Unfolding-based Diagnosis of Systems with an Evolving Topology*

Paolo Baldan[1], Thomas Chatain[2], Stefan Haar[3], and Barbara König[4]

[1] Dipartimento di Matematica Pura e Applicata, Università di Padova, Italy
[2] LSV, ENS Cachan, CNRS, INRIA, France
[3] INRIA, France, and University of Ottawa, Canada
[4] Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen, Germany

Abstract. We propose a framework for model-based diagnosis of systems with mobility and variable topologies, modelled as graph transformation systems. Generally speaking, model-based diagnosis is aimed at constructing explanations of observed faulty behaviours on the basis of a given model of the system. Since the number of possible explanations may be huge we exploit the unfolding as a compact data structure to store them, along the lines of previous work dealing with Petri net models. Given a model of a system and an observation, the explanations can be constructed by unfolding the model constrained by the observation, and then removing incomplete explanations in a pruning phase. The theory is formalised in a general categorical setting: constraining the system by the observation corresponds to taking a product in the chosen category of graph grammars, so that the correctness of the procedure can be proved by using the fact that the unfolding is a right adjoint and thus it preserves products. The theory thus should be easily applicable to a wide class of system models, including graph grammars and Petri nets.

1 Introduction

The *event-oriented model-based diagnosis* problem is a classical topic in discrete event systems [7, 15]. Given an observed alarm stream, the aim is to provide *explanations* in terms of actual system behaviours. Some events of the system are observable (alarms) while others are not. In particular, fault events are usually unobservable; therefore, fault diagnosis is the main motivation of the diagnosis problem. Given a sequence (or partially ordered set) of observable events, the diagnoser has to find all possible behaviours of the model explaining the observation, thus allowing the deduction of invisible causes (faults) of visible events (alarms). The paper [16] provides a survey on fault diagnosis in this direction.

Since the number of possible explanations may be huge, especially in the case of highly concurrent systems, it is advisable to employ space-saving methods. In [16, 10], the global diagnosis is obtained as the fusion of local decisions: this

* Supported by the MIUR Project ART, RNRT project SWAN, INRIA Sabbatical program, the DFG project SANDS and CRUI/DAAD Vigoni.

distributed approach allows one to factor explanations over a set of local observers and diagnoses, rather than centralizing the data storage and handling.

We will build here upon the approach of [5] where diagnoses are stored in the form of unfoldings. The unfolding of a system fully describes its concurrent behaviour in a single branching structure, representing all the possible computation steps and their mutual dependencies, as well as all reachable states; the effectiveness of the approach lies in the use of partially ordered runs, rather than interleavings, to store and handle explanations extracted from the system model.

While [5] and subsequent work in this direction was mainly directed to Petri nets, here we face the diagnosis problem in mobile and variable topologies. This requires the development of a model-based diagnosis approach which applies to other, more expressive, formalisms. Unfoldings of extensions of Petri nets where the topology may change dynamically were studied in [8, 6]. Here we focus on the general and well-established formalism of graph transformation systems.

In order to retain only the behaviour of the system that matches the observations, it is not the model itself that is unfolded, but the product of the model with the observations, which represents the original system constrained by the observation; under suitable observability assumptions, a finite prefix of the unfolding is sufficient. The construction is carried out in a suitably defined category of graph grammars, where such a product can be shown to be the categorical product. A further *pruning* phase is necessary in order to remove incomplete explanations that are only valid for a prefix of the observations.

We show the correctness of this technique, i.e., we show that the runs of the unfolding properly capture all those runs of the model which explain the observation. This non-trivial result is obtained by using the fact that unfolding for graph grammars is a coreflection, hence it preserves limits (and especially products, such as the product of the model and the observation). In order to ensure that the product is really a categorical product, special care has to be taken in the definition of the category.

Additional technical details and the proofs of all the results can be found in the full version of the paper [1].

2 Graph Grammars and Grammar Morphisms

In this section we summarise the basics of graph rewriting in the *single-pushout* (SPO) approach [13]. We introduce a category of graph grammars, whose morphisms are a variation of those in [3] and we characterise the induced categorical product, which turns out to be adequate for expressing the notion of composition needed in our diagnosis framework. Then we show that the unfolding semantics smoothly extends to this setting, arguing that the unfolding construction can still be characterised categorically as a universal construction. The proof relies on the results in [3]; this motivates our choice of the SPO approach as opposed to the more classical *double-pushout* (DPO) approach, for graph rewriting.

2.1 Graph Grammars and their Morphisms

Given a partial function $f : A \rightharpoonup B$ we write $f(a)\downarrow$ whenever f is defined on $a \in A$ and $f(a)\uparrow$ whenever it is undefined. We will denote by $dom(f)$ the *domain* of f, i.e., the set $\{a \in A \mid f(a)\downarrow\}$. Let $f, g : A \rightharpoonup B$ be two partial functions. We will write $f \leq g$ when $dom(f) \subseteq dom(g)$ and $f(x) = g(x)$ for all $x \in dom(f)$.

For a set A, we denote by A^* the set of sequences over A. Given $f : A \rightharpoonup B$, the symbol $f^* : A^* \to B^*$ denotes its extension to sequences defined by $f^*(a_1 \dots a_n) = f(a_1) \dots f(a_n)$, where it is intended that the elements on which f is undefined are "forgotten". Specifically, $f^*(a_1 \dots a_n) = \varepsilon$ whenever $f(a_i) \uparrow$ for any $i \in \{1, \dots, n\}$. Instead, $f^\perp : A^* \rightharpoonup B^*$ denotes the *strict* extension of f to sequences, satisfying $f^\perp(a_1 \dots a_n) \uparrow$ whenever $f(a_i) \uparrow$ for some $i \in \{1, \dots, n\}$.

A *(hyper)graph* G is a tuple (N_G, E_G, c_G), where N_G is a set of nodes, E_G is a set of edges and $c_G : E_G \to N_G^*$ is a connection function. Given a graph G we will write $x \in G$ to say that x is a node or edge in G, i.e., $x \in N_G \cup E_G$.

