Unfolding-based Diagnosis of Systems with an Evolving Topology

Paolo Baldan, Thomas Chatain, Stefan Haar, Barbara König

Unfolding-based Diagnosis of Systems with an Evolving Topology[☆]

Paolo Baldan[a,1,*], Thomas Chatain[b], Stefan Haar[b], Barbara König[c]

[a]*Dipartimento di Matematica Pura e Applicata, Università di Padova, Italy*
[b]*LSV, CNRS & ENS de Cachan, INRIA, France*
[c]*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 should hence be easily applicable to a wide class of system models, including graph grammars and Petri nets.

Key words: Model-based diagnosis, distributed and concurrent systems, partial order methods, graph rewriting
2010 MSC: 68Q60, 68Q85, 68Q42, 68M15

1. Introduction

The *event-oriented model-based diagnosis* problem is a classical topic in discrete event systems [1, 2]. 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 [3] 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 [3, 4], the global diagnosis is obtained as the fusion of local decisions: this *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.

[☆]Supported by RNRT project SWAN, INRIA Sabbatical program, the DFG project SANDS and project AVIAMO of the University of Padova.
[*]Corresponding author
Email addresses: baldan@math.unipd.it (Paolo Baldan), Thomas.Chatain@lsv.ens-cachan.fr (Thomas Chatain), haar@lsv.ens-cachan.fr (Stefan Haar), barbara_koenig@uni-due.de (Barbara König)

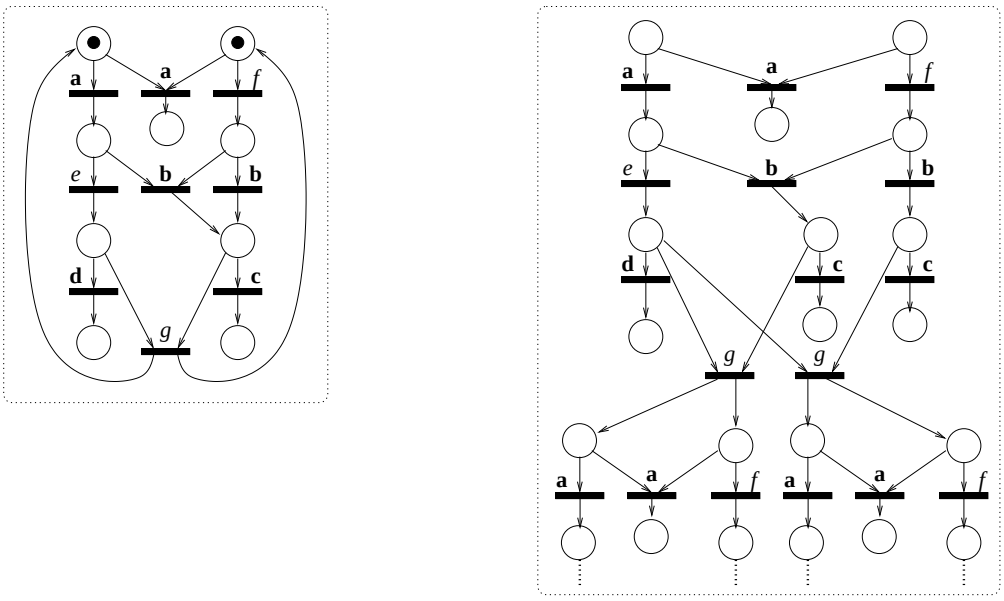


Figure 1: A Petri net (left) and a fragment of its unfolding (right).

In order to provide some intuition about unfoldings, an example of a Petri net and a fragment of its unfolding are reported in Fig. 1. In the Petri net, for later use, a bold face label is used to denote alarms, i.e., observable transitions. More explicitly, transitions labelled **a**, **b**, **c**, **d** are observable, while e, f are not. The unfolding is inductively constructed, starting from the initial marking and recording, step after step, any possible firing of a transition, with the tokens it produces. Hence transitions and places in the unfolding can be seen as occurrences of firing of transitions (events) and of tokens in computations of the original net. The causal dependencies and conflicts between items of the unfolding are made explicit by the struture of the unfolding itself. For instance, let us focus on the upper part of the unfolding. The two events labelled **a** are in conflict, since they consume a common resource, while the leftmost event labelled **a** is a cause for the event labelled e. Instead, the absence of dependencies between the events e and f, means that such events are concurrent and thus they can interleave in any way.

While [5] and subsequent work in this direction were 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 [6, 7]. 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 observation, it is not the model itself that is unfolded, but the product of the model with the observations, which intuitively represents the original system constrained by the observation; under suitable observability assumptions, only a finite prefix of the unfolding must be considered. 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.

The steps of the diagnosis procedure can be illustrated, for the Petri net case, by referring to the net in Fig. 1. Assume that the observation is given by the sequence **a b c**. The idea consists in representing the observation as a special system, in this case a Petri net. This can be easily done, as shown in the left part of Fig. 2. Then, taking the product of the system with the observation, we force events in the system to occur synchronously with those of the observation net. The unfolding of the product intuitively represents all the runs of the system which are consistent with the observation. For the considered example, such an unfolding is represented in the right part of Fig. 2 (for the sake of readability some places which do not influence the overall behaviour are omitted). Some events, although compatible with the observation, cannot be part of a complete explanation of the observation. For instance, in the example, after firing the transition labelled **a** in the middle, depicted in grey, no other event can be fired. Hence there

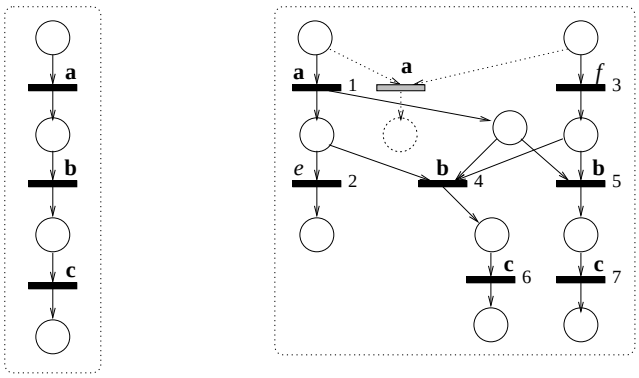


Figure 2: An observation (left) and the corresponding diagnosis net (right).

is no way of explaining the occurrence of **b** and **c** in the observation. This transition is removed in the pruning phase, which produces the actual diagnosis. Observe that, even in this simple case, the set of possible sequential executions explaining the observation would have been larger than the diagnosis in the form of an unfolding, as concurrent events can interleave in any way. For instance, indicating an event with the pair (*label*, *number*), a possible explanation would be $(\mathbf{a}, 1)\,(e, 2)\,(f, 3)\,(\mathbf{b}, 5)\,(\mathbf{c}, 7)$. But since event $(f, 3)$ is concurrent with $(\mathbf{a}, 1)$ and $(e, 2)$, also other interleavings, such as $(\mathbf{a}, 1)\,(f, 3)\,(e, 2)\,(\mathbf{b}, 5)\,(\mathbf{c}, 7)$ or $(f, 3)\,(\mathbf{a}, 1)\,(e, 2)\,(\mathbf{b}, 5)\,(\mathbf{c}, 7)$ are valid explanations. Additionally, since event $(e, 2)$ is not observable and concurrent with $(f, 3)$, $(\mathbf{b}, 5)$, $(\mathbf{c}, 7)$, it can be omitted or inserted in any position after $(\mathbf{a}, 1)$, leading to several more valid explanations.

The diagnosis technique proposed for graph grammars, which, as explained above, follows analogous steps, is shown to be correct. More precisely, we prove that the runs of the unfolding produced by the diagnosis procedure properly capture all those runs of the model which explain the observation. This non-trivial result relies on 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.

The rest of the paper is structured as follows. In Section 2 we introduce the category of graph grammars used in the paper, characterise the product in such a category and discuss the unfolding semantics. In Section 3 we introduce interleaving structures, an intermediate semantic model which is instrumental in developing the theory of diagnosis. In Section 4 we formalise the diagnosis problem and show how to construct a diagnosis for a given system and observation. In Section 5 we prove the correctness of the diagnosis procedure and give some experimental results in Section 6. Finally in Section 7 we draw some conclusions and outline directions of future research. An appendix includes some auxiliary material about asymmetric event structures, along with the proof of a technical result needed in the paper.

This is an extended version of the conference paper [8], which includes detailed definition of the grammar morphisms needed to get the right notion of product, full proofs of the results and a section with experimental evaluations.

2. Graph Grammars and Grammar Morphisms

In this section we summarise the basics of graph rewriting in the *single-pushout* (SPO) approach [9]. We introduce a category of graph grammars, whose morphisms are a variation of those in [10] and we provide a characterisation of the induced categorical product, which turns out to be adequate for expressing the notion of composition needed in our diagnosis framework. Then we argue that the unfolding semantics smoothly extends to this setting and, as in [10], the unfolding construction can be characterised categorically as a universal construction. The existence of a satisfactory unfolding semantics 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 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 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 finite 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 every $i \in \{1, \dots, n\}$. 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 $i \in \{1, \dots, n\}$.

Definition 1 ((hyper)graph). 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 2 (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 $f : G \rightharpoonup H$. Instead, the stronger (apparently natural) condition $c_H \circ f_E = f_N^\perp \circ c_G$ would have imposed f to be defined over an edge whenever it is defined on all its adjacent nodes.

We will work with *typed graphs* [11, 12], which are graphs labelled over a structure that is itself a graph, called the *type graph*.

Definition 3 (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. It is called *edge-injective* if the component on edges of t_G is injective. The category of T-typed graphs and partial typed graph morphisms is denoted by T-**PGraph**.

In Fig. 3 the reader can find an example of a typed graph (top) with the corresponding type graph (bottom). Note that, when depicting a graph, nodes and edges are represented as circles and boxes, respectively. In our examples we have both unary (hyper-)edges (represented by boxes connected to one node only) and binary hyperedges (where the order of nodes is indicated by an arrow, going from the first to the second node.) The typing morphism is implicitly represented by labelling each item of the graph with the item of the type graph it is mapped to.

Definition 4 (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 from G to H using q (based on g)*, written $G \Rightarrow_q H$, 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 $g(L_q - dom(r_q))$, 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. 5. Its left-hand side contains a unary edge and its right-hand side is the empty graph. The application of **fail** to a graph is illustrated in Fig. 4, where the match of the left-hand side is indicated as shaded.

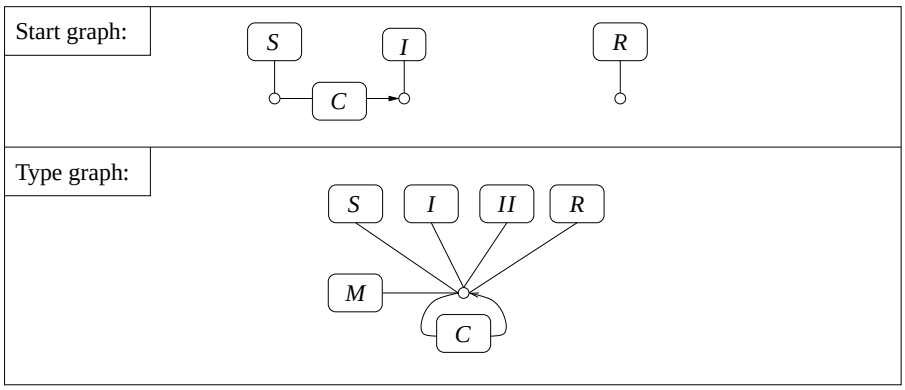


Figure 3: The start graph and type graph of the example graph grammar $\mathcal{S}$.

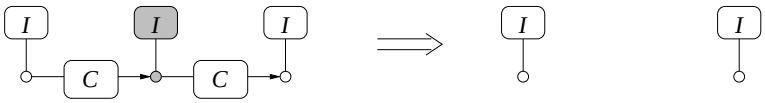


Figure 4: Dangling edge removal in SPO rewriting.

