Under consideration for publication in Math. Struct. in Comp. Science

Processes and Unfoldings: Concurrent Computations in Adhesive Categories

PAOLO BALDAN[1] and ANDREA CORRADINI[2] and TOBIAS HEINDEL[3] and BARBARA KÖNIG[4] and PAWEŁ SOBOCIŃSKI[5]

[1]*Dipartimento di Matematica Pura e Applicata, Università di Padova, Italy*
[2]*Dipartimento di Informatica, Università di Pisa, Italy*
[3]*Institut CARNOT, CEA LIST, Gif sur Yvette, France*
[4]*Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen, Germany*
[5]*ECS, University of Southampton, United Kingdom*

Received 21 February 2011; Revised 9 January 2012

We generalise both the notion of non-sequential process and the unfolding construction (previously developed for concrete formalisms such as Petri nets and graph grammars) to the abstract setting of (single pushout) rewriting of objects in adhesive categories. The main results show that processes are in one-to-one correspondence with switch-equivalent classes of derivations, and that the unfolding construction can be characterised as a coreflection, i.e., the unfolding functor arises as the right adjoint to the embedding of the category of occurrence grammars into the category of grammars. As the unfolding represents potentially infinite computations, we need to work in adhesive categories with "well-behaved" colimits of ω-chains of monos. Compared to previous work on the unfolding of Petri nets and graph grammars, our results apply to a wider class of systems, which is due to the use of a refined notion of grammar morphism.

1. Introduction

When modelling concurrent or distributed systems one often needs a specification formalism that is able to describe the intrinsic parallelism of the system, as well as a semantics providing explicit information concerning causality, conflict and concurrency of events in computations. This is clearly the case if one wants to understand and investigate the inherent concurrency of a given system, but truly concurrent models are also a cornerstone of verification techniques based on partial order methods (McMillan, 1993). In fact, the latter avoid the enumeration of all possible interleavings of events, and in this way, especially in the case of highly concurrent systems, yield compact descriptions of the behaviour of a system.

If the specification formalism itself is expressive enough to provide an explicit account of concurrency and non-determinism, then the same formalism can be used, with suitable precautions, as the semantic domain as well. This is for instance the case for Petri nets: it is well-known that a concurrent computation of a net can be represented faithfully with a *deterministic occurrence net*, i.e., a net satisfying suitable acyclicity and safety constraints, leading to the notion of net *process* (Goltz and Reisig, 1983). Additionally, an *unfolding* construction has been defined which "unravels" a net from a starting state and produces an occurrence net which fully describes its concurrent behaviour, including all reachable states and the mutual dependencies of all possible steps (Winskel, 1987a).

The characterization of concurrent computations as processes and the unfolding construction have been generalised along the years from Petri nets to more expressive rule based specification formalisms, including contextual nets (Vogler, Semenov and Yakovlev, 1998; Baldan, Corradini and Montanari, 2001) and graph transformation systems (Corradini, Montanari and Rossi, 1996; Baldan, Corradini, Montanari and Ribeiro, 2007). But there are many types of graph transformation formalisms, based on undirected and directed graphs, hypergraphs, graphs with scopes, graphs with second-order edges, and so forth. With the introduction of adhesive categories (Lack and Sobociński, 2005), an abstract and unifying framework for the definitions and analysis of this kind of systems has been established, since all of them can be considered as rule-based systems acting on an adhesive category.

As a consequence more abstract definitions of processes and unfoldings are called for, which, unlike the previous proposals, do not assume any knowledge about the internal structure of the objects that are transformed (such as multisets of places, graphs of various kinds, etc.), but only exploit the properties that such objects enjoy, axiomatised by the laws of adhesive categories. This is indeed the contribution of the present paper, which summarises and extends the conference papers (Baldan, Corradini, Heindel, König and Sobociński, 2006; Baldan, Corradini, Heindel, König and Sobocinski, 2009) and presents central results of (Heindel, 2009).

Following the line of research of (Ehrig, Habel, Kreowski and Parisi-Presicce, 1991; Lack and Sobociński, 2005), we shall regard system states as objects of a category $\mathbf{C}$ satisfying suitable properties. Part of the properties of $\mathbf{C}$ ensure a meaningful notion of $\mathbf{C}$-object rewriting, while other additional properties are required to guarantee, first, that the construction of processes and the unfolding procedure are viable and, second, that the unfolding can be characterised as a coreflection in the style of (Winskel, 1987a). The latter result guarantees the compositionality of the unfolding construction with respect to operations on grammars expressed as limits, as exploited for example in the field of model-based diagnosis (Benveniste, Fabre, Haar and Jard, 2003; Baldan, Chatain, Haar and König, 2010).

The approach to rewriting that we shall use is the *single pushout approach* (SPO) (Löwe, 1993). This is one of the most commonly used *algebraic approaches* to rewriting, alternative to the *double pushout* (DPO) approach (Ehrig, Pfender and Schneider, 1973), where some subtle complications due to the implicit negative application conditions of DPO rewriting are avoided. As a categorical framework, as mentioned above, we consider adhesive categories, which turn out to be appropriate for SPO rewriting as the needed pushouts in the partial map category Par($\mathbf{C}$) can be shown to exist. In addition, adhesivity is sufficient to build the process of a finite derivation as a colimit of the corresponding diagram, as well as for constructing the finite prefixes of the unfolding. Then a crucial step consists in joining these prefixes into a single structure. To ensure that this is possible, we need that colimits of ω-chains of monos exist and satisfy suitable properties. Adhesive categories having sufficiently well-behaved colimits of ω-chains of monos will be called *ω-adhesive* (see also (Heindel and Sobociński, 2009; Cockett and Guo, 2007)).