Definition 1 (partial graph morphism). *A* partial graph morphism $f : G \rightharpoonup H$ *is a pair of partial functions* $f = \langle f_N : N_G \rightharpoonup N_H, f_E : E_G \rightharpoonup E_H \rangle$ *such that:*

$$c_H \circ f_E \leq f_N^\perp \circ c_G \ (*)$$

We denote by **PGraph** *the category of hypergraphs and partial graph morphisms. A morphism is called* total *if both components are total, and the corresponding subcategory of* **PGraph** *is denoted by* **Graph**.

Notice that, according to condition (*), if f is defined on an edge then it must be defined on all its adjacent nodes: this ensures that the domain of f is a well-formed graph. The inequality in condition (*) ensures that *any* subgraph of a graph G can be the domain of a partial morphism $f : G \rightharpoonup H$.

We will work with *typed graphs* [9, 14], which are graphs labelled over a structure that is itself a graph, called the *graph of types*.

Definition 2 (typed graph). *Given a graph T, a* typed graph G *over T is a graph $|G|$, together with a total morphism $t_G : |G| \to T$. A* partial morphism *between T-typed graphs $f : G_1 \rightharpoonup G_2$ is a partial graph morphism $f : |G_1| \rightharpoonup |G_2|$ consistent with the typing, i.e., such that $t_{G_1} \geq t_{G_2} \circ f$. A typed graph G is called* injective *if the typing morphism t_G is injective. The category of T-typed graphs and partial typed graph morphisms is denoted by T-***PGraph**.

Definition 3 (graph production, direct derivation). *Fixing a graph T of types, a* (T-typed graph) production q *is an injective partial typed graph morphism $L_q \overset{r_q}{\rightharpoonup} R_q$. It is called* consuming *if r_q is not total. The typed graphs L_q and R_q are called* left-hand side *and* right-hand side *of the production.*

Given a typed graph G and a match*, i.e., a total injective morphism $g : L_q \to G$, we say that there is a* direct derivation δ from G to H using q (based on g)*, written $\delta : G \Rightarrow_q H$, if there is a pushout square in T-***PGraph** *as on the right.*

$$\begin{array}{ccc} L_q & \xrightarrow{r_q} & R_q \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle h} \\ G & \xrightarrow[d]{} & H \end{array}$$

Roughly speaking, the rewriting step removes from G the image of the items of the left-hand side which are not in the domain of r_q, namely $g(L_q - dom(r_q))$,

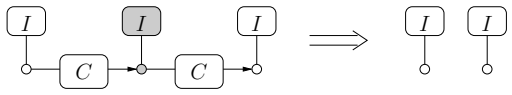

Fig. 1. Dangling edge removal in SPO rewriting.

adding the items of the right-hand side which are not in the image of r_q, namely $R_q - r_q(dom(r_q))$. The items in the image of $dom(r_q)$ are "preserved" by the rewriting step (intuitively, they are accessed in a "read-only" manner). Additionally, whenever a node is removed, all the edges incident to such a node are removed as well. For instance, consider production **fail** at the bottom of Fig. 2. Its left-hand side contains a unary edge (i.e., an edge connected to only one node) and its right-hand side is the empty graph. Nodes and edges are represented as circles and boxes, respectively. The application of **fail** to a graph is illustrated in Fig. 1, where the match of the left-hand side is indicated as shaded.

Definition 4 (typed graph grammar). *A* (T-typed) SPO graph grammar $\mathcal{G}$ *is a tuple* $\langle T, G_s, P, \pi, \Lambda, \lambda \rangle$*, where* G_s *is the* (typed) start graph*,* P *is a set of* production names*,* π *is a function which associates to each name* $q \in P$ *a production* $\pi(q)$*, and* $\lambda : P \to \Lambda$ *is a labelling over the set* Λ*. A graph grammar is* consuming *if all the productions in the range of* π *are consuming.*

As standard in unfolding approaches, in the paper we will consider *consuming* graph grammars only, where each production deletes some item. Hereafter, when omitted, we will assume that the components of a given graph grammar $\mathcal{G}$ are $\langle T, G_s, P, \pi, \Lambda, \lambda \rangle$. Subscripts carry over to the component names.

For a graph grammar $\mathcal{G}$ we denote by $Elem(\mathcal{G})$ the set $N_T \cup E_T \cup P$. As a convention, for each production name q the corresponding production $\pi(q)$ will be $L_q \stackrel{r_q}{\rightharpoonup} R_q$. Without loss of generality, we will assume that the injective partial morphism r_q is a partial inclusion (i.e., that $r_q(x) = x$ whenever defined). Moreover we assume that the domain of r_q, which is a subgraph of both $|L_q|$ and $|R_q|$, is the *intersection* of these two graphs, i.e., that $|L_q| \cap |R_q| = dom(r_q)$, componentwise. Since in this paper we work only with typed notions, we will usually omit the qualification "typed", and, sometimes, we will not indicate explicitly the typing morphisms.

In the sequel we will often refer to the runs of a grammar defined as follows.

Definition 5 (runs of a grammar). *Let* $\mathcal{G}$ *be a graph grammar. Then* $Runs(\mathcal{G})$ *contains all sequences* $r_1 r_2 \ldots r_n$ *where* $r_i \in P$ *and* $G_s \stackrel{r_1}{\Rightarrow} G_1 \stackrel{r_2}{\Rightarrow} G_2 \ldots \stackrel{r_n}{\Rightarrow} G_n$*.*

Example. As a running example we will consider the graph grammar $\mathcal{M}$ whose start graph and productions are given in Fig. 2. It models a network with mobility whose nodes are either senders (labelled S), receivers (R) or intermediary nodes (I). Senders may send messages which can then cross connections and should finally arrive at a receiver. However, a connection may be spontaneously

corrupted, which causes the corruption of any message which crosses it. The network is variable and of unbounded size as we allow the creation of a new connection between existing intermediary nodes and the creation of a new connection leading from an existing intermediary node to a new intermediary node.