Definition 5 (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 $L_q \overset{r_q}{\rightharpoonup} R_q$. 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 6 (runs of a grammar). Let $\mathcal{G}$ be a graph grammar. Then $Runs(\mathcal{G})$ consists of all sequences $r_1 r_2 \dots r_n$ where $r_i \in P$ and $G_s \overset{r_1}{\Rightarrow} G_1 \overset{r_2}{\Rightarrow} G_2 \cdots \overset{r_n}{\Rightarrow} G_n$ for some $G_1, \dots, G_n$.

Example 1. As a first example, let us consider the graph grammar $\mathcal{S}$, whose start and type graph are in Fig. 3, while productions are given in Fig. 5. For productions (and the corresponding partial morphisms) we adopt the following graphical representation: 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.

Grammar $\mathcal{S}$ models a network with mobility whose nodes are either senders (labelled S), receivers (R) or intermediary nodes (I). Senders may send messages (production **snd**) which can then cross connections (production *cross*) and should finally arrive at a receiver (production **rcv**). The network is variable and of unbounded size as we allow the creation of new intermediary nodes and connections to such nodes (production **cnode**). Note that a newly created node is initially inactive (label II) and it will become active only later, by means of production *act*. Other rules are given to connect a sender to an intermediary node and an intermediary node to a receiver (productions **cconn**$_1$ and

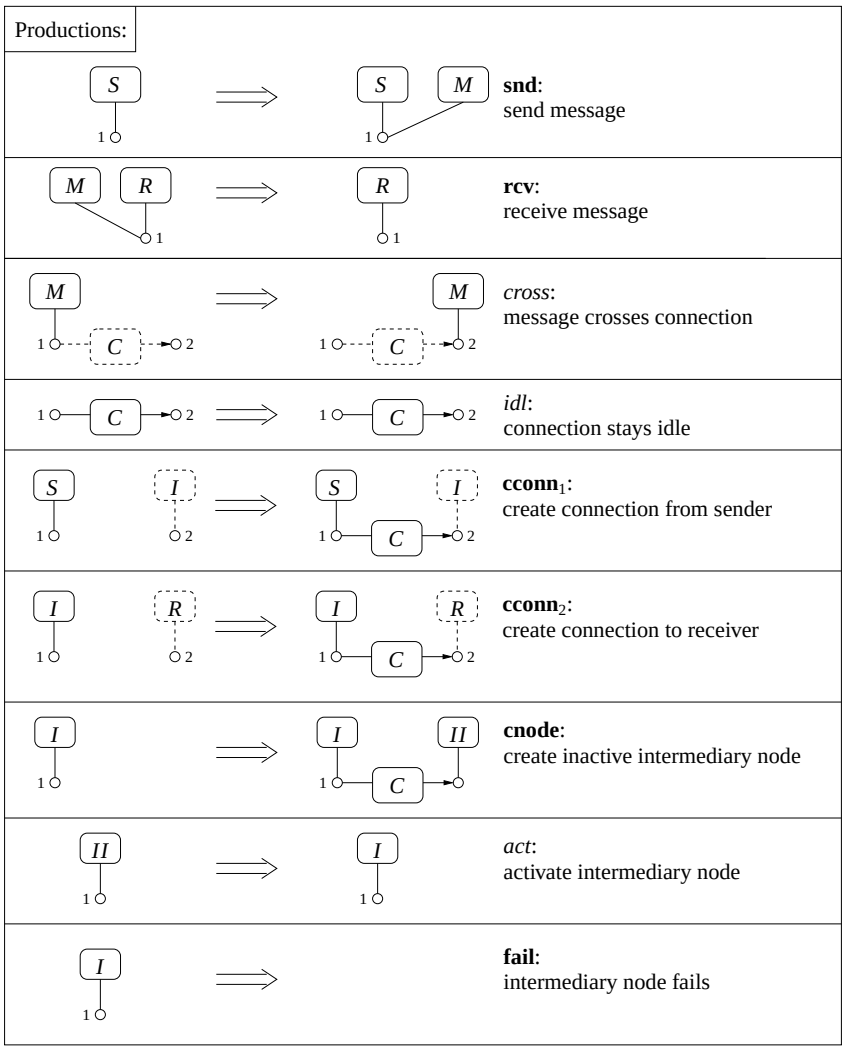


Figure 5: Example grammar $\mathcal{S}$: message passing over an evolving network.

cconn$_2$). Finally, a node can disappear (and in this case, as commented before, also all its connections are removed) as expressed by production **fail**. Production *idl* simply deletes and generates again a connection. It can be interpreted as a transient failure of the connection, which makes it unavailable for a while.

An example of run in $\mathcal{S}$ is given by the sequence **snd** *cross* **cconn**$_2$ *cross* **rcv**, where a message is generated by a sender, it travels towards a receiver and it is finally received. This is made possible by an extension of the network which, in the second step, is enriched with a new connection. A graphical representation of the run can be found in Fig. 6.

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

Definition 7 (semi-weighted SPO graph grammars). A grammar $\mathcal{G}$ is *semi-weighted* if (i) the start graph G_s is edge-injective, (ii) for each $q \in P$, for any $x, y \in E_{|R_q|} - E_{|L_q|}$ 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 edges" and (iii) in the start graph G_s and, for any $q \in P$, in L_q and in the graph $L_q \cup R_q$, there are no isolated nodes.

Intuitively, Conditions (i) and (ii) ensure that in a semi-weighted grammar each edge generated in a computation has a uniquely determined causal history. Condition (iii) essentially says that only edges carry semantic information, while

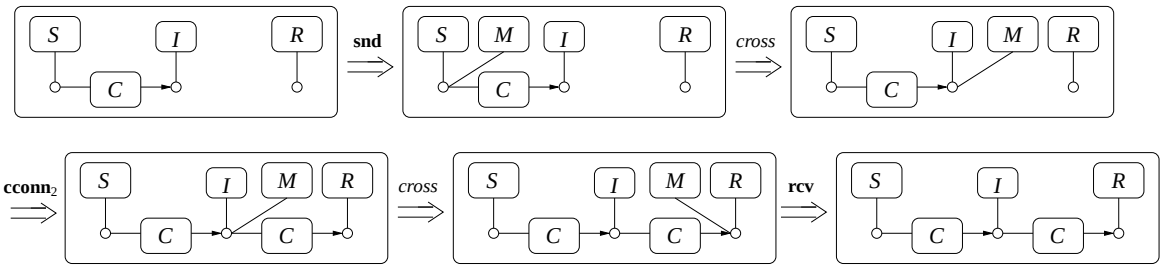


Figure 6: A run in the example grammar $\mathcal{S}$.

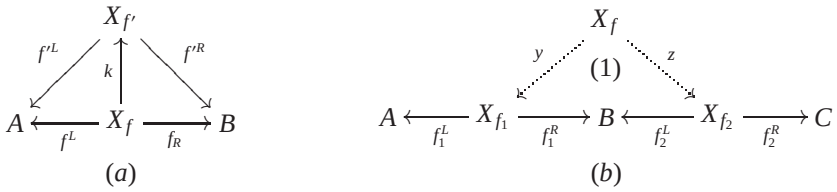


Figure 7: Equivalence and composition of spans.

nodes are just used as attaching points for edges. At a more technical level, it ensures that the correspondence between the type graphs of two graph grammars as established by a grammar morphism, is determined by the correspondence on edges. These facts are essential for the validity of Theorem 1.

We next introduce a category of graph grammars, which will be used to define products and to characterise the unfolding construction as a coreflection. The choice of arrows of the category is quite subtle: other possible notions of morphisms are conceivable but they do not necessarily provide the right notion of product and the coreflection result. In order to define grammar morphisms we need to introduce the notion of semi-abstract span, which, roughly speaking, provides a categorical generalisation of the notion of multirelation.

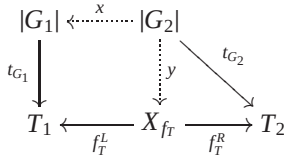
Given a category **C**, a *(concrete) span* $f : A \leftrightarrow B$ in **C** is a pair of total graph morphisms $f = \langle f^L : X_f \to A, f^R : X_f \to B \rangle$, where X_f is called the *support*.

A *semi-abstract span* $[f] : A \leftrightarrow B$ is an equivalence class of spans obtained by considering the support up to isomorphism, i.e., $[f] = \{f' : A \leftrightarrow B \mid \exists k\colon X_f \to X_{f'}.(k \text{ isomorphism} \wedge f'^L \circ k = f^L \wedge f'^R \circ k = f^R)\}$ (see Fig. 7.(a)).

If **C** is a category with pullbacks, then semi-abstract spans can be composed as follows: given two semi-abstract spans $[f_1] : A \leftrightarrow B$ and $[f_2] : B \leftrightarrow C$, their composition is the (equivalence class of a) span f constructed as in Fig. 7.(b) (i.e., $f^L = f_1^L \circ y$ and $f^R = f_2^R \circ z$), where the square (1) is a pullback. This allows one to consider a category **Span**(**C**) which has the same objects as **C** and semi-abstract spans in **C** as arrows.

The following definition generalises the notion of image of a set through a multirelation (see, e.g., [13]).

Definition 8 (pullback-retyping relation). Let $[f_T] : T_1 \leftrightarrow T_2$ be a semi-abstract span in **Graph**, let G_1 be a T_1-typed graph, and let G_2 be a T_2-typed graph. Then G_1 and G_2 are *related by pullback-retyping (via* $[f_T]$*)* if there exist total morphisms $x : |G_2| \to |G_1|$ and $y : |G_2| \to X_{f_T}$ such that the square in the following diagram is a pullback:



In this case we will write $f_T\{x,y\}(G_1,G_2)$, or simply $f_T(G_1,G_2)$ if we are not interested in morphisms x and y.

Some concrete examples of retypings will be discussed after Definition 10.

We are now ready to introduce grammar morphisms. Except for the treatment of the labels, these morphisms coincide with those in [10], which are, in turn, a generalisation of Winskel's morphisms for Petri nets (see [14]).

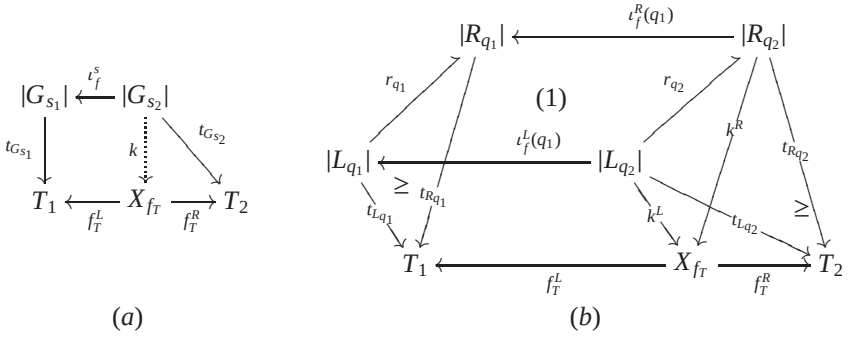


Figure 8: Diagrams for SPO grammar morphisms.

The latter ensure the existence of products, which can be interpreted as asynchronous compositions, and of some coproducts, modelling nondeterministic choice [15].

Besides the component specifying the (multi)relation between the type graphs, a morphism from $\mathcal{G}_1$ to $\mathcal{G}_2$ includes a (partial) mapping between production names. Furthermore a third component explicitly relates the (untyped) graphs underlying corresponding productions of the two grammars, as well as the graphs underlying the start graphs.