The main results of the paper state that there is a one-to-one correspondence between processes and switch-equivalence classes of derivations, and that the unfolding construction induces a *coreflection*, i.e., it can be expressed as a functor that is right adjoint to the embedding of the category of occurrence grammars, the category where unfoldings live, into the category of all

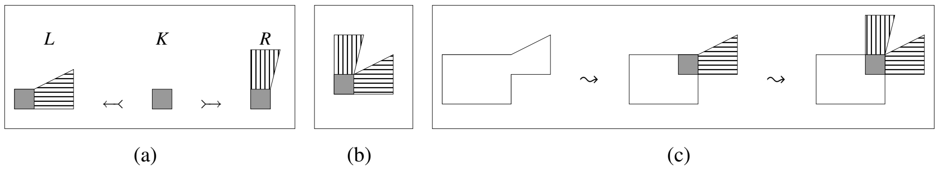


Fig. 1. (a) Rule as a partial map; (b) Combined rule representation; (c) Schematic representation of an unfolding step

grammars. In order to define the category of grammars we introduce an original notion of grammar morphism which is similar to the graph grammar morphisms proposed in (Baldan et al., 2007) but more concrete; as a consequence, we can treat uniformly the whole class of grammars, without the need to restrict to so-called *semi-weighted* grammars as it was done in several approaches for Petri nets (see, e.g., (Meseguer, Montanari and Sassone, 1997)) and for graph grammars (Baldan et al., 2007).

Roadmap: In order to motivate at a more intuitive level the definitions and constructions that will follow, we first sketch the general ideas of our work. We work in a setting of abstract objects (which could be sets, multisets, graphs, etc.) that are rewritten according to a rule by removing (the image of) its left-hand side and by gluing its right-hand side to the remaining object. According to the SPO approach, the left- and right-hand sides of a rule are related by a partial map, i.e., a span $L \leftarrowtail K \to R$ in the underlying category where the left leg is a mono. As it is usually done in unfolding approaches we restrict to linear rules where both legs are mono; for a schematic representation see Figure 1(a).

A rule essentially indicates what is deleted (▤), what is preserved (■) and what is created (▥). This can either be represented by a span as in Figure 1(a) or by a combined representation (see Figure 1(b)). Very roughly, given a (linear) rule $L \leftarrowtail K \rightarrowtail R$, which is the generalization of a pair of inclusions $L \supseteq K \subseteq R$, an object G that contains the left-hand side L is rewritten to the counterpart of the set $G \backslash (L \backslash K) \cup R$. This however is properly defined only if a suitable complementation operation can be defined.

Already in the case of graphs, complementation is problematic if the so-called dangling condition is not satisfied, i.e., it is unclear what happens if a node is to be removed, and this node is attached to an edge that is not explicitly deleted. There are two ways to resolve this issue: the DPO solution which forbids the rewriting step, and the SPO solution which removes the edge; this is the essential difference between the two approaches. In the present paper we follow the latter path, which, as we shall discuss, amounts to defining the term $G \backslash (L \backslash K)$ using the more general construction of *relative pseudo-complements*, known from lattice theory. Previously, in (Baldan et al., 2006) we studied a process semantics for DPO rewriting: the inhibiting effects arising in this approach required some extra conditions in the notion of a process. These conditions could be adapted also to the unfolding semantics, but at the price of some complications in the theory that we have chosen to avoid in the present paper.

The unfolding of a given grammar with a fixed start object, which is the counterpart of a Petri net with an initial marking, provides a partial order representation of all derivations from the start object. Intuitively the construction works as follows: look for an occurrence of a left-hand side of a rule and, instead of replacing it, attach the right-hand side as in Figure 1(c) and record the occurrence of the rule. Doing this iteratively one obtains a growing object, called a type object, which is possibly infinite, and a set of rules that describe the dependencies on this type object.

Now, in order to characterise the unfolding construction abstractly and to show its universality, we shall need the following concepts:

— *A category with properties that ensure good behaviour of* SPO *rewriting:* For this we shall use adhesive categories which can be used for defining abstractly a notion of rewriting which enjoys suitable Church-Rosser properties (Section 2).
— *An analogue of occurrence nets:* The processes and the unfolding of a Petri net are a special kind of nets, called occurrence nets; similarly, the processes and the structure produced by the unfolding construction in this abstract setting will be a special kind of grammar, which will be called occurrence grammar. In occurrence grammars suitable notions of causality, concurrency and conflict can be defined, allowing for a "static" characterization of reachable states as concurrent objects (Section 3).
— *A category of grammars and grammar morphisms:* We introduce a category of grammars, defining a suitable notion of grammar morphisms which establish a simulation between the source and target grammars (Section 4).
— *Processes:* The process of a derivation is an occurrence grammar, obtained as the colimit of the derivation diagram or by a related inductive construction, with a morphims back to the original grammar. As a main result we show that processes are in one-to-one correspondence with shift-equivalence classes of derivations (Section 5).
— *The unfolding construction:* We show how to construct incrementally larger and larger finite prefixes of the unfolding by taking suitable colimits. Such prefixes are occurrence nets equipped with a morphism back to the original grammar (Section 6).
— *Well-behaved ω-colimits:* In order to construct potentially infinite unfoldings, we have to be able to glue together a countable chain of finite prefixes of the unfolding. To this aim we require that colimits of ω-chains exist and are well-behaved: adhesive categories enjoying this property are called ω-adhesive. The notion of ω-adhesivity is a natural extension of adhesivity that has several closure properties (Section 7).
— *The coreflection result:* Finally we shall present a coreflection result, i.e., we shall show that the unfolding is in a sense the "best" approximation of the original grammar in the realm of occurrence grammars (Section 7).