Productions (and the corresponding partial morphisms) are depicted as follows: edges that are deleted or created are drawn with solid lines, whereas edges that are preserved are indicated with dashed lines. Nodes which are preserved are indicated with numbers, whereas newly created nodes are not numbered.

Productions that should be observable (a notion that will be made formal in Section 4) are indicated by bold face letters.

We next define the class of grammars which will focus on in the development.

Definition 6 (semi-weighted SPO graph grammars). *A grammar $\mathcal{G}$ is* semi-weighted *if (i) the start graph G_s is injective, and (ii) for each $q \in P$, for any x, y in $|R_q| - |L_q|$ if $t_{R_q}(x) = t_{R_q}(y)$ then $x = y$, i.e., the right-hand side graph R_q is injective on the "produced part" $|R_q| - |L_q|$.*

Intuitively, conditions (i) and (ii) ensure that in a semi-weighted grammar each item generated in a computation has a uniquely determined causal history, a fact which is essential for the validity of Theorem 15.

Note that grammar $\mathcal{M}$ of Fig. 2 is not semi-weighted (if we assume the simplest type graph that contains one node and exactly one edge for every edge label). It could easily be converted into a semi-weighted grammar, for instance by creating the start graph (which is not injectively typed) step by step. However, for the sake of simplicity we do not carry out this construction in the paper.

A grammar morphism consists of a (partial) mapping between production names and a component specifying the (multi)relation between the type graphs. A morphism must "preserve" the graphs underlying corresponding productions of the two grammars as well as the start graphs. Since these conditions are exactly the same as in [3] and they are not relevant for understanding this paper, in the sequel we will refer to the morphisms in [3], making explicit only the new condition regarding the labelling. The interested reader can find the details in the full version [1].

Definition 7 (grammar morphism). *Let $\mathcal{G}_i$ ($i \in \{1, 2\}$) be graph grammars such that $\Lambda_2 \subseteq \Lambda_1$. A* grammar morphism *$f : \mathcal{G}_1 \to \mathcal{G}_2$ is a morphism in the sense of [3, Def. 15] where the component on productions, i.e., the partial function $f_P : P_1 \rightharpoonup P_2$, additionally satisfies, for all $q_1 \in P_1$*

$$f_P(q_1) \downarrow \quad \textit{iff} \quad \lambda_1(q_1) \in \Lambda_2 \textit{ and, in this case, } \lambda_2(f_P(q_1)) = \lambda_1(q_1).$$

Note that a morphism from $\mathcal{G}_1$ to $\mathcal{G}_2$ might exist only when $\Lambda_2 \subseteq \Lambda_1$.

Definition 8 (category of graph grammars). *We denote by* **GG** *the category where objects are* SPO *graph grammars and arrows are grammar morphisms. By* **SGG** *we denote the full subcategory of* **GG** *having semi-weighted graph grammars as objects.*

The choice of grammar morphisms and, in particular, the conditions on the labelling, lead to a categorical product suited for composing two grammars $\mathcal{G}_1$ and

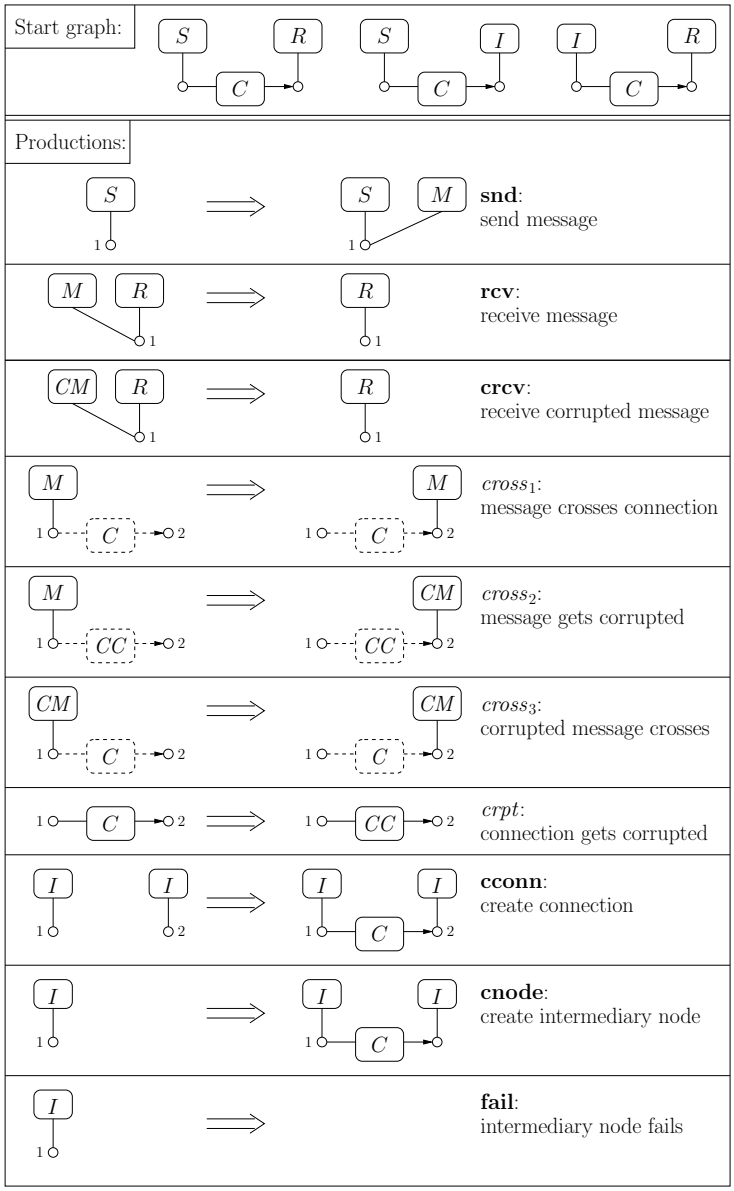

Fig. 2. Example grammar $\mathcal{M}$: message passing over possibly corrupted connections.