Definition 9 (grammar morphism). Let $\mathcal{G}_i$ ($i \in \{1,2\}$) be graph grammars such that $\Lambda_2 \subseteq \Lambda_1$. A morphism $f : \mathcal{G}_1 \to \mathcal{G}_2$ is a triple $\langle [f_T], f_P, \iota_f \rangle$ where

- $[f_T] : T_1 \leftrightarrow T_2$ is a semi-abstract span in **Graph**, called the *type-span*;
- $f_P : P_1 \to P_2 \cup \{\emptyset\}$ is a total function, where $\emptyset$ is a new production name (not in P_2), with associated production $\emptyset \rightharpoonup \emptyset$;
- ι_f is a family $\{\iota_f(q_1) \mid q_1 \in P_1\} \cup \{\iota_f^s\}$ of morphisms in **Graph** such that $\iota_f^s : |G_{s_2}| \to |G_{s_1}|$ and for each $q_1 \in P_1$, if $f_P(q_1) = q_2$, then $\iota_f(q_1)$ is a pair

$$\langle \iota_f^L(q_1) : |L_{q_2}| \to |L_{q_1}|, \iota_f^R(q_1) : |R_{q_2}| \to |R_{q_1}| \rangle.$$

such that the following conditions are satisfied:

1. *Preservation of the start graph.*
   There exists a morphism k such that $f_T\{\iota_f^s, k\}(G_{s_1}, G_{s_2})$, i.e., the diagram in Fig. 8.(a) commutes and the square is a pullback.
2. *Preservation of productions.*
   For each $q_1 \in P_1$, with $q_2 = f_P(q_1)$, there exist morphisms k^L and k^R such that the square (1) in Fig. 8.(b) commutes, and $f_T\{\iota_f^Y(q_1), k^Y\}(Y_{q_1}, Y_{q_2})$ for $Y \in \{L, R\}$.
3. *Preservation of labelling.*
   For each $q_1 \in P_1$, $f_P(q_1) \neq \emptyset$ iff $\lambda_1(q_1) \in \Lambda_2$ and, in this case, $\lambda_2(f_P(q_1)) = \lambda_1(q_1)$.

For technical convenience, the partial mapping on production names is represented as a total mapping by enriching the target set with a distinguished element $\emptyset$, representing "undefinedness". With respect to the morphisms in [10], note that here for the existence of a morphism from $\mathcal{G}_1$ to $\mathcal{G}_2$ we require that $\Lambda_2 \subseteq \Lambda_1$ and there are some restrictions on the labelling as expressed by Condition 3 above.

Definition 10 (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.

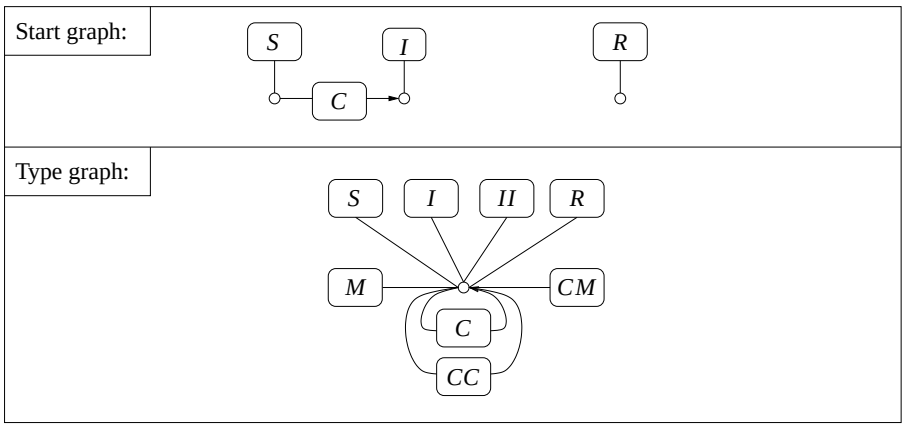


Figure 9: Start graph and type graph for the running example grammar $\mathcal{M}$.

Example 2. Let us consider a second graph grammar $\mathcal{M}$ which will be used as a running example. The start and type graph are given in Fig. 9, while the productions can be found in Fig. 10.

Grammar $\mathcal{M}$ still models an evolving network with message passing, but here a connection may spontaneously get corrupted (production *crpt*), a fact which causes the corruption of any message which crosses it (production $cross_2$). Note that in order to represent corrupted items there are two additional types in the type graph: *CC* for corrupted connections and *CM* for corrupted messages. It could also be natural to add another rule $cross_4$ modelling a corrupted message crossing a corrupted connection, but we omit this rule for keeping the presentation simpler.

The two example grammars $\mathcal{S}$ and $\mathcal{M}$ are used to illustrate the notion of grammar morphism. We define a morphism $f : \mathcal{M} \to \mathcal{S}$ which intuitively maps $\mathcal{M}$ into $\mathcal{S}$ by forgetting about the distinction between corrupted and non corrupted items. More formally, the type span is $f_T : T_\mathcal{M} \leftrightarrow T_\mathcal{S}$, where $X_{f_T} = T_\mathcal{M}$, the left leg $f_T^L : T_\mathcal{M} \to T_\mathcal{M}$ is the identity and the right leg $f_T^R : T_\mathcal{M} \to T_\mathcal{S}$ maps the two connection edges in $T_\mathcal{M}$ (i.e., *C* and *CC*) to the only connection edge in $T_\mathcal{S}$ (i.e., *C*); the same happens for messages, while any other item in $T_\mathcal{M}$ is mapped to the corresponding item in $T_\mathcal{S}$ (see Fig. 11).

The component on productions $f_P : P_\mathcal{M} \to P_\mathcal{S} \cup \{\emptyset\}$ maps productions **crcv** and *crpt* in $\mathcal{M}$ to **rcv** and *idl* in $\mathcal{S}$, respectively, while productions $cross_i$ ($i \in \{1,2,3\}$) in $\mathcal{M}$ are all mapped to production *cross* in $\mathcal{S}$. The remaining productions **snd**, **rcv**, $\mathbf{cconn}_i$ ($i \in \{1,2\}$), **cnode**, **act**, **fail** in $\mathcal{M}$ are mapped to the corresponding productions, with the same name, in $\mathcal{S}$. Note that in this case no production in $\mathcal{M}$ is mapped to $\emptyset$, which intuitively means that the mapping is total on productions. All the morphisms in the ι_f family are isomorphisms.

It can be easily seen that the conditions of Definition 9 concerning the preservation of start graph and productions are satisfied. Since, in this case, the left leg of the type span f_T^L is the identity, the pullback-retyping of a typed graph *G* will result in a diagram of the kind

$$\begin{array}{ccccc} |G| & \xleftarrow{id} & |G| & & \\ {\scriptstyle t_G}\downarrow & & {\scriptstyle t_G}\downarrow & \searrow^{t_G;f_T^R} & \\ T_\mathcal{M} & \xleftarrow[id]{} & T_\mathcal{M} & \xrightarrow[f_T^R]{} & T_\mathcal{S} \end{array}$$

i.e., it just amounts to a retyping via a post-composition with f_T^R.

We observe that morphisms can be more sophisticated, as spans can represent general (multi-)relations. The pullback retyping construction can, e.g., remove some types and multiply some others. For instance Fig. 12(a) shows a span f_T' (the left and right leg are implicitly given by the labelling) and Fig. 12(b) the graph which would result by applying the pullback retyping to the left-hand side of $cross_2$. Note that the connection *CC* disappears, while the message *M* is doubled.

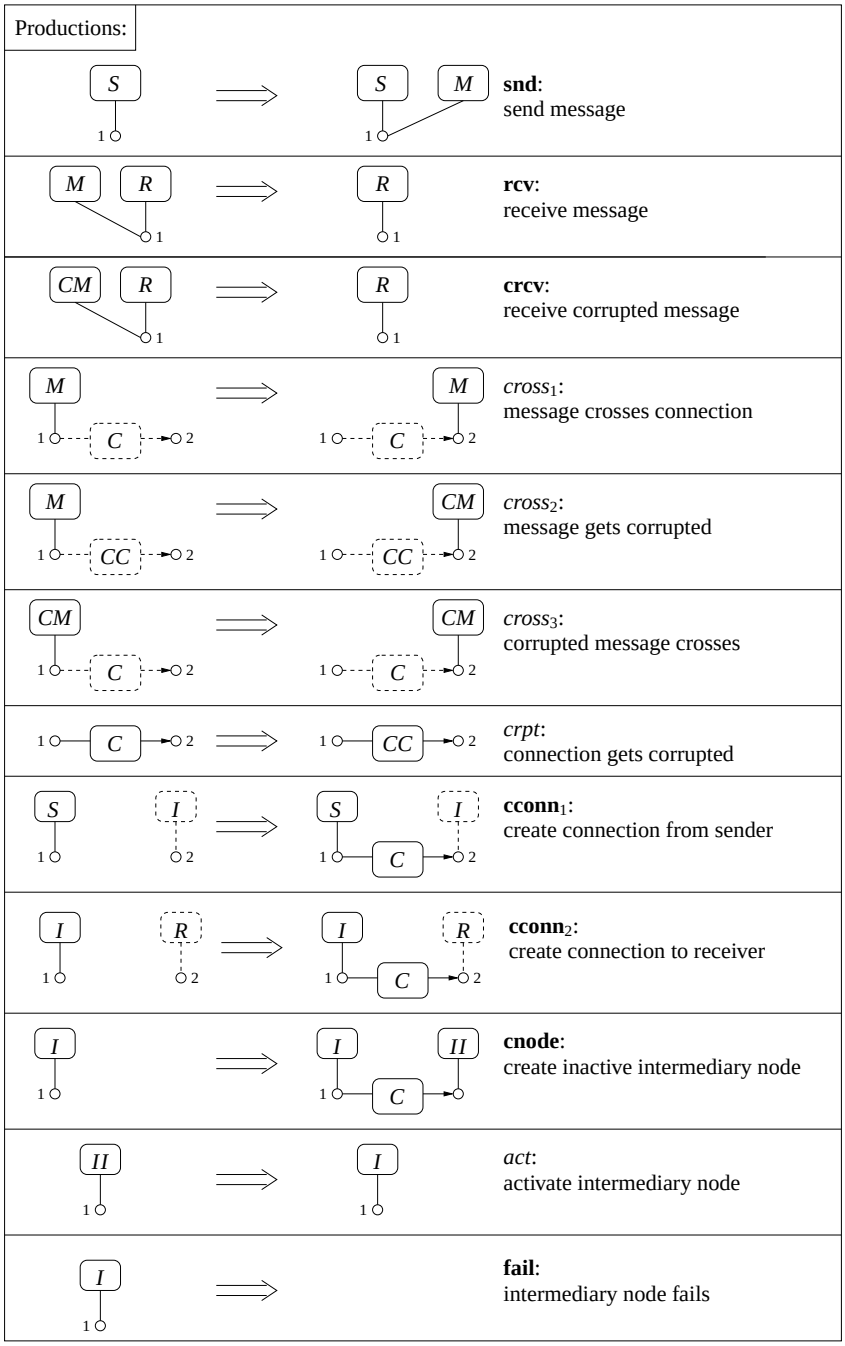


Figure 10: Productions for the running example grammar $\mathcal{M}$.

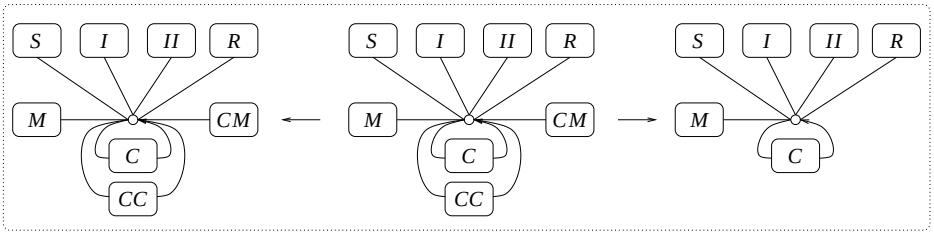
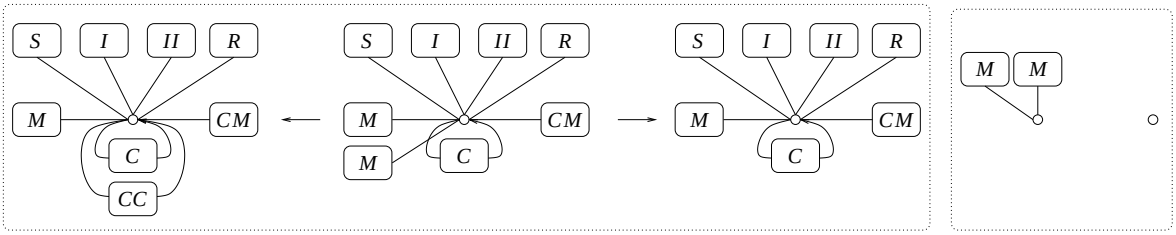


Figure 11: The type component of the morphism $f : \mathcal{M} \to \mathcal{S}$.