We illustrate the theory with two running examples: a simple graph transformation system and a Petri net. We deliberately keep the examples simple. Note however, that due to the generality of our results we could just as well apply the theory to more complex graph-like structures, such as graphs with second-order edges or graphs with scopes.

2. Preliminaries

We assume familiarity with some basic notions of category theory, including limits, colimits and natural transformations. We use boldface font for categories, such as $\mathbf{C}, \mathbf{X}$. For each category $\mathbf{C}$, we denote the collection of its objects by $\mathrm{ob}(\mathbf{C})$ and we write $A \in \mathbf{C}$ if $A \in \mathrm{ob}(\mathbf{C})$; given two objects $A, B \in \mathbf{C}$, the collection of morphisms with domain A and codomain B is denoted by

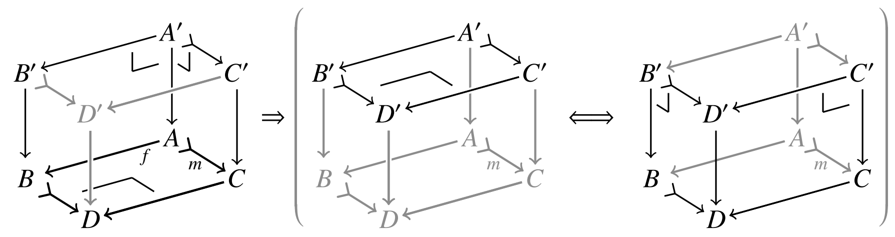
$\mathbf{C}(A, B)$ and we write $f\colon A \to B$ in $\mathbf{C}$ if $f \in \mathbf{C}(A, B)$. We use a calligraphic font for functors and write $\mathcal{F}\colon \mathbf{C} \to \mathbf{D}$ if $\mathcal{F}$ is a functor from $\mathbf{C}$ to $\mathbf{D}$. We use greek letters for natural transformations between functors and thus $\tau\colon \mathcal{F} \stackrel{.}{\to} \mathcal{G}$ is a natural transformation between two functors $\mathcal{F}$ and $\mathcal{G}$. Finally, given two categories $\mathbf{C}$ and $\mathbf{D}$, the category of functors and natural transformations among them is denoted by $[\mathbf{C}, \mathbf{D}]$.

2.1. *Adhesive Categories as a Framework for Single-Pushout Rewriting*

Rewriting objects of adhesive categories has become a standard in the theory of graph transformation, and since the original definition several weaker variants of adhesivity have been proposed in the literature (e.g., (Lack and Sobociński, 2005; Ehrig, Ehrig, Prange and Taentzer, 2006; Heindel, 2009; Heindel, 2010)). For an analysis of the differences among these variants see (Heindel, 2009; Ehrig, Golas and Hermann, 2010). In this paper we consider plain adhesivity only to keep the presentation simpler, but the results we present have been generalised to larger classes of categories in (Heindel, 2009).

Definition 2.1 (Adhesive Category). A category is *adhesive* if

1 it has all pullbacks;
2 pushouts along monos exist, i.e., for each span $B \xleftarrow{f} A \xrightarrow{m} C$ with monic m, a pushout $B \xrightarrow{n} D \xleftarrow{g} C$ exists, which completes the span to a *pushout square* $\begin{smallmatrix} B & \leftarrow & A \\ \downarrow & & \downarrow \\ D & \leftarrow & C \end{smallmatrix}$;
3 all pushouts along monos are *Van Kampen squares*: for every commuting cube diagram as shown below on the left with: (i) m monic, (ii) the bottom square a pushout and (iii) the back squares pullbacks, we have that the top square is a pushout iff the front squares are pullbacks.



The paradigmatic example of an adhesive category is the category of graphs and graph morphisms; however there are several other well-known examples.

Example 2.2 (Examples of Adhesive and Non-Adhesive Categories). It is known that every topos is adhesive (Lack and Sobociński, 2006), thus in particular **Set** is adhesive. Futhermore, adesive categories enjoy several closure properties (Lack and Sobociński, 2005): if $\mathbf{C}$ and $\mathbf{C}'$ are adhesive, $\mathbf{X}$ is a category and T is an object of $\mathbf{C}$, then the product category $\mathbf{C} \times \mathbf{C}'$, the functor category $[\mathbf{X}, \mathbf{C}]$, the slice category $\mathbf{C}{\downarrow}T$ (see below) and the co-slice category $T{\downarrow}\mathbf{C}$ are all adhesive. Therefore every presheaf $[\mathbf{X}, \mathbf{Set}]$ is adhesive and in particular the category $\mathbf{Graph} = [E \rightrightarrows V, \mathbf{Set}]$ is adhesive where $E \rightrightarrows V$ is the two object category with two morphisms $s, t\colon E \to V$.

Instead, the full subcategory of simple graphs, denoted by **sGraph**, is not adhesive (Johnstone, Lack and Sobociński, 2007), where a graph $\mathcal{G} \in \mathbf{Graph}$ is *simple* if for any two nodes, there is at most one edge from the first to the second. Finally, also **Top**, the category of topological spaces, is not adhesive (Lack and Sobociński, 2005).

As observed above, given an object of an adhesive category **C**, the slice category over this object is again adhesive; the reason is that all relevant constructions and properties are directly inherited from **C**.