$\mathcal{G}_2$: productions with labels in $\Lambda_1 \cap \Lambda_2$ are forced to be executed in a synchronous way, while the others are executed independently in the two components.

Proposition 9 (product of graph grammars). *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two graph grammars. Their product object $\mathcal{G} = \mathcal{G}_1 \times \mathcal{G}_2$ in* **GG** *is defined as follows:*

- $T = T_1 \uplus T_2$;
- $G_s = G_{s1} \uplus G_{s2}$, *with the obvious typing;*
- $P = \{(p_1, p_2) \mid \lambda_1(p_1) = \lambda_2(p_2)\} \cup \{(p_1, \emptyset) \mid \lambda_1(p_1) \notin \Lambda_2\} \cup \{(\emptyset, p_2) \mid \lambda_2(p_2) \notin \Lambda_1\}$;
- $\pi(p_1, p_2) = \pi_1(p_1) \uplus \pi_2(p_2)$, *where $\pi_i(\emptyset)$ is the empty rule $\emptyset \to \emptyset$;*
- $\Lambda = \Lambda_1 \cup \Lambda_2$;
- $\lambda(p_1, p_2) = \lambda_i(p_i)$, *for any $i \in \{1, 2\}$ such that $p_i \neq \emptyset$;*

where, p_1 and p_2 range over P_1 and P_2, respectively, and disjoint unions are taken componentwise. If $\mathcal{G}_1, \mathcal{G}_2$ are both semi-weighted grammars, then $\mathcal{G}$ as defined above is semi-weighted, and it is the product of $\mathcal{G}_1$ and $\mathcal{G}_2$ in **SGG**.

2.2 Occurrence Grammars and Unfolding

A grammar $\mathcal{G}$ is *safe* if (i) for all H such that $G_s \Rightarrow^* H$, H is injective, and (ii) for each $q \in P$, the left- and right-hand side graphs L_q and R_q are injective.

In words, in a safe grammar each graph G reachable from the start graph is injectively typed, and thus we can identify it with the corresponding subgraph $t_G(|G|)$ of the type graph. With this identification, a production can only be applied to the subgraph of the type graph which is the image via the typing morphism of its left-hand side. Thus, according to its typing, we can think that a production *produces*, *preserves* or *consumes* items of the type graph, and using a net-like language, we speak of pre-set, context and post-set of a production, correspondingly. Intuitively the type graph T stands for the *places* of a net, whereas the productions P represent the *transitions*.

Definition 10 (pre-set, post-set and context of a production). *Let $\mathcal{G}$ be a graph grammar. For any production $q \in P$ we define its* pre-set $^\bullet q$, context $\underline{q}$ *and* post-set $q^\bullet$ *as the following subsets of $E_T \cup N_T$:*

$$^\bullet q = t_{L_q}(|L_q| - |dom(r_q)|) \qquad \underline{q} = t_{L_q}(|dom(r_q)|) \qquad q^\bullet = t_{R_q}(|R_q| - r_q(|dom(r_q)|)).$$

Symmetrically, for each item $x \in T$ we define $^\bullet x = \{q \in P \mid x \in q^\bullet\}$, $x^\bullet = \{q \in P \mid x \in {}^\bullet q\}$, $\underline{x} = \{q \in P \mid x \in \underline{q}\}$.

Causal dependencies between productions are captured as follows.

Definition 11 (causality). *The* causality *relation of a grammar $\mathcal{G}$ is the (least) transitive relation $<$ over $Elem(\mathcal{G})$ satisfying, for any node or edge $x \in T$, and for productions $q, q' \in P$,*

1. *if $x \in {}^\bullet q$ then $x < q$;*
2. *if $x \in q^\bullet$ then $q < x$;*
3. *if $q^\bullet \cap \underline{q'} \neq \emptyset$ then $q < q'$.*

As usual $\leq$ is the reflexive closure of $<$. Moreover, for $x \in Elem(\mathcal{G})$ we denote by $\lfloor x \rfloor$ the set of causes of x in P, namely $\{q \in P \mid q \leq x\}$.

As in Petri nets with read arcs, the fact that a production application not only consumes and produces, but also preserves a part of the state, leads to a form of asymmetric conflict between productions; for a thorough discussion of asymmetric event structures see [2].

Definition 12 (asymmetric conflict). *The* asymmetric conflict relation *of a grammar $\mathcal{G}$ is the binary relation $\nearrow$ over the set of productions, defined by:*

1. *if $q \cap {}^\bullet q' \neq \emptyset$ then $q \nearrow q'$;*
2. *if $\overline{{}^\bullet q} \cap {}^\bullet q' \neq \emptyset$ and $q \neq q'$ then $q \nearrow q'$;*
3. *if $q < q'$ then $q \nearrow q'$.*

Intuitively, whenever $q \nearrow q'$, q can never follow q' in a computation. This holds when q preserves something deleted by q' (Condition 1), trivially when q and q' are in conflict (Condition 2) and also when $q < q'$ (Condition 3). Conflicts (in acyclic grammars) are represented by cycles of asymmetric conflict: if $q_1 \nearrow q_2 \nearrow \ldots \nearrow q_n \nearrow q_1$ then $\{q_1, \ldots, q_n\}$ will never appear in the same computation.

An *occurrence grammar* is an acyclic grammar which represents, in a branching structure, several possible computations beginning from its start graph and using each production at most once. Recall that a relation $R \subseteq X \times X$ is *finitary* if for any $x \in X$, the set $\{y \in X \mid R(y, x)\}$ is finite.