(a) (b)

Figure 12: An example of pullback-retyping.

2.2. Product in the category of grammars

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 $\mathcal{G}_2$: 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. At a more technical level, the type and start graph of the product grammar are obtained by taking the disjoint union of the type and start graphs of $\mathcal{G}_1$ and $\mathcal{G}_2$. Similarly, productions in the product grammar which arise from the synchronisation of productions of $\mathcal{G}_1$ and $\mathcal{G}_2$ are constructed by taking the disjoint union of the left- and right-hand sides of the original productions.

Proposition 1 (product of graph grammars). *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two graph grammars. Their product in* **GG** *is defined as $\mathcal{G} = \mathcal{G}_1 \times \mathcal{G}_2$ with the following components:*

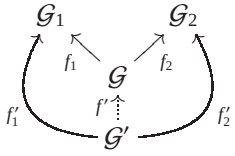
- $T = T_1 \uplus T_2$;
- $G_s = G_{s1} \uplus G_{s2}$, *with the obvious typing;*
- $P = \{(p_1, p_2) \mid \lambda_1(p_1) = \lambda_2(p_2)\} \cup \{(p_1, \emptyset) \mid \lambda_1(p_1) \notin \Lambda_2\} \cup \{(\emptyset, p_2) \mid \lambda_2(p_2) \notin \Lambda_1\}$;
- $\pi(p_1, p_2) = \pi_1(p_1) \uplus \pi_2(p_2)$, *where $\pi_i(\emptyset)$ is the empty rule $\emptyset \rightharpoonup \emptyset$;*
- $\Lambda = \Lambda_1 \cup \Lambda_2$;
- $\lambda(p_1, p_2) = \lambda_i(p_i)$, *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. The projections $f_i : \mathcal{G} \to \mathcal{G}_i$ ($i \in \{1, 2\}$) are morphisms $f_i = \langle f_{iT}, f_{iP}, \iota^{f_i} \rangle$, where

- $f_{iT} : T_1 \uplus T_2 \leftrightarrow T_i$ *has support T_i, the left leg $f_{iT}^L : T_i \to T_1 \uplus T_2$ is the obvious injection, while the right leg $f_{iT}^L : T_i \to T_i$ is the identity;*
- $f_{iP} : P \to P_i \uplus \{\emptyset\}$ *is the obvious projection;*
- *concerning the components of ι_{f_i}: for the start graph $\iota_{f_i}^s : G_{si} \to G_{s1} \uplus G_{s2}$ is the obvious inclusion; $\iota_{f_i}^L$ and $\iota_{f_i}^R$ are defined analogously.*

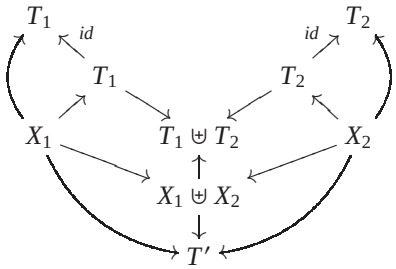
If $\mathcal{G}_1, \mathcal{G}_2$ are both semi-weighted grammars, then $\mathcal{G}$ as defined above is semi-weighted, and it is the product of $\mathcal{G}_1$ and $\mathcal{G}_2$ in **SGG**.

Proof. Let $\mathcal{G}'$ be another graph grammar with morphisms $f_i' \colon \mathcal{G}' \to \mathcal{G}_i$. We have to show that there exists a unique morphism $f' \colon \mathcal{G}' \to \mathcal{G}$ with $f_i \circ f' = f_i'$.



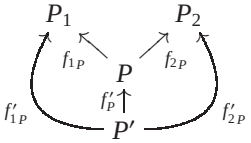
The morphism f' will be defined as follows:

- Assume that $f_{i\,T}' \colon T' \leftrightarrow T_i$ has support X_i. Then for $f_T' \colon T' \leftrightarrow T_1 \uplus T_2$ take $X_1 \uplus X_2$ as support, where the arrows $X_1 \uplus X_2 \to T'$ and $X_1 \uplus X_2 \to T_1 \uplus T_2$ are obtained as mediating morphisms (see diagram below).



  The category of graphs with total morphisms is an adhesive category with a strict initial object (the empty graph), which is hence extensive [16]. This implies that the two squares $X_i, T_i, X_1 \uplus X_2, T_1 \uplus T_2$ must be pullbacks. It follows that the composition of the spans f_T' and f_{iT} equals $f_{i\,T}'$. Furthermore the span f_T' is unique since, again by extensivity, taking $X_1 \uplus X_2$ as support is the only way to obtain two pullbacks; then the arrows of the span are fixed as mediating morphisms.

- Now define $f_P' \colon P' \to P \uplus \{\emptyset\}$ as follows: $f_P'(q') = (f_{1\,P}'(q'), f_{2\,P}'(q'))$ if at least one of the components of the pair is different from $\emptyset$ and $f_P'(q') = \emptyset$ otherwise. The situation is depicted in the diagram below:



  Clearly $f_{iP} \circ f_P' = f_{i\,P}'$ and it is easy to see that f_P' is unique. In fact, let $f_P'' \colon P' \to P \uplus \{\emptyset\}$ another mapping with $f_{iP} \circ f_P'' = f_{i\,P}'$ and let $q' \in P'$. It holds that $\lambda'(q') \in \Lambda' \supseteq \Lambda = \Lambda_1 \cup \Lambda_2$. We distinguish two cases

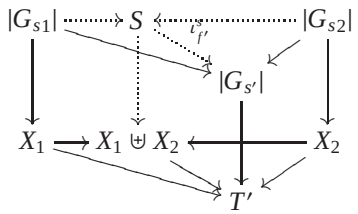
  - $\lambda'(q') \in \Lambda_1 \cup \Lambda_2$: in this case at least one of $f_{1\,P}'(q')$ and $f_{2\,P}'(q')$ must be different from $\emptyset$. Hence, since the f_{iP} are simply projections, we deduce that $f_P''(q') = (f_{1\,P}'(q'), f_{2\,P}'(q')) = f_P'(q')$.
  - $\lambda'(q') \notin \Lambda$: in this case $f_{i\,P}'(q') = \emptyset$ for $i \in \{1, 2\}$. Since $(\emptyset, \emptyset)$ is not in P, we deduce that $f_P''(q') = \emptyset = f_P'(q')$.

  Note that for the argument above, the condition about labels in the definition of grammar morphisms (Condition 3 in Definition 9) is essential.

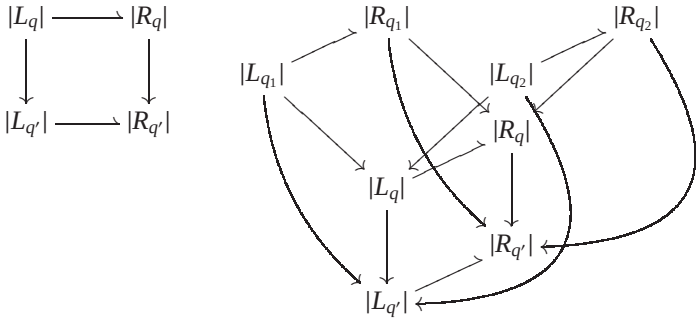
- Finally, the components of $\iota_{f'}$ can be defined as follows: in the diagram below take the pullback of $X_1 \uplus X_2 \to T'$ and $|G_{s'}| \to T'$ in order to obtain S. The arrows $|G_{si}| \to S$ are obtained as mediating morphisms.

  Since the squares $|G_{si}|$, $|G_{s'}|$, X_i, T' are pullbacks, the squares $|G_{si}|$, X_i, S, $X_1 \uplus X_2$ are also pullbacks, due to pullback splitting. Hence, since we are working in an extensive category, S must be the coproduct of $|G_{s1}|$, $|G_{s2}|$

and thus it is isomorphic to $|G_s|$. This gives morphism $\iota^s_{f'}$ and shows that $|G_s|$ can be obtained from $|G_{s'}|$ via pullback-retyping. The arrow $\iota^s_{f'}$ must be unique since it is the mediating morphism of a coproduct.



Analogously one can define morphisms $\iota^L_{f'} : |L_q| \rightharpoonup |L_{q'}|$, $\iota^R_{f'} : |R_q| \rightharpoonup |R_{q'}|$ for productions $q' \in P'$, $q \in P$ where $f'_P(q') = q$. It remains to show that the square on the left below commutes. Let $q_i = f_{1P}(q)$ and consider the diagram on the right below. It is known that all triangles and squares—apart from the bottom square—commute and that $|L_q|$ is the coproduct of $|L_{q_1}|$ and $|L_{q_2}|$. Since the two coprojections $|L_{q_i}| \rightharpoonup |L_q|$ are jointly epi (also in the category of partial morphisms), commutativity follows from a simple diagram-chasing argument.



Note also that whenever $\mathcal{G}_1$ and $\mathcal{G}_2$ are semi-weighted, then $\mathcal{G}$ is semi-weighted. And since every arrow in **GG** between semi-weighted grammars is also an arrow in **SGG**, the same argument applies. □

2.3. 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 plays the role of the set of *places* of a net, whereas the productions in P correspond to the *transitions*.

Definition 11 (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 $E_T \cup N_T$:

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

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 12 (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 it happens 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 more thorough discussion of asymmetric event structures see Appendix A.1 and [17].

Definition 13 (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 $\underline{q} \cap {}^\bullet q' \neq \emptyset$ then $q \nearrow q'$;
2. if ${}^\bullet q \cap {}^\bullet q' \neq \emptyset$ 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 are represented by cycles of asymmetric conflict: if $q_1 \nearrow q_2 \nearrow \ldots \nearrow q_n \nearrow q_1$ then the entire set $\{q_1, \ldots, q_n\}$ 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 called *finitary* if for any $x \in X$, the set $\{y \in X \mid R(y, x)\}$ is finite.

Definition 14 (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. any item x in T is created by at most one production in P, i.e., $|{}^\bullet x| \leq 1$;
3. 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);

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.

Intuitively, by condition 1 the causes of any event are finite and free of conflicts (cycles of asymmetric conflict), while condition 2 ensures that any item is generated at most once in a computation. By condition 3, the start graph of an occurrence grammar is determined by $Min(\mathcal{O})$. An occurrence grammar is deterministic when it does not contain conflicts so that all its productions can be executed in the same computation. Given two occurrence grammars $\mathcal{O}_1$ and $\mathcal{O}_2$, we say that $\mathcal{O}_1$ is a *sub-grammar* of $\mathcal{O}_2$, if $\mathcal{O}_1 \subseteq \mathcal{O}_2$, componentwise, and the inclusion of $\mathcal{O}_1$ into $\mathcal{O}_2$ is a grammar morphism. 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 15 (configuration). Let $\mathcal{O}$ be an occurrence grammar. A *configuration* is a subset $C \subseteq P$ satisfying the following requirements

1. for any $q \in C$ it holds that $\lfloor q \rfloor \subseteq C$;
2. $\nearrow_C$, the asymmetric conflict restricted to C, is acyclic and finitary.

The set of configurations of $\mathcal{O}$ is denoted as $Conf(\mathcal{O})$.

It is shown in [10] that, indeed, configurations faithfully represent computations in an occurrence grammar: all the productions in a configuration can be applied in a derivation exactly once in any order compatible with $\nearrow$, and all and only the derivations in $\mathcal{O}$ can be obtained in this way. Hence the runs of an occurrence grammar are exactly the linearisations, compatible with the asymmetric conflict relation, of its configurations, i.e., the following holds:

Proposition 2 (configurations as runs). *Let O be an occurrence grammar. Then* $Runs(O) = \{q_1 \dots q_n \mid \{q_1, \dots, q_n\} \in Conf(O) \ \wedge \ \forall i < j.\ \neg(q_j \nearrow q_i)\}$.