Definition 2.3 (Slice Category). For an object $T \in \mathbf{C}$, the *slice category over* T, denoted $\mathbf{C}{\downarrow}T$, has **C**-morphisms $A \xrightarrow{a} T$ with codomain T as objects. A $\mathbf{C}{\downarrow}T$-morphism $\psi \colon (A \xrightarrow{a} T) \to (B \xrightarrow{b} T)$ is a **C**-morphism $\psi \colon A \to B$ that satisfies $a = b \circ \psi$. Further, we denote by $|\_|_T \colon \mathbf{C}{\downarrow}T \to \mathbf{C}$ the *forgetful functor*, which maps each object $A \xrightarrow{a} T$ to A and acts as the identity on morphisms.

In this paper, we shall often use the fact that subobjects of any object in an adhesive category form a distributive lattice (Lack and Sobociński, 2005) (see also Proposition 2.5).

Definition 2.4 (Subobject Poset). Let $T \in \mathbf{C}$ be an object, and let $a \colon A \rightarrowtail T$ and $a' \colon A' \rightarrowtail T$ be two monos. The monos a and a' are *isomorphic*, written $a \cong_T a'$, if there exists an isomorphism $i \colon A \to A'$ such that $a = a' \circ i$. A *subobject of* T is an isomorphisms class of a mono $a \colon A \rightarrowtail T$, denoted by $[a]$, i.e., $[a] = \{a' \mid a \cong_T a'\}$. Given two monos $a \colon A \rightarrowtail T$ and $b \colon B \rightarrowtail T$, $[a]$ *is included in* $[b]$, written $a \sqsubseteq_T b$, if there exists an arrow $j \colon A \rightarrowtail B$ such that $a = b \circ j$. Finally, the *subobject poset over* T is $\langle \mathrm{Sub}(T), \sqsubseteq_T \rangle$ where $\mathrm{Sub}(T)$ is the set of all subobjects of T.

In the sequel, when a is a mono we shall sometimes just write a in place of $[a]$ to denote the corresponding suboject.

In the category of sets and functions, we have a canonical choice of representatives for each subobject, namely the inclusion of a subset. Hence subobject posets are just power set lattices. As mentioned above, for adhesive categories, subobject posets can be proved to be distributive lattices.

Proposition 2.5 (Distributive Subobject Lattices (Lack and Sobociński, 2005)). Any subobject poset in an adhesive category is a distributive lattice, where the meet $[a] \sqcap [b]$ of two subobjects $[a]$, $[b]$ is obtained by taking the pullback of their representatives and the join $[a] \sqcup [b]$ is obtained by taking a pullback, followed by a pushout (i.e., adhesive categories have *effective unions*).

$$\begin{tikzcd}
& B \arrow[dr, tail, "b"] & \\
A\sqcap B \arrow[ur, tail] \arrow[rr, tail, "a\sqcap b"] \arrow[dr, tail] & & T \\
& A \arrow[ur, tail, "a"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
& B \arrow[dr] \arrow[drr, tail, "b"] & & \\
A\sqcap B \arrow[ur, tail] \arrow[dr, tail] & & A\sqcup B \arrow[r, tail, "a\sqcup b"] & T \\
& A \arrow[ur] \arrow[urr, tail, "a"'] & &
\end{tikzcd}$$

Another operation on subobjects that is directly connected to the SPO rewriting mechanism, as shown in Proposition 2.10, is *relative pseudo-complementation* (Birkhoff, 1967).

Definition 2.6 (Relative Pseudo-Complement (RPC)). Let $\langle L, \sqsubseteq \rangle$ be a lattice. A *relative pseudo-complement* (RPC) *of a with respect to b* is an element $d \in L$ that satisfies $a \sqcap x \sqsubseteq b \iff x \sqsubseteq d$ for all $x \in L$. If such an element exists then it is unique, and it is denoted $a \mathrel{-\!\!\bullet} b$. The lattice L is *relatively pseudo-complemented* (RPC) if the RPC $a \mathrel{-\!\!\bullet} b$ exists for all pairs $a, b \in L$.

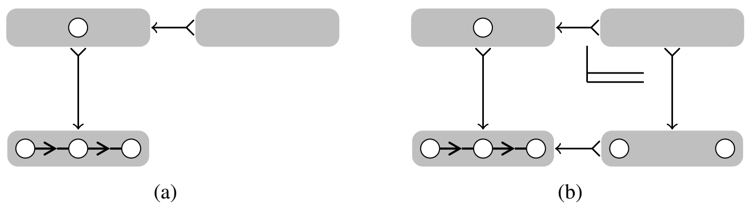


Fig. 2. A relative pseudo-complement

In a finite distributive lattice the RPC of a w.r.t. b always exists and can be characterised as $a \multimap b = \bigsqcup\{y \mid a \sqcap y \sqsubseteq b\}$. We consider the following two special cases:

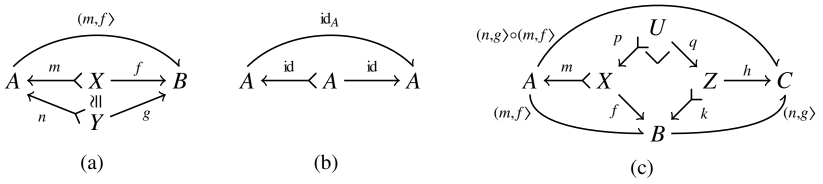
— In the case of a powerset lattice, given two sets $B, A \in \wp(M)$, the RPC of A w.r.t. B, is the set $M \setminus (A \setminus B) = \{m \in M \mid m \notin A \text{ or } m \in B\}$.