Definition 13 (occurrence grammar). *An* occurrence grammar *is a safe grammar $\mathcal{O} = \langle T, G_s, P, \pi, \Lambda, \lambda \rangle$ such that*

1. *causality $<$ is irreflexive, its reflexive closure $\leq$ is a partial order, and, for any $q \in P$, the set $\lfloor q \rfloor$ is finite and asymmetric conflict $\nearrow$ is acyclic on $\lfloor q \rfloor$;*
2. *the start graph G_s is the set $Min(\mathcal{O})$ of minimal elements of $\langle Elem(\mathcal{O}), \leq \rangle$ (with the graphical structure inherited from T and typed by the inclusion);*
3. *any item x in T is created by at most one production in P, i.e., $|{}^\bullet x| \leq 1$;*

A finite occurrence grammar is deterministic *if relation $\nearrow^+$, the transitive closure of $\nearrow$, is irreflexive. We denote by* **OGG** *the full subcategory of* **GG** *with occurrence grammars as objects.*

Note that the start graph of an occurrence grammar $\mathcal{O}$ is determined by $Min(\mathcal{O})$. An occurrence grammar is deterministic if it does not contain conflicts (cycles of asymmetric conflict) so that all its productions can be executed in the same computation. In the sequel, the productions of an occurrence grammar will often be called events.

The notion of configuration captures the intuitive idea of (deterministic) computation in an occurrence grammar.

Definition 14 (configuration). *Let $\mathcal{O} = \langle T, P, \pi \rangle$ be an occurrence grammar. A* configuration *is a subset $C \subseteq P$ such that (i) for any $q \in C$ it holds $\lfloor q \rfloor \subseteq C$ and (ii) $\nearrow_C$, the asymmetric conflict restricted to C, is acyclic and finitary.*

It can be shown that, indeed, all the productions in a configuration can be applied in a derivation exactly once in any order compatible with $\nearrow$.

Since occurrence grammars are particular semi-weighted grammars, there is an inclusion functor $\mathcal{I} : \mathbf{OGG} \to \mathbf{SGG}$. Such functor has a right adjoint.

Theorem 15. *The inclusion functor* $\mathcal{I} : \mathbf{OGG} \to \mathbf{SGG}$ *has a right adjoint, the so-called unfolding functor* $\mathcal{U} : \mathbf{SGG} \to \mathbf{OGG}$.

As a consequence of the above result $\mathcal{U}$, as a right adjoint, preserves all limits and in particular products.

The result is a corollary of [3], which, in turn, is obtained through the explicit definition of the unfolding $\mathcal{U}(\mathcal{G})$. Given a grammar $\mathcal{G}$ the unfolding construction produces an occurrence grammar which fully describes its behaviour recording all the possible graph items which are generated and the occurrences of productions. The unfolding is obtained by starting from the start graph (as type graph), applying productions in any possible way, without deleting items but only generating new ones, and recording such production instances in the type graph. The result is an occurrence grammar $\mathcal{U}(\mathcal{G})$ and a grammar morphism $f : \mathcal{U}(\mathcal{G}) \to \mathcal{G}$, called the *folding morphism*, which maps each item (instance of production or graph item) of the unfolding to the corresponding item of the original grammar. Because of space limitations, the construction is not formally defined here. In Section 4 we will show an example of an unfolding.

3 Interleaving Structures

Interleaving structures [4] are a semantic model which captures the behaviour of a system as the collection of its possible runs. They are used as a simpler intermediate model which helps in stating and proving the correctness of the diagnosis procedure.

An interleaving structure is essentially a collection of runs (sequences of events) satisfying suitable closure properties. Given a set E, we will denote by $E^{\odot}$ the set of sequences over E in which each element of E occurs at most once.

Definition 16 (interleaving structures). *A (labelled)* interleaving structure *is a tuple* $\mathcal{I} = (E, R, \Lambda, \lambda)$ *where* E *is a set of* events, $\lambda\colon E \to \Lambda$ *is a labelling of events and* $R \subseteq E^{\odot}$ *is the set of* runs*, satisfying: (i)* R *is prefix-closed, (ii)* R *contains the empty run* ε*, and (iii) every event* $e \in E$ *occurs in at least one run.*

The category of interleaving structures, as defined below, is adapted from [4] by changing the notion of morphisms in order to take into account the labels. This is needed to obtain a product which expresses a suitable form of synchronised composition.

Definition 17 (interleaving morphisms). *Let* $\mathcal{I}_i$ *with* $i \in \{1, 2\}$ *be interleaving structures. An* interleaving morphism *from* $\mathcal{I}_1$ *to* $\mathcal{I}_2$ *is a partial function* $\theta\colon E_1 \rightharpoonup E_2$ *on events such that*

1. $\Lambda_2 \subseteq \Lambda_1$*;*
2. *for each* $e_1 \in E_1$, $\theta(e_1)\downarrow$ *iff* $\lambda_1(e_1) \in \Lambda_2$ *and, in this case,* $\lambda_2(\theta(e_1)) = \lambda_1(e_1)$*;*
3. *for every* $r \in R_1$ *it holds that* $\theta^*(r) \in R_2$.

Morphism θ *is called a projection if* θ *is surjective on runs (as a function from* R_1 *to* R_2*). The category of interleaving structures and morphisms is denoted* **Ilv**.

An occurrence grammar can be easily mapped to an interleaving structure, by simply taking all the runs of the grammar.

Definition 18 (interleaving structures for occurrence grammars). *For an occurrence grammar $\mathcal{G}$ we denote by $Ilv(\mathcal{G})$ the interleaving structure which consists of all runs of $\mathcal{G}$, i.e., $Ilv(\mathcal{G}) = (P, Runs(\mathcal{G}), \Lambda, \lambda)$.*

We next characterise the categorical product in **Ilv**, which turns out to be, as in **GG**, the desired form of synchronised product.