Since occurrence grammars are particular semi-weighted grammars, there is an inclusion functor $\mathcal{I} : \mathbf{OGG} \to \mathbf{SGG}$. As an easy corollary of [10, Theorem 45], this functor has a right adjoint. We remark that this theorem would not hold if we considered **GG** instead of **SGG**.

Theorem 1 (coreflection). *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, i.e., given two grammars $\mathcal{G}_1$ and $\mathcal{G}_2$, it holds that $\mathcal{U}(\mathcal{G}_1 \times \mathcal{G}_2) = \mathcal{U}(\mathcal{G}_1) \times \mathcal{U}(\mathcal{G}_2)$.

The result in [10] 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 of $\mathcal{G}$ is constructed inductively by starting from a grammar which only includes the start graph of $\mathcal{G}$ (which is also used as a type graph), and then extending such grammar, at any step, by applying productions in any possible way to the type graph, without deleting items but only generating new ones, and recording the corresponding production instances. 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. The construction is not formally defined here, we refer the reader to [10]. In Section 4 we will show an example of an unfolding.

As an immediate consequence of the fact that the unfolding $\mathcal{U}(\mathcal{G})$ completely captures the behaviour of a grammar $\mathcal{G}$, we have the following result.

Proposition 3 (completeness of the unfolding). *For any semi-weighted graph grammar $\mathcal{G}$ it holds that*

$$\lambda^*(Runs(\mathcal{U}(\mathcal{G}))) = \lambda^*(Runs(\mathcal{G})).$$

3. Interleaving Structures

Interleaving structures [18] 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 components of an interleaving structure $\mathcal{I}$ will be denoted by E, R, Λ, λ, possibly with subscripts. The category of interleaving structures, as defined below, is adapted from [18] 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 it is surjective on runs, i.e., $\theta^* : R_1 \to R_2$ is surjective. The category of interleaving structures and morphisms is denoted **Ilv**.

Observe that an interleaving morphism $\theta : \mathcal{I}_1 \to \mathcal{I}_2$ is necessarily injective on the events occurring in each run, i.e., for any run $r_1 = e_1 \dots e_k \in R_1$, for $i \neq j$ we have $\theta(e_i) \neq \theta(e_j)$, when both are defined. Otherwise, $\theta(r)$ could not be a run in $\mathcal{I}_2$ as it would contain two occurrences of the same event.

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 O we define $Ilv(O) = (P, Runs(O), \Lambda, \lambda)$.

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

Proposition 4 (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*

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

and let $\pi_i : E \rightharpoonup E_i$ be the obvious partial projections (e.g., $\pi_1(e_1, x_2) = e_1$ and $\pi_1(*, x_2) \uparrow$ for $e_1 \in E_1$ and $x_2 \in E_2 \cup \{*\}$). 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.

Proof. Let $\mathcal{I}'$ be any interleaving structure and let $\theta_i : \mathcal{I}' \to \mathcal{I}_i$ be morphisms. Let us define $\theta : \mathcal{I}' \to \mathcal{I}$ as follows:

$$\begin{aligned} &\theta(e') \uparrow && \text{if } \theta_1(e')\uparrow \text{ and } \theta_2(e')\uparrow \\ &\theta(e') && = \begin{cases} (\theta_1(e'), \theta_2(e')) & \text{if } \theta_1(e')\downarrow \text{ and } \theta_2(e')\downarrow \\ (\theta_1(e'), *) & \text{if } \theta_1(e')\downarrow \text{ and } \theta_2(e')\uparrow \\ (*, \theta_2(e')) & \text{if } \theta_1(e')\uparrow \text{ and } \theta_2(e')\downarrow \end{cases} \end{aligned}$$

Trivially, the diagram, seen in the category of sets and partial functions, commutes and θ is uniquely determined.

Hence to conclude we just need to show that θ is a well-defined interleaving morphism. In fact,

1. Since there are morphisms $\theta_i : \mathcal{I}' \to \mathcal{I}_i$, necessarily $\Lambda_i \subseteq \Lambda'$, for $i \in \{1, 2\}$ and thus $\Lambda = \Lambda_1 \cup \Lambda_2 \subseteq \Lambda'$.
2. For each $e' \in E'$, $\theta(e')\downarrow$ iff $\lambda'(e') \in \Lambda$. In more detail:
$$\begin{aligned} \theta(e')\downarrow &\iff \theta_1(e')\downarrow \text{ or } \theta_2(e')\downarrow \\ &\iff \lambda'(e') \in \Lambda_1 \text{ or } \lambda'(e') \in \Lambda_2 \\ &\iff \lambda'(e') \in \Lambda_1 \cup \Lambda_2 = \Lambda. \end{aligned}$$
And clearly, when defined, $\lambda'(e') = \lambda(\theta(e'))$
3. For any $r' \in R'$, $\theta^*(r') \in R$. In fact, notice that, due to commutativity, $\pi_i^*(\theta^*(r')) = \theta_i^*(r') \in R_i$. Hence, by construction, $\theta^*(r') \in R$. □

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

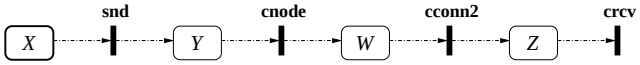


Figure 13: A graph grammar representing an observation $\mathcal{A}$, given in a Petri-net-like notation.

order to keep explanations finite, we will only consider systems that satisfy the following *observability assumption* (compare [2, 19]): *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 19 (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.

Definition 20 (observation grammar). An *observation grammar* $\mathcal{A}$ 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 $\mathcal{A}$ 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 of observations.

Example 3. In the running example grammar $\mathcal{M}$ (see Fig. 9 and Fig. 10), assume that we have the following observation: **snd cnode cconn$_2$ crcv**, i.e., we observe, in sequence, the sending of a message, the creation of a new intermediary node with the corresponding connection, the creation of a connection to a receiver and the reception of a corrupted message. As explained above, these four observations can be represented by a simple grammar $\mathcal{A}$ (see Fig. 13) with four 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, W, Z). The start graph is depicted with bold lines, and the left- and right-hand sides of the productions of the occurrence grammar are indicated by using a Petri-net-like notation: productions are drawn with black rectangles connected to the edges they consume or produce by dashed lines.

When unfolding the product of a grammar $\mathcal{G}$ with its observation $\mathcal{A}$, we obtain a grammar $\mathcal{U} = \mathcal{U}(\mathcal{G} \times \mathcal{A})$ with a morphism $\pi\colon \mathcal{U} \to \mathcal{A}$, arising as the image through the unfolding functor of the projection $\mathcal{G} \times \mathcal{A} \to \mathcal{A}$ (since the unfolding of an occurrence grammar is the grammar itself). Now, as grammar morphisms are simulations [10], given the morphism $\pi\colon \mathcal{U} \to \mathcal{A}$ we know that any configuration in $\mathcal{U}$ is mapped to a configuration in $\mathcal{A}$. Say that a configuration in $\mathcal{U}$ is a *full explanation* of $\mathcal{A}$ if it is mapped to the configuration of $\mathcal{A}$ 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 21 (pruning). Let $\pi\colon \mathcal{U} \to \mathcal{A}$ be a grammar morphism from an occurrence grammar $\mathcal{U}$ to an observation $\mathcal{A}$. Then, the *pruning* of π, denoted by $Pr(\pi)$, is the sub-grammar of $\mathcal{U}$ obtained by keeping only the productions in

$$\{q \in \mathcal{P}_{\mathcal{U}} \mid \exists C \in Conf(\mathcal{U})\colon (q \in C \wedge \pi(C) = \mathcal{P}_{\mathcal{A}})\}$$

The next technical lemma shows that applying the pruning operation to a morphism $\pi\colon \mathcal{U} \to \mathcal{A}$ no runs in $\mathcal{U}$ which provide a full explanation of $\mathcal{A}$ are lost.

Lemma 1 (interleavings of a pruned grammar). *Let $\pi\colon \mathcal{U} \to \mathcal{A}$ be a grammar morphism from an occurrence grammar to an observation $\mathcal{A}$ with n productions. Then*

$$Runs^n(Pr(\pi)) = \{r \in Runs(\mathcal{U}) \mid \pi^*(r) \in Runs^n(\mathcal{A})\}.$$

PROOF. We first show that $Runs^n(Pr(\pi)) \subseteq \{r \in Runs(\mathcal{U}) \mid \pi^*(r) \in Runs^n(\mathcal{A})\}$. Take any run $r \in Runs^n(Pr(\pi))$. Then clearly $r \in Runs(\mathcal{U})$ since $Pr(\pi)$ is a subgrammar of $\mathcal{U}$. Furthermore, since π preserves the observable productions, $\pi^*(r)$ must contain exactly n observable productions and thus it belongs to $Runs^n(\mathcal{A})$.

For the other direction take any run $r \in Runs(\mathcal{U})$ such that $\pi^*(r) \in Runs^n(\mathcal{A})$. Then, again, r must contain exactly n observable productions. Furthermore take the configuration C consisting of the productions in r. Since π maps C to the set of all productions of $\mathcal{A}$, none of the productions of r is removed during the pruning phase. Hence $r \in Runs^n(Pr(\pi))$. □

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

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

Since the observability assumption holds, it can be shown that the diagnosis grammar is finite whenever the observation is finite. Roughly, the argument is as follows. Assume that the diagnosis grammar is infinite. Since it is finitely branching it must contain an infinite computation, which, by the observability assumption, would contain infinitely many observable events. However, this cannot be the case since all computations in the diagnosis grammar contain at most as many observable events as the observation.

Example 4. We can compute the product of grammars $\mathcal{M}$ and $\mathcal{A}$ and unfold it. For the sake of clarity Fig. 14 shows only a prefix of the unfolding (The full unfolding is presented in Section 6.) In order to give a compact representation of such prefix we again use a Petri-net-like notation. Edges that are preserved by a production are indicated by read arcs, i.e., dashed arcs without arrowhead that connect an edge and a black rectangle.

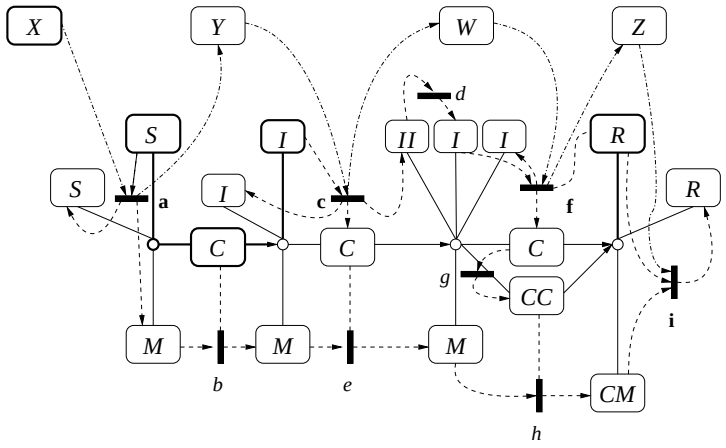
The considered prefix depicts one possible explanation: here the message is sent (event **a**) and crosses the first connection (*b*). Possibly concurrently a new intermediate node and a connection to this node is created (**c**). The new node is initially inactive and it becomes active immediately after (*d*). The new connection is crossed by the message (*e*) and, in a possibly concurrent step, a new connection to the receiver is created (**f**). Such connection is corrupted (*g*), leading to the corruption of the message (*h*) and its reception by the receiver (**i**). *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 have been the case that the new connection was created from the original intermediate node directly to the receiver.
- Events that are not directly related to the failure, such as the corruption of the first or second connection after the message has crossed.

Furthermore there are events belonging to prefixes of the unfolding that cannot be extended to a full observation. For instance, the full unfolding would contain an event corresponding to an (uncorrupted) message crossing the rightmost uncorrupted connection *C*. However, this is a false trail since, after this, it would be impossible to complete the explanation with the reception of a corrupted message (in fact, this would require the sending of a new message, an event which would be visible and inconsistent with the observation). These events belonging only to incomplete explanations are removed from the unfolding in the pruning phase.