— In the case of subobject lattices, if $[a], [b] \in \mathrm{Sub}(T)$, with $[a] \sqsupseteq [b]$, the RPC $[c] = [a] \multimap [b]$ with $c\colon (A \multimap B) \rightarrowtail T$ gives rise to a pullback square. In fact, using the defining condition of RPCs we have that $[b] \sqsubseteq [c]$ (because clearly $[a] \sqcap [b] \sqsubseteq [b]$), and thus $[a] \sqcap [c] \sqsubseteq [b]$. Furthermore, $[c]$ is the *largest* pullback complemement of arrows i and a among the subobjects of T: in fact, if $[b] = [a] \sqcap [d]$ for some $[d] \in \mathrm{Sub}(T)$, then $[d] \sqsubseteq [c]$ follows.

$$\begin{tikzcd}
A \arrow[d, tail, "a"'] & B \arrow[l, tail, "i"'] \arrow[d, tail] \arrow[dl, phantom, "\sqsubseteq"] \\
T & A \multimap B \arrow[l, tail, "a \multimap (a \circ i)"]
\end{tikzcd}$$

The induced pullback square is sometimes referred to as an RPC *square* and is marked by an asymmetric “double” corner. In adhesive categories, if a given pair of composable monos has a *final* pullback complement (Dyckhoff and Tholen, 1987), then the largest and the final pullback complements coincide (Corradini, Heindel, Hermann and König, 2006, Lemma 2). Finally, note that the RPC $[a] \multimap [a]$ is $[\mathrm{id}_T]$ (if the top arrow of the pullback is an iso so is the bottom arrow).
As an example we consider the category **Graph** of directed graphs and graph morphisms of Example 2.2. Given the two graph inclusions that are shown in Figure 2(a), i.e., the middle node of two consecutive edges and the empty graph, the RPC of the middle node w.r.t. the empty graph are the extremal nodes, which is the graph inclusion on the bottom in Figure 2(b).

Next, we mention two lemmas about RPCs that will feature prominently in the proof of Proposition 2.18.



$$\begin{tikzcd}
A & X \arrow[l, tail, "m"'] \arrow[r, "f"] \arrow[d, phantom, "\cong"] & B \\
& Y \arrow[ul, tail, "n"] \arrow[ur, "g"'] &
\arrow[from=1-1, to=1-3, bend left, "{\langle m,f\rangle}"]
\end{tikzcd}
\qquad
\begin{tikzcd}
A & A \arrow[l, tail, "\mathrm{id}"'] \arrow[r, "\mathrm{id}"] & A
\arrow[from=1-1, to=1-3, bend left, "\mathrm{id}_A"]
\end{tikzcd}
\qquad
\begin{tikzcd}
& & U \arrow[dl, tail, "p"'] \arrow[dr, "q"] \arrow[dd, phantom, very near start, "\lrcorner" rotate=-45] & & \\
A & X \arrow[l, tail, "m"'] \arrow[dr, "f"'] & & Z \arrow[r, "h"] \arrow[dl, tail, "k"] & C \\
& & B & &
\arrow[from=2-1, to=2-5, bend left=50, "{\langle n,g\rangle\circ\langle m,f\rangle}"]
\arrow[from=2-1, to=3-3, bend right, "{\langle m,f\rangle}"']
\arrow[from=3-3, to=2-5, bend right, "{\langle n,g\rangle}"']
\end{tikzcd}$$

(a) (b) (c)

Fig. 3. Partial maps, their identies and their composition

Lemma 2.7 (Vertical RPC Lemma). Let **C** be an arbitrary category and let the diagram below, in (1) on the left, consist of two commuting squares of monos in **C** such that $A \overset{i}{\leftarrowtail} B \overset{b'}{\rightarrowtail} C$ is the pullback of $A \overset{a}{\rightarrowtail} T \overset{c}{\leftarrowtail} C$.

$$\begin{tikzcd}
X & Y \arrow[l, tail, "j"'] \arrow[d, tail, "y'"] \\
A \arrow[from=1-1, tail, "x"'] & B \arrow[l, tail, "i"'] \arrow[d, tail, "b'"] \arrow[dl, phantom, very near start, "\llcorner"] \\
T \arrow[from=2-1, tail, "a"'] & C \arrow[l, tail, "c"]
\end{tikzcd}
\quad\Rightarrow\quad
\left(
\begin{tikzcd}
X & Y \arrow[l, tail, "j"'] \arrow[d, tail, "y'"] \arrow[dl, phantom, very near start, "\llcorner"] \\
A \arrow[from=1-1, tail, "x"'] & B \arrow[l, tail, "i"'] \arrow[d, tail, "b'"] \arrow[dl, phantom, very near start, "\llcorner"] \\
T \arrow[from=2-1, tail, "a"'] & C \arrow[l, tail, "c"]
\end{tikzcd}
\quad\Longleftrightarrow\quad
\begin{tikzcd}
X \arrow[d, tail, "a\circ x"'] & Y \arrow[l, tail, "j"'] \arrow[d, tail, "b'\circ y'"] \arrow[dl, phantom, very near start, "\llcorner"] \\
T & C \arrow[l, tail, "c"]
\end{tikzcd}
\right)
\tag{1}$$

Then the two (small) squares are RPC squares if and only if the outer rectangle is an RPC square, i.e., $[c] = [a] \multimap [a \circ i]$ (in $\mathrm{Sub}(T)$) and $[i] = [x] \multimap [x \circ j]$ (in $\mathrm{Sub}(A)$) if and only if $[c] = [a \circ x] \multimap [a \circ x \circ j]$ (in $\mathrm{Sub}(T)$).