Proposition 19 (product of interleaving structures). *Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be two interleaving structures. Then the product object $\mathcal{I}_1 \times \mathcal{I}_2$ is the interleaving structure $\mathcal{I} = (E, R, \Lambda, \lambda)$ defined as follows. Let*

$$E' = \{(e_1, e_2) \mid e_1 \in E_1, e_2 \in E_2, \lambda_1(e_1) = \lambda_2(e_2)\}$$
$$\cup \{(e_1, *) \mid e_1 \in E_1, \lambda_1(e_1) \notin \Lambda_2\} \cup \{(*, e_2) \mid e_2 \in E_2, \lambda_2(e_2) \notin \Lambda_1\}$$

and let $\pi_i : E \rightharpoonup E_i$ be the obvious (partial) projections. Then $R = \{r \in (E')^{\odot} \mid \pi_1^*(r) \in R_1, \pi_2^*(r) \in R_2\}$, $E = \{e' \in E' \mid e \text{ occurs in some run } r \in R\}$, $\Lambda = \Lambda_1 \cup \Lambda_2$ and λ is defined in the obvious way.

4 Diagnosis and Pruning

In this section we use the tools introduced so far in order to formalise the diagnosis problem. Then we show how, given a graph grammar model and an observation for such a grammar, the diagnosis can be obtained by first taking the product of the model and the observation, considering its unfolding and finally pruning such unfolding in order to remove incomplete explanations. As already mentioned, typically only a subset of the productions in the system is observable. Hence, for this section, we fix a graph grammar $\mathcal{G}$ with Λ as the set of labels, and a subset $\Lambda' \subseteq \Lambda$ of *observable labels*; an event or production is called *observable* if it has an observable label. In order to keep explanations finite, we will only consider systems that satisfy the following observability assumption (compare [15, 11]): any infinite run must contain an infinite number of observable productions.

In the sequel we will need to consider the runs of a system which have a number of observable events coinciding with the number of events in the observation. For this aim the following definition will be useful.

Definition 20 (n-runs of a grammar). *Let $\mathcal{G}$ be a graph grammar. For a given $n \in \mathbb{N}$ we denote by $Runs^n(\mathcal{G})$ the set of all runs for which the number of observable productions equals n.*

The outcome of the diagnosis procedure is an occurrence grammar which, intuitively, collects all the behaviours of the grammar $\mathcal{G}$ modelling the system, which are able to "explain" the observation.

An observation can be a sequence (in the case of a single observer) or a set of sequences (in the case of multiple distributed observers) of alarms (observable events). Here we consider, more generally, partially ordered sets of observations, which can be conveniently modelled as deterministic occurrence grammars $\mathcal{O}$.

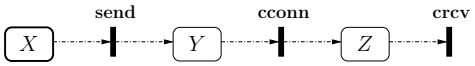

Fig. 3. A graph grammar representing an observation $\mathcal{O}$.

Definition 21 (observation grammar). *An* observation grammar *for a given grammar $\mathcal{G}$, with observable labels Λ', is a (finite) deterministic occurrence grammar labelled over Λ'.*

Given a sequence of observed events, we can easily construct an observation grammar $\mathcal{O}$ having that sequence as observable behaviour. It will have a production for each event in the sequence (with the corresponding label). Each such production consumes a resource generated by the previous one in the sequence (or an initial resource in the case of the first production). The same construction applies to general partially ordered sets observations.

Example. In the example grammar $\mathcal{M}$ (see Fig. 2), assume that we have the following observation: $\mathbf{snd}_2\ \mathbf{cconn}\ \mathbf{crcv}_2$, i.e., we observe, in sequence, the sending of a message, the creation of a connection and the reception of a corrupted message. These three observations can be represented by a simple grammar $\mathcal{O}$ (see Fig. 3) with three productions, each of which either consumes an initial resource or a resource produced by the previous production. These resources are modeled as 0-ary edges (labelled X, Y, Z). The initial graph is depicted with bold lines, and the left- and right-hand sides of the productions of the occurrence grammar are indicated by a Petri-net-like notation: events are drawn with black rectangles connected to the respective edges by dashed lines.

When unfolding the product of a grammar $\mathcal{G}$ with its observation $\mathcal{O}$, we obtain a grammar $\mathcal{U} = \mathcal{U}(\mathcal{G} \times \mathcal{O})$ with a morphism $\pi{:}\mathcal{U} \to \mathcal{O}$, arising as the image through the unfolding functor of the projection $\mathcal{G} \times \mathcal{O} \to \mathcal{O}$ (since the unfolding of an occurrence grammar is the grammar itself). Now, as grammar morphisms are simulations, given the morphism $\pi{:}\mathcal{U} \to \mathcal{O}$ we know that any computation in $\mathcal{U}$ is mapped to a computation in $\mathcal{O}$. Say that a computation in $\mathcal{U}$ is a full explanation of $\mathcal{O}$ if it is mapped to a computation of O including all its productions. As $\mathcal{U}$ can still contain events belonging only to incomplete explanations, the aim of *pruning* is to remove such events.

Definition 22 (pruning). *Let $\pi{:}\mathcal{U} \to \mathcal{O}$ be a grammar morphism from an occurrence grammar $\mathcal{U}$ to an observation $\mathcal{O}$. We define the pruning of π, denoted by $Pr(\pi)$, to be the grammar obtained from $\mathcal{U}$ by removing all events (including their consequences) not belonging to the following set:*

$$\{q \in \mathcal{P}_{\mathcal{U}} \mid \exists C \in \mathit{Conf}(\mathcal{U}){:}\, (q \in C \wedge \pi(C) = \mathcal{P}_{\mathcal{O}})\}$$

Discussing the efficiency of pruning algorithms is outside the scope of the paper; for sequential observations an on-the-fly algorithm is discussed in [5].

As described above, the diagnosis is constructed by first taking the product of $\mathcal{G}$ with the observation (this intuitively represents the system constrained by the observation). This product is then unfolded to get an explicit representation

of the possible behaviours explaining the observation. Finally, a pruning phase removes from the resulting occurrence grammar the events belonging (only) to incomplete explanations. This is formalised in the definition below.