We remark 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.



event	label
a	**snd**$_2$
b	*cross*$_1$
c	**cnode**
d	*act*
e	*cross*$_1$
f	**cconn**$_2$
g	*crpt*
h	*cross*$_2$
i	**crcv**

Figure 14: Running example: prefix of the unfolding of the product.

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 1) and by additionally taking care of the pruning phase (Definition 21).

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.

A first lemma shows that the labelled runs of a product of occurrence grammars $\mathcal{O}_1 \times \mathcal{O}_2$ can be obtained by suitably combining pairs of compatible runs of $\mathcal{O}_1$ and $\mathcal{O}_2$. By "compatible" we mean that they admit a common extension, obtained by interleaving the run of $\mathcal{O}_1$ with events labelled in $(\Lambda_1 \cup \Lambda_2) - \Lambda_2$ and, dually, the run of $\mathcal{O}_2$ with events labelled in $(\Lambda_1 \cup \Lambda_2) - \Lambda_1$.

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

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

Proof. Let $\mathcal{I}_1 = \mathit{Ilv}(\mathcal{O}_1)$, $\mathcal{I}_2 = \mathit{Ilv}(\mathcal{O}_2)$ and $\mathcal{I} = \mathcal{I}_1 \times \mathcal{I}_2 = \mathit{Ilv}(\mathcal{O}_1) \times \mathit{Ilv}(\mathcal{O}_2)$. Furthermore let $\pi_i : \mathcal{I} \to \mathcal{I}_i$, $i \in \{1,2\}$ be the projections. We first observe that

$$\begin{aligned}
\lambda^*(\mathcal{I}) = \lambda^*(R_\mathcal{I}) &= \{\lambda_\mathcal{I}^*(r) \mid r \in (E_\mathcal{I})^\odot \ \wedge\ \pi_1^*(r) \in R_1 \ \wedge\ \pi_2^*(r) \in R_2\} \\
&\overset{(\dagger)}{=} \{w \in \Lambda^* \mid f_1^*(w) \in \lambda_1^*(R_1),\ f_2^*(w) \in \lambda_2^*(R_2)\} \\
&= f_1^{-1}(\lambda_1^*(R_1)) \cap f_2^{-1}(\lambda_2^*(R_2)) \\
&= f_1^{-1}(\lambda_1^*(\mathcal{I}_1)) \cap f_2^{-1}(\lambda_2^*(\mathcal{I}_2))
\end{aligned}$$

The equality marked (†) above, which is not obvious, can be shown as follows.

$\subseteq$: Take $r \in R_\mathcal{I}$ and consider the sequence of labels $\lambda_\mathcal{I}^*(r)$ associated with r. Since π_1 is defined exactly on the events whose label is in Λ_1, it holds that $\lambda_1^*(\pi_1^*(r)) = f_1^*(\lambda_\mathcal{I}^*(r))$, hence $f_1^*(\lambda_\mathcal{I}^*(r)) \in \lambda_1^*(R_1)$. Similarly one can show that $f_2^*(\lambda_\mathcal{I}^*(r)) \in \lambda_2^*(R_2)$.

$\supseteq$: Let $w \in \Lambda^*$ be such that $f_1^*(w) \in \lambda_1^*(R_1)$ and $f_2^*(w) \in \lambda_2^*(R_2)$. Assume that $w = c_1 \dots c_k$, $f_1^*(w) = c_{i_1} \dots c_{i_n}$ and $f_2^*(w) = c_{j_1} \dots c_{j_m}$. Furthermore let $r_1 = f_{i_1} \dots f_{i_n} \in R_1$ be a run with $\lambda_1^*(r_1) = f_1^*(w)$ and let $r_2 = g_{j_1} \dots g_{j_m} \in R_2$

be a run with $\lambda_2^*(r_2) = f_2^*(w)$. Now construct a run $r = e_1 \dots e_n$ with

$$e_i = \begin{cases} (f_i, g_i) & \text{if } c_i \in \Lambda_1 \cap \Lambda_2 \\ (f_i, *) & \text{if } c_i \in \Lambda_1 \backslash \Lambda_2 \\ (*, g_i) & \text{if } c_i \in \Lambda_2 \backslash \Lambda_1 \end{cases}$$

It holds that $\pi_1^*(r) = r_1$, $\pi_2^*(r) = r_2$ and obviously $\lambda_I^*(r) = w$. Hence w is contained in the left-hand set.

We next show that $\lambda^*(\mathcal{I}) = \lambda^*(Ilv(O_1 \times O_2))$, and thus we conclude. This follows from Lemma 3 in Appendix A.2 which guarantees the existence of a total projection $\delta \colon Ilv(O_1 \times O_2) \to \mathcal{I}$. In fact, since δ respects labellings and is total, whenever it maps two runs of $Ilv(O_1 \times O_2)$ to one run of $\mathcal{I}$, they must be associated with the same label sequence. Using, additionally, the fact that δ is surjective on runs, the desired equality immediately follows. □

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

Proposition 5. *Let $\mathcal{G}$ be a grammar and $\mathcal{A}$ an observation, where Λ is the set of labels of $\mathcal{G}$ and $\Lambda' \subseteq \Lambda$ the set of labels of $\mathcal{A}$. Furthermore let $f \colon \Lambda \rightharpoonup \Lambda'$ be the obvious partial inclusion. Then it holds that:*

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

Proof. First, recall that, by Proposition 3, $\lambda^*(Runs(\mathcal{G})) = \lambda^*(Ilv(\mathcal{U}(\mathcal{G})))$ and $\lambda^*(Runs(\mathcal{A})) = \lambda^*(Ilv(\mathcal{U}(\mathcal{A}))) = \lambda^*(Ilv(\mathcal{A}))$. Furthermore due to the fact that unfolding is a coreflection (Theorem 1), we have $\mathcal{U}(\mathcal{G} \times \mathcal{A}) = \mathcal{U}(\mathcal{G}) \times \mathcal{U}(\mathcal{A})$.

Hence we have to show that

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

This is a corollary of Lemma 2 for $O_1 = \mathcal{U}(\mathcal{G})$, $O_2 = \mathcal{U}(\mathcal{A}) = \mathcal{A}$, $\Lambda_1 = \Lambda$, $\Lambda_2 = \Lambda'$; furthermore f corresponds to f_2 and f_1 is the identity since $\Lambda' \subseteq \Lambda$. □

We can finally prove that the described diagnosis procedure is complete, i.e., given an observation of size n, the runs of the diagnosis grammar $D(\mathcal{G}, \mathcal{A})$ with 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 5, one could have shown that the same holds replacing the diagnosis grammar with $\mathcal{U}(\mathcal{G} \times \mathcal{A})$, i.e., the unpruned unfolding. The result below additionally shows that no valid explanation is lost during the pruning phase.

Theorem 2 (correctness of the diagnosis). *With the notation of Proposition 5 it holds that:*

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

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.

Proof. By definition

$$\lambda^*(Runs^n(D(\mathcal{G}, \mathcal{A}))) = \lambda^*(Runs^n(Pr(\pi_2))), \tag{1}$$

where, if we let $\mathcal{U} = \mathcal{U}(\mathcal{G} \times \mathcal{A})$, then $\pi_2 \colon \mathcal{U} \to \mathcal{A}$ is the (image through the unfolding functor of the) second projection of the product. By Lemma 1, the set (1) is the same as

$$\lambda^*(\{r \in Runs(\mathcal{U}) \mid \pi^*(r) \in Runs^n(\mathcal{A})\}), \tag{2}$$

We will now show that set (2) coincides with

$$\lambda^*(Runs(\mathcal{G})) \cap f^{-1}(\lambda^*(Runs^n(\mathcal{A}))). \tag{3}$$

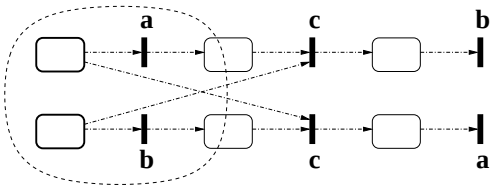


Figure 15: Spurious runs in a diagnosis grammar.

$\subseteq$**:** Let $w = \lambda^*(r)$ for some $r \in \mathit{Runs}(\mathcal{U})$ and $\pi^*(r) \in \mathit{Runs}^n(\mathcal{A})$. By Proposition 3, $w \in \lambda^*(\mathit{Runs}(\mathcal{G}))$. The fact that $w \in f^{-1}(\lambda^*(\mathit{Runs}^n(\mathcal{A})))$ easily follows by observing that $\lambda^*(\pi_2^*(r)) \in \lambda^*(\mathit{Runs}^n(\mathcal{A}))$ is a subsequence of w where all unobservable labels are missing and these labels, by definition, can be reinserted by f^{-1}. This proves the inclusion (2) $\subseteq$ (3).

$\supseteq$**:** Let $w \in \lambda^*(\mathit{Runs}(\mathcal{G})) \cap f^{-1}(\lambda^*(\mathit{Runs}^n(\mathcal{A})))$. Observe that

$$\begin{aligned} & \lambda^*(\mathit{Runs}(\mathcal{G})) \cap f^{-1}(\lambda^*(\mathit{Runs}^n(\mathcal{A}))) \\ \subseteq\ & \lambda^*(\mathit{Runs}(\mathcal{G})) \cap f^{-1}(\lambda^*(\mathit{Runs}(\mathcal{A}))) \\ =\ & \lambda^*(\mathit{Ilv}(\mathcal{U})) \qquad \text{(Proposition 5)} \end{aligned}$$

Hence there exists a run $r \in \mathit{Runs}(\mathcal{U})$ such that $\lambda^*(r) = w$. By definition of a morphism, $\pi_2^*(r) \in \mathit{Runs}(\mathcal{A})$ and $\lambda^*(\pi_2^*(r)) = f^*(\lambda^*(r)) = f^*(w) \in \lambda^*(\mathit{Runs}^n(\mathcal{A}))$ (by choice of w). Hence also $\pi_2^*(r) \in \mathit{Runs}^n(\mathcal{A})$.

Summing up $r \in \{r \in \mathit{Runs}(\mathcal{U}) \mid \pi_2(r) \in \mathit{Runs}^n(\mathcal{A})\}$ and since $\lambda^*(r) = w$ we conclude $w \in \lambda^*(\pi_2^{-1}(\mathit{Runs}^n(\mathcal{A})))$.

□

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 explanation 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. 15, 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, since each event is part of a chain consisting of **a**, **b**, **c** which fully explains the observation. However, since different explanations can interfere, there is a configuration (indicated by the dashed closed line) that cannot be further extended to an explanation.

6. Experimental Evaluation

In order to give an idea about the practical applicability of our approach, we use some existing tools in order to compare the size of the unfolding with that of interleaving-based models which could be used for analysing the running example (message passing over a network).

Graph grammars are unfolded by using an extension of the tool AUGUR [20], whose original purpose is to compute approximated unfoldings in order to abstract infinite-state graph transformation systems. The extension, under the assumption that rules do not delete nodes, can also be used to produce an ordinary, non-approximated, unfolding of a graph grammar. In our running example this assumption is violated only by rule **fail**, which is however not involved in the specific observation that we are considering and thus can be safely omitted. We have not yet implemented the computation of the product of two graph grammars (this is done manually) and pruning.