Proof. Similarly to (Löwe, 2010, Proposition 5), the desired result can be verified in a straight-forward manner using the universal properties of RPC squares and the fact that all arrows (and in particular a) are monic. □

Lemma 2.8 (RPC squares by pushout ((Lack and Sobociński, 2005, Lemma 2.3))). Let **C** be an adhesive category, let $B \overset{m}{\leftarrowtail} A \overset{n}{\rightarrowtail} C$ be a span of monos in **C** and let $B \overset{b}{\rightarrowtail} D \overset{c}{\leftarrowtail} C$ be its pushout. Then the resulting pushout square is an RPC square, i.e., $[c] = [b] \multimap [b \circ m]$ (and $[b] = [c] \multimap [c \circ n]$).

We will show that the existence of all relative pseudo-complements in subobject lattices ensures that SPO rewriting is "well-behaved", i.e., as discussed later in detail, rules can always be applied at (monic) matches. In SPO rewriting, a rule is essentially a partial map which specifies what is deleted, what is preserved and what is created in any rule application. The formal definition is in terms of categories of partial maps, which we define next (see also (Robinson and Rosolini, 1988)).

Definition 2.9 (Partial Map Categories). Let **C** be a category with pullbacks. The *category* $\mathrm{Par}(\mathbf{C})$ *of partial maps (in* **C***)* has the same objects as **C**, i.e., $\mathrm{ob}(\mathrm{Par}(\mathbf{C})) = \mathrm{ob}(\mathbf{C})$. An arrow in $\mathrm{Par}(\mathbf{C})$ is a **C**-span $A \overset{m}{\leftarrowtail} X \xrightarrow{f} B$ with monic m, taken up to isomorphisms at X (Figure 3(a)). It is called a *partial map* and is written $\langle m_{(X)} f\rangle \colon A \rightharpoonup B$ or just $\langle m, f\rangle \colon A \rightharpoonup B$.

If m is an isomorphism, then $\langle m, f\rangle$ is called a *total map*. The identity on an object A is $\langle \mathrm{id}_A, \mathrm{id}_A\rangle \colon A \rightharpoonup A$ (Figure 3(b)); composition is defined via pullbacks (Figure 3(c)). Note that a partial map $\langle m, f\rangle \colon A \rightharpoonup B$ is monic in Par(**C**) if and only if it is total and f is monic in **C**; hence we often write $C \overset{g}{\rightarrowtail} D$ instead of $C \xrightharpoonup{\langle \mathrm{id}, g\rangle} D$.

We next examine pushouts along monos in the category of partial maps. This is later used for defining SPO rewriting "directly" in the category **C**, without explicitly referring to Par(**C**).

Proposition 2.10 (Partial Map Pushouts along Monos). Let **C** be an adhesive category. Then pushouts along monos exist in Par(**C**) if and only if subobject lattices in **C** are relatively pseudo-complemented.

Proof sketch. First, assume that we have pushouts along monos in Par(**C**). Let $T \in \mathbf{C}$ be an object and let $[a], [b] \in \mathrm{Sub}(T)$ be subobjects; now one can verify that the RPC $[a] \multimap\!\!\bullet\, [b]$ exists iff the RPC $[a] \multimap\!\!\bullet\, [a] \sqcap [b]$ exists (since Sub(T) is a meet-semilattice). Hence, w.l.o.g., we can assume that $[a] \sqsupseteq [b]$, i.e., we have a unique morphism $i \colon b \to a$ in $\mathbf{C}{\downarrow}T$. Next, construct the following pushout in Par(**C**).

$$\begin{tikzcd}
A \arrow[r, harpoon, "\langle i, \mathrm{id}_B\rangle"] \arrow[d, tail, "a"'] & B \arrow[d, harpoon, "a'"] \\
T \arrow[r, harpoon, "\langle j, k\rangle"'] \arrow[ur, phantom, "\ulcorner", very near end] & C
\end{tikzcd}$$

Now, it remains to verify that $[j] = [a] \multimap\!\!\bullet\, [b]$.

Conversely, assume that subobject lattices in **C** have all RPCs. Now the pushout of a partial map $\langle \alpha_{(K)}\beta\rangle \colon L \rightharpoonup R$ along a mono $m \colon L \rightarrowtail A$ can be constructed as in Figure 4: first, we construct the RPC $[m] \multimap\!\!\bullet\, [m \circ \alpha]$; second, we take the pushout of the (domain of the) constructed RPC and R over K. □

$$\begin{tikzcd}
L \arrow[d, tail, "m"'] & K \arrow[l, tail, "\alpha"'] \arrow[r, "\beta"] & R & \rightsquigarrow & L \arrow[d, tail, "m"'] & K \arrow[l, tail, "\alpha"'] \arrow[r, "\beta"] \arrow[d, tail, "d"] \arrow[dl, phantom, "\sqsubseteq", very near start] & R \arrow[d, tail, "n"] \\
A & & & & A & D \arrow[l, tail, "m \multimap\bullet m \circ \alpha", "\varepsilon"'] \arrow[r, "\eta"'] \arrow[ur, phantom, "\ulcorner", very near end] & C
\end{tikzcd}$$

Fig. 4. Construction of partial map pushouts using relative pseudo-complements

Another property that will play a crucial role in the paper is pullback stability of RPCs (in the sense of Lemma 1.4.13 in (Johnstone, 2002)). In order to formalise it we need to introduce the so-called pullback functor.

