nielsen et al. (1981)] Nielsen M., Plotkin G., Winskel G., Petri Nets, Event Structures and Domains, Part I. Theoretical Computer Science **13**, 85-108, (1981).

[Cuny et al. (1996)] Cuny J., Ehrig H, Engels G., Rozenberg G. (eds.) Proc. Fifth International Workshop on Graph Grammars and their Application to Computer Science 1994. L.N.C.S. **1073**, Springer, Berlin, (1996).

[Schneider and Ehrig (1993)] Schneider H-J., Ehrig H., (eds.), Graph Transformations in Computer Science. L.N.C.S. **776**, Springer, Berlin, (1993).

[Winskel (1986)] Winskel G., Event Structures. *in*: Petri Nets, An Advanced Course, L.N.C.S. **255**, 325-392, Springer, Berlin, (1986).

[Winskel (1988)] Winskel G., An Introduction to Event Structures. *in*: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. de Bakker, de Roever, Rozenberg (eds.), L.N.C.S. **354**, 364-397, Springer, Berlin, (1988).