We took the running example grammar $\mathcal{M}$ (see Figures 9 and 10) and computed the product of this grammar and the observation grammar for the sequence **snd cnode cconn**$_2$ **crcv**. The unfolding is shown in Figures 16 and 17, visualized by using GraphViz.[1] For reasons of clarity the output of the tool, i.e. the unfolding, is split into two

[1] http://www.graphviz.org/

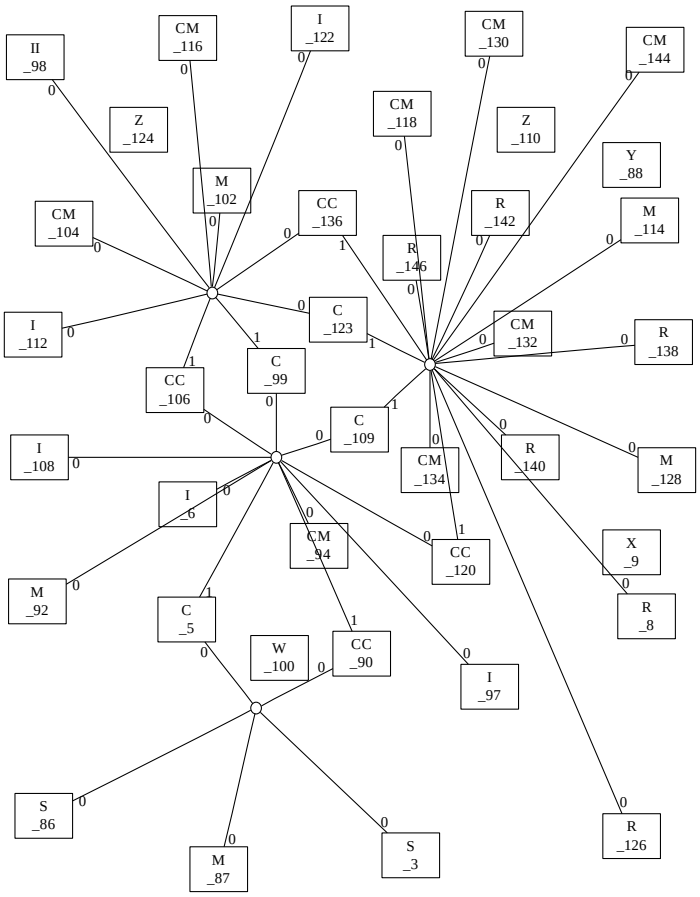


Figure 16: Type graph of the unfolding of the product.

components: the underlying type graph and a Petri net which represents the rules, depicting deletion, preservation and creation of the edges of the type graph.

The unfolding includes 40 edges, 4 nodes and 26 transitions (a prefix of this unfolding was already presented in Figure 14). The unfolding shown here is still unpruned. In the pruned version the transitions labelled `cross1_115` and `cross1_129` would be removed. They correspond to cases where an uncorrupted message arrives at the receiver, thus making the last observation (**crcv**) impossible.

The five transitions labelled **crcv** represent five distinct situations: either the message has passed from the sender to the original intermediate node cirectly to the receiver and the corruption of the message has been caused by the first connection (`crcv_127`) or by the second connection (`crcv_139`). Or the message has passed from the sender to the original intermediate node to a new intermediate node to the receiver and its corruption has been caused by the first (`crcv_143`), second (`crcv_141`) or third (`crcv_147`) connection. Note that the last event (`crcv_147`) corresponds to event **i** in Figure 14.

In order to get an idea of the cost of a diagnosis algorithm based on interleavings, we can compute the product of the transition system of $\mathcal{M}$ with an automaton representing the observation. The size of the resulting transition system is the same as the size of the state space of the product grammar. We determined the corresponding number of states (taken up to isomorphisms) by using Groove[2], a tool for state space exploration and model-checking of graph transformation systems [21]. The state space consists of 205 states and it is not straightforward to deduce manually the diagnosis information from the corresponding transition system.

[2] `http://groove.cs.utwente.nl/groove-home/`

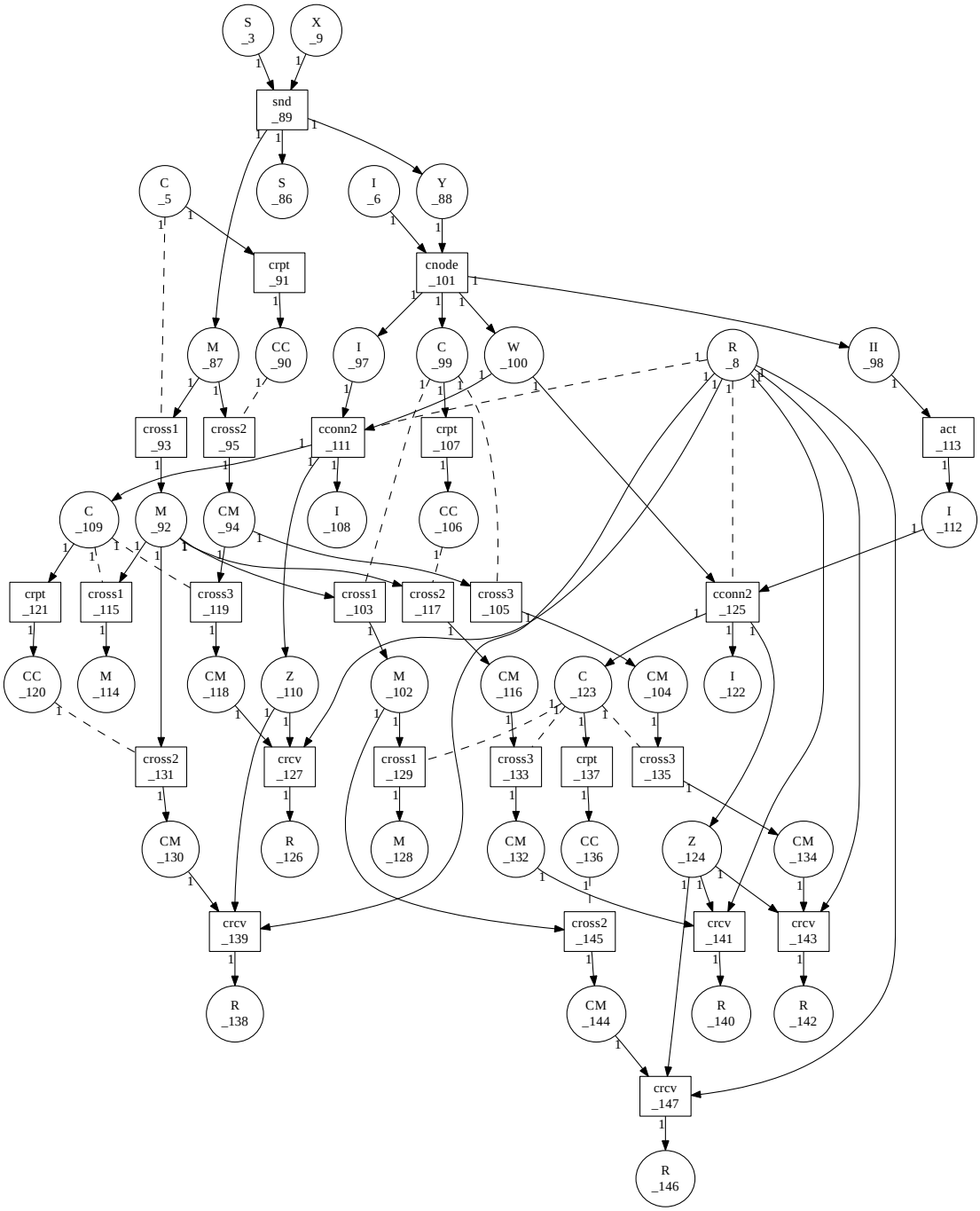


Figure 17: Petri net underlying the unfolding of the product.

Since this example is still fairly small we took the same grammar $\mathcal{M}$ and considered a slightly longer observation sequence: **snd cnode cconn2 cnode crcv**. In this case, the unfolding consists of 83 edges, 8 vertices and 57 transitions, whereas the state space contains 1338 states. In general, for highly concurrent systems, the unfolding tends to be exponentially smaller than the interleaving model; this blow up in the size will decrease for systems where the degree of concurrency is limited.

7. 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 transformation systems over hypergraphs, can be generalised to the more abstract setting of adhesive categories. In particular we have developed a generalization of the unfolding procedure [22] that works for SPO-rewriting in (suitable variations of) adhesive categories [16]. 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, e.g., rewriting graphs with scopes, graphs with second-order edges, and other kind of graph structures (which for instance occur in the various UML diagrams).

We are also interested in distributed diagnosis where every observer separately computes possible explanations of local observations that however have to be synchronized. In [18] 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. Pullbacks are needed since we want to describe system composition via a common interface.

Acknoledgements. We are very grateful to the anonymous referees for their valuable comments on the preliminary version of the paper.

Appendix A. Auxiliary Material

In this appendix we first briefly recap the functorial event structure semantics for graph transformation systems, as defined in [10] and adapt them to the labelled setting. The semantics is given in terms of *asymmetric event structures* (AES's) [17], a generalization of prime event structures where the conflict relation is allowed to be non-symmetric. A functor mapping any occurrence grammar into an AES is defined. The event structure semantics of a graph grammar is obtained by taking the AES associated to the unfolding of the grammar.

Then, using the characterisation of the AES's semantics as a right adjoint, we prove a property of the product of interleaving structures which is fundamental for the theory in the paper.

Appendix A.1. From occurrence grammars to asymmetric event structures

For technical reasons we first introduce pre-asymmetric event structures. Then asymmetric event structures will be defined as special pre-asymmetric event structures satisfying a suitable condition of "saturation".

Definition 23 (asymmetric event structure). A *pre-asymmetric event structure (pre-*AES*)* is a tuple $\mathcal{A} = \langle E, \leq, \nearrow, \Lambda, \lambda \rangle$, where E is a set of *events*, $\leq$, $\nearrow$ are binary relations on E called *causality* and *asymmetric conflict*, respectively, and $\lambda : P \to \Lambda$ is a labelling over the set of labels Λ, such that:

1. causality $\leq$ is a partial order and $\lfloor e \rfloor = \{e' \in E \mid e' \leq e\}$ is finite for all $e \in E$;
2. asymmetric conflict $\nearrow$ satisfies, for all $e, e' \in E$:
   (a) $e < e' \quad \Rightarrow \quad e \nearrow e'$,
   (b) $\nearrow$ is acyclic in $\lfloor e \rfloor$,

where, as usual, $e < e'$ means $e \leq e'$ and $e \neq e'$.

An *asymmetric event structure* (AES) is a pre-AES which additionally satisfies:

3. for any $e, e' \in E$, if $\nearrow$ is cyclic in $\lfloor e \rfloor \cup \lfloor e' \rfloor$ then $e \nearrow e'$ (and also $e' \nearrow e$).

Conditions 1 and 2 are easily understandable in the light of the analogous properties of causality and asymmetric conflict in occurrence grammars (see Definition 14). As cycles of asymmetric conflict play the role of conflicts in this setting, Condition 3 requires that conflicts in an AES are inherited through causality.

It can be shown easily that any pre-AES can be "saturated" to produce an AES. More precisely, given a pre-AES $\mathcal{A} = \langle E, \leq, \nearrow, \Lambda, \lambda \rangle$, its saturation, denoted by $\overline{\mathcal{A}}$, is the AES $\langle E, \leq, \nearrow', \Lambda, \lambda \rangle$, where $\nearrow'$ is defined as $e \nearrow' e'$ iff $(e \nearrow e')$ or $\nearrow$ is cyclic in $\lfloor e \rfloor \cup \lfloor e' \rfloor$.

Definition 24 (category of AESs). Let $\mathcal{A}_0$ and $\mathcal{A}_1$ be two AES's such that $\Lambda_1 \subseteq \Lambda_0$. An AES-*morphism* $f : \mathcal{A}_0 \to \mathcal{A}_1$ is a partial function $f : E_0 \rightharpoonup E_1$ such that,

1. for all $e_0 \in E_0$, $f(e_0)\downarrow$ iff $\lambda_0(e_0) \in \Lambda_1$ and, in this case, $\lambda_1(f(e_0)) = \lambda_0(e_0)$;

and for all $e_0, e_0' \in E_0$, assuming that $f(e_0)\downarrow$ and $f(e_0')\downarrow$,

2. $\lfloor f(e_0) \rfloor \subseteq f(\lfloor e_0 \rfloor)$;
3. (a) $f(e_0) \nearrow_1 f(e_0') \quad \Rightarrow \quad e_0 \nearrow_0 e_0'$;
   (b) $(f(e_0) = f(e_0')) \wedge (e_0 \neq e_0') \quad \Rightarrow \quad e_0 \nearrow_0 e_0'$.