Definition 2.11 (Pullback Functor). Let **C** be a category with (some choice of) pullbacks, and let $f \colon T' \to T$ in **C** be a morphism. The *pullback functor (along f)*, written $f^* \colon \mathbf{C}{\downarrow}T \to \mathbf{C}{\downarrow}T'$, maps every $(A \xrightarrow{a} T) \in \mathbf{C}{\downarrow}T$ to the right morphism $f^*(A) \xrightarrow{f^*(a)} T'$ in the left diagram below.

$$\begin{tikzcd}
A \arrow[d, "a"'] & f^*(A) \arrow[l, "f_a"'] \arrow[d, "f^*(a)"] \arrow[dl, phantom, "\llcorner"] \\
T & T' \arrow[l, "f"]
\end{tikzcd}
\qquad
\begin{tikzcd}
B \arrow[dd, bend right, "b"'] & & & f^*(B) \arrow[lll, "f_b"'] \arrow[dd, bend left, "f^*(b)"] \\
 & A \arrow[ul, "\varphi"] \arrow[d, "a"'] & f^*(A) \arrow[l, "f_a"'] \arrow[ur, "f^*(\varphi)"'] \arrow[d, "f^*(a)"] \arrow[dl, phantom, "\llcorner"] & \\
 & T & T' \arrow[l, "f"] &
\end{tikzcd}$$

The functor f^* maps each morphism $\varphi\colon a \to b$ in $\mathbf{C}{\downarrow}T$ to the unique $\mathbf{C}$-morphism $f^*(\varphi)$ that satisfies both $f_b \circ f^*(\varphi) = \varphi \circ f_a$ and $f^*(a) = f^*(b) \circ f^*(\varphi)$ as shown above on the right.

Then we say that in a category $\mathbf{C}$ RPCs are pullback stable if for every morphism $f\colon T' \to T$, the operation $\multimap$ on $\mathrm{Sub}(T)$ is preserved by the pullback functor f^*, i.e., the equation

$$[f^*(a \multimap b)] = [f^*(a)] \multimap [f^*(b)] \tag{2}$$

holds for all subobjects $[a], [b] \in \mathrm{Sub}(T)$ (where $a \multimap b$ is a representative of $[a] \multimap [b]$).

Every topos and every regular category satisfies this requirement.

2.2. *SPO Rewriting*

In this section we define SPO rewriting in adhesive categories. More precisely, throughout the paper we fix an adhesive category $\mathbf{C}$, where RPCs are pullback stable. Existence of RPCs will follow from the fact that we work on finite grammars. Pullback stability of RPCs will play a crucial role in the proof that that SPO rewriting is preserved by grammar morphisms (Lemma 4.6).

Single-pushout rewriting is performed by taking a pushout of a partial map along a mono – whence the name. Traditionally, partial maps are given by concrete representatives, which are called rules. To avoid switching continuously between the category $\mathbf{C}$ and the partial map category $\mathrm{Par}(\mathbf{C})$, we define rewriting "directly" in $\mathbf{C}$ (cf. Proposition 2.10).

Definition 2.12 (SPO Rules and Rewriting). A $\mathbf{C}$-*rule* is a partial map span $q = L \xleftarrow{\alpha} K \xrightarrow{\beta} R$, i.e., a span where α is monic; it is called *linear* if the right hand morphism β is monic and *consuming* if α is not an isomorphism. The class of consuming, linear $\mathbf{C}$-rules is denoted by $\mathscr{R}_{\mathbf{C}}$.

Let $A \in \mathbf{C}$ be an object. A *(monic) match* for a rule $q = L \xleftarrow{\alpha} K \rightarrowtail^{\beta} R$ into A is a mono $m\colon L \rightarrowtail A$ in $\mathbf{C}$. An SPO *direct derivation from A to C for q and m* is a double square diagram

$$\mathcal{X} = \begin{tikzcd}
L \arrow[d, tail, "m"'] & K \arrow[l, tail, "\alpha"'] \arrow[r, tail, "\beta"] \arrow[d, tail, "d"] \arrow[dl, phantom, "m \multimap m\circ\alpha"] & R \arrow[d, tail, "n"] \arrow[dl, phantom, "\ulcorner"] \\
A & D \arrow[l, tail, "\varepsilon"] \arrow[r, tail, "\eta"'] & C
\end{tikzcd}$$

where $[\varepsilon] = [m] \multimap [m \circ \alpha]$ in $\mathrm{Sub}(A)$ and C is the pushout of D and R over K in $\mathbf{C}$; in this situation we write $\mathcal{X}\colon A \overset{\langle q,m\rangle}{\Longrightarrow} C$ or simply $\mathcal{X}\colon A \overset{q}{\Longrightarrow} C$ (if m is not relevant). The rule q *rewrites* A *(at m) to* C, written $A \overset{q}{\Longrightarrow} C$ ($A \overset{\langle q,m\rangle}{\Longrightarrow} C$), if there exists a direct derivation $\mathcal{X}\colon A \overset{q}{\Longrightarrow} C$ ($\mathcal{X}\colon A \overset{\langle q,m\rangle}{\Longrightarrow} C$).

Remark 2.13 (Equivalence of Definitions). Note that, in the category **Graph**, the construction in Definition 2.12 is a special case of Construction 2.6 of (Löwe, 1993); moreover, Proposition 2.10 is a special case of Theorem 2.7 of (Löwe, 1993). For the purposes of the present paper, it suffices to generalise this special case to arbitrary adhesive categories to obtain a definition of single pushout rewriting without reference to the category of partial maps.