Definition 23 (diagnosis grammar). *Let $\mathcal{G}$ be the grammar modelling the system of interest and let $\mathcal{O}$ be an observation. Take the product $\mathcal{G} \times \mathcal{O}$, the right projection $\varphi : \mathcal{G} \times \mathcal{O} \to \mathcal{O}$ and consider $\pi = \mathcal{U}(\varphi) : \mathcal{U}(\mathcal{G} \times \mathcal{O}) \to \mathcal{O}$.*

Then the occurrence grammar $Pr(\pi)$ is called the diagnosis grammar *of the model and the observation and denoted by $D(\mathcal{G}, \mathcal{O})$.*

Note that since the observability assumption holds, it can easily be shown that the diagnosis grammar is finite, whenever the observation is finite.

Example. We can compute the product of grammars $\mathcal{M}$ and $\mathcal{O}$ and unfold it. For reasons of space Fig. 4 shows only a prefix of the unfolding that depicts one possible explanation: here the message is sent (event **a**) and crosses the first connection (b). Possibly concurrently a new connection between the two intermediate nodes is created (**c**), which is then also crossed by the message (d). Again in a possibly concurrent step the last connection is corrupted (e), leading to the corruption of the message (f) and its reception by the receiver (**g**). Observable events are indicated by bold face letters.

Several events of the unfolding have been left out due to space constraints, for instance:

- Events belonging to alternative explanations: the corruption of the first connection or the corruption of the newly created middle connection (or the corruption of any non-empty subset of these connections). Alternatively it might also have been the case that the other sender/receiver pair handles the message, while the connection (which is not involved in any way) is created between the two intermediate nodes.
- Events that happen concurrently but are not directly related to the failure, such as the corruption of a connection over which no message is sent.

Furthermore there are events belonging to prefixes of the unfolding that cannot be extended to a full observation. For instance, the unfolding would contain concurrent events representing sending by the right-hand sender and the creation of a new connection leading from right to left instead of left to right. However, this is a false trail since this would never cause the reception of a message by the right-hand receiver. These incomplete explanations are removed from the unfolding in the pruning phase.

Note that—due to the presence of concurrent events—the unfolding is a much more compact representation of everything that might have happened in the system than the set of all possible interleavings of events.

5 Correctness of the Diagnosis

We now show our main result, stating that the runs of the diagnosis grammar properly capture all those runs of the system model which explain the observation. This is done by exploiting the coreflection result (Theorem 15) and by

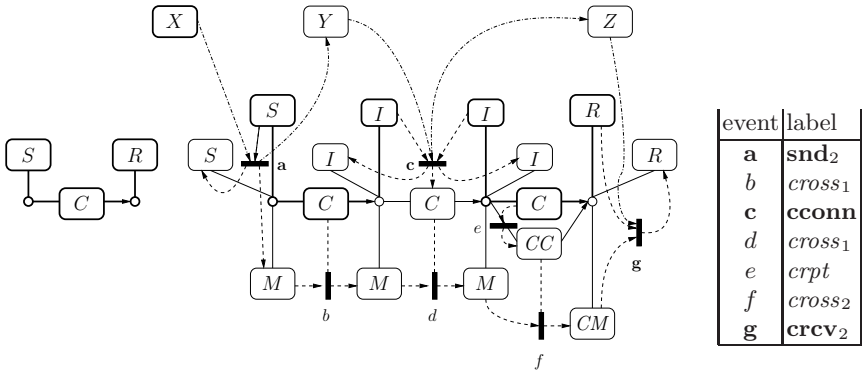

Fig. 4. Running example: prefix of the unfolding of the product.

additionally taking care of the pruning phase (Definition 22). We first need some technical results.

Lemma 24. *Let $\mathcal{G}_1, \mathcal{G}_2$ be two occurrence grammars. Consider the product of the two grammars and its image through the Ilv functor as shown below. Furthermore consider the product of the interleaving structures $Ilv(\mathcal{G}_1)$, $Ilv(\mathcal{G}_2)$. Then the mediating morphism δ is a projection which is total on events.*

$$\begin{array}{ccccc} Ilv(\mathcal{G}_1) & \xleftarrow{\delta_1} & Ilv(\mathcal{G}_1) \times Ilv(\mathcal{G}_2) & \xrightarrow{\delta_2} & Ilv(\mathcal{G}_2) \\ & {\scriptstyle \pi_1}\nwarrow & \uparrow{\scriptstyle \delta} & \nearrow{\scriptstyle \pi_2} & \\ & & Ilv(\mathcal{G}_1 \times \mathcal{G}_2) & & \end{array}$$

To lighten the notation, hereafter, given an interleaving structure $\mathcal{I}$ we write $\lambda^*(I)$ for $\lambda^*(R_I)$. Recall that, given $f : \Lambda_1 \rightharpoonup \Lambda_2$, $f^* : \Lambda_1^* \to \Lambda_2^*$ denotes the (non-strict) extension of f to sequences. Then $f^{-1} : \mathcal{P}(\Lambda_2^*) \to \mathcal{P}(\Lambda_1^*)$ is its inverse.

Lemma 25. *Let $\mathcal{G}_1, \mathcal{G}_2$ be two occurrence grammars and let $f_i \colon \Lambda_1 \cup \Lambda_2 \rightharpoonup \Lambda_i$ ($i \in \{1, 2\}$) be the obvious partial inclusions. Then it holds that*

$$\lambda^*(Ilv(\mathcal{G}_1 \times \mathcal{G}_2)) = f_1^{-1}(\lambda_1^*(Ilv(\mathcal{G}_1))) \cap f_2^{-1}(\lambda_2^*(Ilv(\mathcal{G}_2))).$$

The next proposition shows that considering the product of the original grammar $\mathcal{G}$ and of the observation $\mathcal{O}$, taking its unfolding and the corresponding labelled runs, we obtain exactly the runs of $\mathcal{G}$ compatible with the observation.