We denote by **AES** the category having asymmetric event structures as objects and AES-morphisms as arrows.

The notion of configuration for AES's is completely analogous to that of occurrence grammars.

Definition 25 (configurations). Let A be an AES. A *configuration* of A is a set of events $C \subseteq E$ such that

1. for any $e \in C$ it holds $\lfloor e \rfloor \subseteq C$;
2. $\nearrow_C$, the asymmetric conflict restricted to C, is acyclic and finitary.

Given any occurrence grammar, the corresponding asymmetric event structure is readily obtained by taking the production names as events. Causality and asymmetric conflict are the relations defined in Definitions 12 and 13.

Definition 26 (AES for an occurrence grammar). Let $O = \langle T, G_s, P, \pi, \Lambda, \lambda \rangle$ be an occurrence grammar. The AES associated to O, denoted $\mathcal{E}_s(O)$, is the saturation of the pre-AES $\langle P, \leq, \nearrow, \Lambda, \lambda \rangle$, with $\leq$ and $\nearrow$ as in Definitions 12 and 13.

The above construction naturally gives rise to a functor, which is a right adjoint. The following theorem is adapted from a result of [10] which can straightforwardly be extended to labelled grammar morphisms.

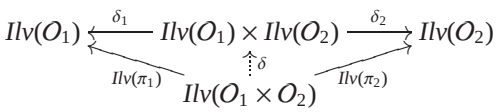
Theorem 3 (coreflection). *For any occurrence grammar morphism $h : O_0 \to O_1$ let $\mathcal{E}_s(h)(q) = h_P(q)$ if $h_P(q) \neq \emptyset$ and $\mathcal{E}_s(h)(q) \uparrow$, otherwise. Then $\mathcal{E}_s$: **OGG** $\to$ **AES** is a well-defined functor, which is a right adjoint.*

As a right adjoint $\mathcal{E}_s$ preserves limits, specifically it preserves products.

Appendix A.2. A property of the product of interleaving structures

Lemma 3. *Let O_1, O_2 be two occurrence grammars. Consider the product of the interleaving structures $Ilv(O_1)$, $Ilv(O_2)$ and the image through the Ilv functor of the product $O_1 \times O_2$ in* **OGG** *as shown below.*

Then the mediating morphism δ is a projection which is total on events.



Proof. In order to prove that δ is a p¡rojection, consider any run $r = e_1 \dots e_n$ in $Ilv(O_1) \times Ilv(O_2)$. We have to prove that there exists a run r' of $Ilv(O_1 \times O_2)$ such that $\delta^*(r') = r$.

By definition of interleaving structure morphisms, $r_i = \delta_i^*(r)$ is a run in $Ilv(O_i)$, for $i \in \{1, 2\}$. Hence, by Proposition 2, the set C_i of events which occur in r_i is a configuration $C_i \in Conf(O_i)$ and r_i is a linearisation of C_i compatible with the asymmetric conflict relation $\nearrow_i$ in O_i, for $i \in \{1, 2\}$

Consider the AES $\mathcal{E}_s(O_i)$ underlying grammar O_i for $i \in \{1, 2\}$. Note that functor $\mathcal{E}_s$ is a right adjoint by Theorem 3 and thus it preserves limits and, in particular products. Hence $\mathcal{E}_s(O_1 \times O_2) = \mathcal{E}_s(O_1) \times \mathcal{E}_s(O_2)$, i.e.,

$$\mathcal{E}_s(O_1) \xleftarrow{\mathcal{E}_s(\pi_1)} \mathcal{E}_s(O_1 \times O_2) \xrightarrow{\mathcal{E}_s(\pi_2)} \mathcal{E}_s(O_2)$$

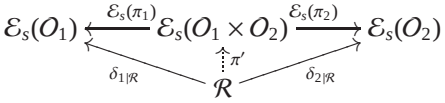
is a product diagram in **AES**.

Define an AES corresponding to the run r, i.e.,

$$\mathcal{R} = \langle \{e_1, \dots, e_n\}, \leq_{\mathcal{R}}, \nearrow_{\mathcal{R}}, \Lambda_{\mathcal{R}}, \lambda_{\mathcal{R}} \rangle,$$

where $<_{\mathcal{R}}$ is defined by $e_i <_{\mathcal{R}} e_{i+1}$ for $i \in \{1, \dots, n-1\}$ and $\leq_{\mathcal{R}} = (<_{\mathcal{R}})^*$. Moreover $\nearrow_{\mathcal{R}} = <_{\mathcal{R}}$, $\Lambda_{\mathcal{R}} = \Lambda_1 \cup \Lambda_2$ and $\lambda_{\mathcal{R}}$ is the restriction of the labelling in $Ilv(O_1) \times Ilv(O_2)$.

It is not difficult to see that $\delta_{i|\mathcal{R}} : \mathcal{R} \to \mathcal{E}_s(O_i)$, the restriction of δ_i to $\{e_1, \dots, e_n\}$, is a well-defined AES morphism for $i \in \{1, 2\}$. Therefore we deduce the existence of a unique mediating morphism $\pi' : \mathcal{R} \to \mathcal{E}_s(O_1 \times O_2)$ as shown in the diagram below



Recalling that AES-morphisms map configurations into configurations (see [17, Lemma 3.6]), we have that $C' = \{\pi'(e_1), \dots, \pi'(e_n)\}$ is a configuration in $\mathcal{E}_s(O_1 \times O_2)$ and it is immediate to see that $r' = \pi'^*(r)$ is a linearisation of C' compatible with asymmetric conflict. Since for any occurrence grammar O, we have $Conf(O) = Conf(\mathcal{E}_s(O))$, we deduce that $C' \in Conf(O_1 \times O_2)$ and

$$r' \in Ilv(O_1 \times O_2).$$

Additionally, by commutativity of the diagram above, $\mathcal{E}_s(\pi_i)^*(r') = \delta_{i|\mathcal{R}}^*(r) = \delta_i^*(r) = r_i$ for $i \in \{1, 2\}$ and thus, since both $\mathcal{E}_s$ and Ilv when applied to a grammar morphism leave the production mapping unchanged, for $i \in \{1, 2\}$ we have

$$Ilv(\pi_i)^*(r') = r_i$$

Coming back to the diagram in the statement of the lemma, by commutativity, for $i \in \{1, 2\}$ we have:

$$\delta_i^*(\delta^*(r')) = Ilv(\pi_i)^*(r') = r_i$$

From this, recalling how the product of interleaving structures is characterised (Proposition 4), we deduce that $\delta^*(r') = r$, as desired.

In order to prove that δ is total it suffices to observe that, for any event e in $Ilv(O_1 \times O_2)$ (and hence in $O_1 \times O_2$), either $Ilv(\pi_1(e))$ or $Ilv(\pi_2)(e)$ are defined. This immediately follows by from the fact that, by Proposition 1, either $\pi_1(e) \neq \emptyset$ or $\pi_2(e) \neq \emptyset$. Now, this implies that $\delta(e)$ must be defined, since otherwise either $\delta_1 \circ \delta = \pi_1$ or $\delta_2 \circ \delta = \pi_2$ would not hold. □

References

[1] C. Cassandras, S. Lafortune, Introduction to Discrete Event Systems, Kluwer Academic Publishers, 1999.

[2] M. Sampath, R. Sengupta, K. Sinnamohideen, S. Lafortune, D. Teneketzis, Failure diagnosis using discrete event models, IEEE Transaction on Control System and Technology 4 (2) (1996) 105–124.

[3] Y. Wang, T.-S. Yoo, S. Lafortune, Diagnosis of discrete event systems using decentralized architectures, Discrete Event Dynamic Systems 17 (2) (2007) 233–263.

[4] E. Fabre, A. Benveniste, S. Haar, C. Jard, Distributed monitoring of concurrent and asynchronous systems, Journal of Discrete Event Dynamic Systems 15 (1) (2005) 33–84.
[5] A. Benveniste, E. Fabre, S. Haar, C. Jard, Diagnosis of asynchronous discrete event systems, a net unfolding approach, IEEE Transactions on Automatic Control 48 (5) (2003) 714–727.
[6] T. Chatain, C. Jard, Models for the supervision of web services orchestration with dynamic changes, in: AICT/SAPIR/ELETE'05, IEEE, 2005, pp. 446–451.
[7] R. Bruni, H. C. Melgratti, Non-sequential behaviour of dynamic nets, in: S. Donatelli, P. S. Thiagarajan (Eds.), Proceedings of ICATPN'06, Vol. 4024 of LNCS, Springer, 2006, pp. 105–124.
[8] P. Baldan, T. Chatain, S. Haar, B. König, Unfolding-based diagnosis of systems with an evolving topology, in: F. van Breugel, M. Chechik (Eds.), Proceedings of CONCUR'08, Vol. 5201 of LNCS, Springer, 2008, pp. 203–217.
[9] M. Löwe, Algebraic approach to single-pushout graph transformation, Theoretical Computer Science 109 (1993) 181–224.
[10] P. Baldan, A. Corradini, U. Montanari, L. Ribeiro, Unfolding semantics of graph transformation, Information and Computation 205 (2007) 733–782.
[11] A. Corradini, U. Montanari, F. Rossi, Graph processes, Fundamenta Informaticae 26 (1996) 241–265.
[12] M. Löwe, M. Korff, A. Wagner, An Algebraic Framework for the Transformation of Attributed Graphs, in: M. Sleep, M. Plasmeijer, M. van Eekelen (Eds.), Term Graph Rewriting: Theory and Practice, Wiley, 1993, pp. 185–199.
[13] R. Bruni, F. Gadducci, Some algebraic laws for spans (and their connections with multirelations), in: W. Kahl, D. Parnas, G. Schmidt (Eds.), Proceedings of RelMiS'01, Vol. 44 of ENTCS, 2001, also published as Technical Report, Bericht Nr. 2001-02, Fakultät für Informatik, Universität der Bundeswehr München.
[14] G. Winskel, Event Structures, in: W. Brauer, W. Reisig, G. Rozenberg (Eds.), Petri Nets: Applications and Relationships to Other Models of Concurrency, Vol. 255 of LNCS, Springer, 1987, pp. 325–392.
[15] G. Winskel, Petri nets, algebras, morphisms, and compositionality, Information and Computation 72 (3) (1987) 197–238.
[16] S. Lack, P. Sobociński, Adhesive and quasiadhesive categories, RAIRO – Theoretical Informatics and Applications 39 (3) (2005) 511–545.
[17] P. Baldan, A. Corradini, U. Montanari, Contextual Petri nets, asymmetric event structures and processes, Information and Computation 171 (1) (2001) 1–49.
[18] P. Baldan, S. Haar, B. König, Distributed unfolding of petri nets, in: W. Aceto, A. Ingólfsdóttir (Eds.), Proceedings of FoSSaCS '06, Vol. 3921 of LNCS, Springer, 2006, pp. 126–141.
[19] S. Haar, A. Benveniste, E. Fabre, C. Jard, Partial order diagnosability of discrete event systems using Petri net unfoldings, in: Proceedings of 42nd Conf. on Decision and Control, Vol. 4, IEEE, 2003, pp. 3748–3753.
[20] B. König, V. Kozioura, Augur 2—a new version of a tool for the analysis of graph transformation systems, in: Proceedings of GT-VMT '06 (Workshop on Graph Transformation and Visual Modeling Techniques), Vol. 211 of ENTCS, 2008, pp. 201–210.
[21] A. Rensink, The GROOVE simulator: A tool for state space generation, in: J. L. Pfaltz, M. Nagl, B. Böhlen (Eds.), Proceedings of AGTIVE'03, Vol. 3062 of LNCS, Springer, 2004, pp. 479–485.
[22] P. Baldan, A. Corradini, T. Heindel, B. König, P. Sobocinski, Unfolding grammars in adhesive categories, in: M. Lenisa, A. Kurz, A. Tarlecki (Eds.), Proceedings of CALCO'09, Vol. 5728 of LNCS, Springer, 2009, pp. 350–366.