Proposition 26. *Let $\mathcal{G}$ be a grammar and $\mathcal{O}$ an observation, where Λ is the set of labels of $\mathcal{G}$ and $\Lambda' \subseteq \Lambda$ the set of labels of $\mathcal{O}$. Furthermore let $f \colon \Lambda \rightharpoonup \Lambda'$ be the obvious partial inclusion. Then it holds that:*

$$\lambda^*(Ilv(\mathcal{U}(\mathcal{G} \times \mathcal{O}))) = \lambda^*(Runs(\mathcal{G})) \cap f^{-1}(\lambda^*(Runs(\mathcal{O}))).$$

We can conclude that the described diagnosis procedure is complete, i.e., given an observation of size n, the runs of the diagnosis grammar $D(\mathcal{G}, \mathcal{O})$ with

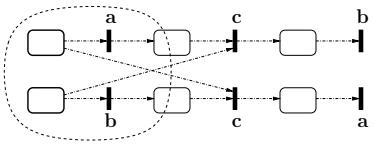

Fig. 5. Spurious runs in a diagnosis grammar.

n observable events are in 1-1 correspondence with those runs of $\mathcal{G}$ that provide a full explanation of the observation. As a preliminary result, on the basis of Proposition 26 one could have shown that the same holds replacing the diagnosis grammar with $\mathcal{U}(\mathcal{G} \times \mathcal{O})$, i.e., the unpruned unfolding. The result below additionally shows that no valid explanation is lost during the pruning phase.

Theorem 27 (correctness of the diagnosis). *With the notation of Proposition 26 it holds that:*

$$\lambda^*(Runs^n(D(\mathcal{G}, \mathcal{O}))) = \lambda^*(Runs(\mathcal{G})) \cap f^{-1}(\lambda^*(Runs^n(\mathcal{O}))).$$

That is, the maximal interleavings of the diagnosis grammar (seen as label sequences) are exactly the runs of the model which explain the full observation.

Observe that, due to the nondeterministic nature of the diagnosis grammar, events which are kept in the pruning phase as they are part of some full explanantion of the observation, can also occur in a different configuration. As a consequence, although all inessential events have been removed, the diagnosis grammar can still contain spurious configurations which cannot be extended to full explanations. As an example, consider the graph grammar $\mathcal{G}$ in Fig. 5, given in a Petri-net-like notation. Assume we observe three unordered events **a**, **b**, **c**. Then the unfolding of the product basically corresponds to $\mathcal{G}$ itself. In the pruning phase nothing is removed. However there is a configuration (indicated by the dashed closed line) that cannot be further extended to an explanation.

6 Conclusion

In this paper we formalised event-based diagnosis for systems with variable topologies, modelled as graph transformation systems. In particular we have shown how to exploit the coreflection result for the unfolding of graph grammars in order to show the correctness of a diagnosis procedure generating partially ordered explanations for a given observation.

We are confident that the approach presented in the paper, although developed for tranformation systems over hypergraphs, can be generalised to the more abstract setting of adhesive categories. In particular we are currently working on a generalization of the unfolding procedure that works for SPO-rewriting in (suitable variations of) adhesive categories [12]. This would allow one to have a kind of parametric framework which can be used to instantiate the results of this paper to more general rewriting theories.

We are also interested in distributed diagnosis where every observer separately computes possible explanations of local observations that however have to be synchronized. In [4] we already considered distributed unfolding of Petri nets; for *diagnosis* however, the non-trivial interaction of distribution and pruning has to be taken into account. Distribution will require the use of pullbacks of graph morphisms, in addition to products.

References

1. P. Baldan, T. Chatain, S. Haar, and B. König. Unfolding-based diagnosis of systems with an evolving topology. Technical Report 2008-2, Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen, 2008.
2. P. Baldan, A. Corradini, and U. Montanari. Contextual Petri nets, asymmetric event structures and processes. *Information and Computation*, 171(1):1–49, 2001.
3. P. Baldan, A. Corradini, U. Montanari, and L. Ribeiro. Unfolding semantics of graph transformation. *Information and Computation*, 205:733–782, 2007.
4. P. Baldan, S. Haar, and B. König. Distributed unfolding of Petri nets. In *Proc. of FOSSACS '06*, pages 126–141. Springer, 2006. LNCS 3921.
5. A. Benveniste, E. Fabre, S. Haar, and C. Jard. Diagnosis of asynchronous discrete event systems, a net unfolding approach. *IEEE Transactions on Automatic Control*, 48(5):714–727, 2003.
6. R. Bruni and H. C. Melgratti. Non-sequential behaviour of dynamic nets. In *ICATPN*, volume 4024 of *LNCS*, pages 105–124. Springer, 2006.
7. C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.
8. T. Chatain and C. Jard. Models for the supervision of web services orchestration with dynamic changes. In *AICT/SAPIR/ELETE*, pages 446–451. IEEE, 2005.
9. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.
10. E. Fabre, A. Benveniste, S. Haar, and C. Jard. Distributed monitoring of concurrent and asynchronous systems. *Journal of Discrete Event Dynamic Systems*, 15(1):33–84, 2005.
11. S. Haar, A. Benveniste, E. Fabre, and C. Jard. Partial order diagnosability of discrete event systems using Petri net unfoldings. In *Proc. 42nd IEEE Conf. on Decision and Control (CDC)*, 2003.
12. S. Lack and P. Sobociński. Adhesive and quasiadhesive categories. *RAIRO – Theoretical Informatics and Applications*, 39(3), 2005.
13. M. Löwe. Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science*, 109:181–224, 1993.
14. M. Löwe, M. Korff, and A. Wagner. An Algebraic Framework for the Transformation of Attributed Graphs. In M.R. Sleep, M.J. Plasmeijer, and M.C. van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, pages 185–199. Wiley, London, 1993.
15. M. Sampath, R. Sengupta, K. Sinnamohideen, S. Lafortune, and D. Teneketzis. Failure diagnosis using discrete event models. *IEEE Trans. on Systems Technology*, 4(2):105–124, 1996.
16. Y. Wang, T.-S. Yoo, and S. Lafortune. Diagnosis of discrete event systems using decentralized architectures. *Discrete Event Dynamic Systems*, 17(2):233–263, 2007.