Following a common practice for process and unfolding semantics (Baldan et al., 2007), we restrict attention to linear and consuming rules. In order to understand this choice, observe that the process and unfolding approaches to the semantics, dating back to the seminal work in (Goltz and Reisig, 1983) and (Winskel, 1987a), are essentially based on the idea of tracing the histories of items in computations, identifying item occurrences with their histories. This allows one to properly define how item occurrences are related in terms of causality, conflict and concurrency. Non-consuming rules are problematic since, once a match is found, an unbounded number occurrences of such rules can be applied in parallel. Such occurrences have exactly the same history and thus are causally indistinguishable, a phenomenon, called autoconcurrency. Similarly, the idea of tracing univocally the histories of items in computations conflicts with the presence of fusions (non right-linearity) and duplications (non left-linearity). For this reason, we restrict to linear rules.

Instead, non-monic matches could in principle be allowed, without adding too many complications. Here we decided to use monic matches only, since, on the one hand, they allow for a simpler presentation, and, on the other hand, they are typically more expressive than general matches (see, e.g., (Habel, Müller and Plump, 2001) for the case of graphs).

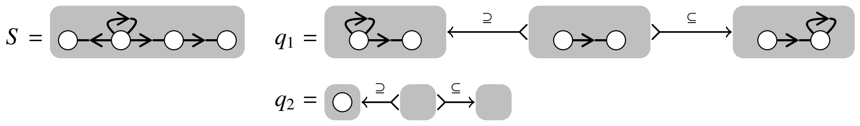


Fig. 5. The running graph transformation example

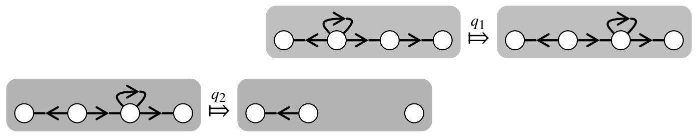


Fig. 6. Two rewriting steps in the running example

Example 2.14. The graph S in Figure 5 models a tiny network: vertices are network nodes, edges are directed network links, and looping edges represent stored messages. Further, the rules q_1 and q_2 in Figure 5 model message dispatching and failure of network nodes, respectively. Now the rule q_1 can rewrite the state S as it is shown in the first line of Figure 6; this rule application corresponds to the dispatching of the message to the right. In the latter state, the failure of the network node at which the message has arrived is captured by the application of the rule q_2 at that node as shown on the bottom in Figure 6.

Any two consecutive direct derivations $\mathcal{X}\colon A \vDash q \Rightarrow B$ and $\mathcal{X}'\colon B \vDash q' \Rightarrow C$ might either be dependent on each other – or not. The formal definition is as follows.

Definition 2.15 (Sequential Independence). Let $\mathcal{X}\colon A \vDash q \Rightarrow B$ and $\mathcal{X}'\colon B \vDash q' \Rightarrow C$ be two direct derivations as shown in the following display.

$$\begin{tikzcd}
L \arrow[d, tail, "m"'] & K \arrow[l, tail, "\alpha"'] \arrow[r, tail, "\beta"] \arrow[d, tail, "d"] \arrow[ld, phantom, "\llcorner"] & R \arrow[d, tail, "n"] \arrow[rrd, dotted, "s"] & L' \arrow[d, tail, "m'"] \arrow[lld, dotted, "t"'] & K' \arrow[l, tail, "\alpha'"'] \arrow[r, tail, "\beta'"] \arrow[d, tail, "d'"] \arrow[ld, phantom, "\llcorner"] & R' \arrow[d, tail, "n'"] \\
A & D \arrow[l, tail, "\varepsilon"] \arrow[r, tail, "\eta"'] \arrow[ru, phantom, "\ulcorner"] & B \arrow[r, equal] & B & D' \arrow[l, tail, "\varepsilon'"] \arrow[r, tail, "\eta'"'] \arrow[ru, phantom, "\ulcorner"] & C
\end{tikzcd}$$

Then $\mathcal{X}$ and $\mathcal{X}'$ are *sequential-independent* if there exist arrows $s\colon R \to D'$ and $t\colon L' \to D$ such that the above diagram commutes, i.e., such that the two equations $\varepsilon' \circ s = n$ and $\eta \circ t = m'$ hold.

Intuitively, consecutive direct derivations are considered independent of each other if the first rule does not create anything that is a pre-condition for the second rule (i.e., the first rule should not cause the second one) and the second rule does not destroy anything that is in the pre-condition of the first rule (i.e., the second rule should not prevent the application of the first one).

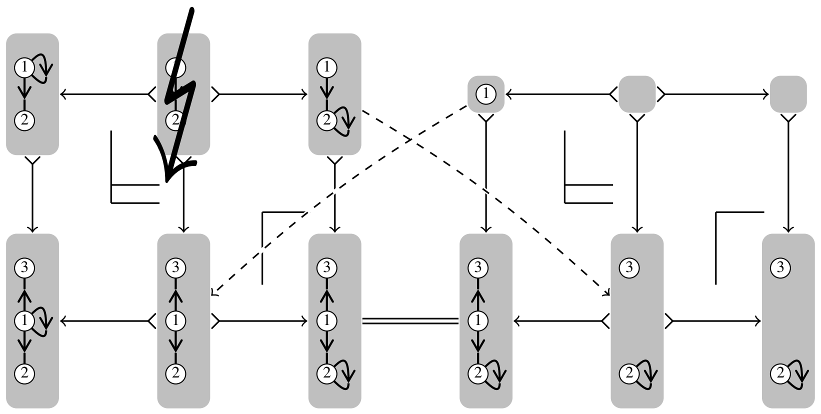
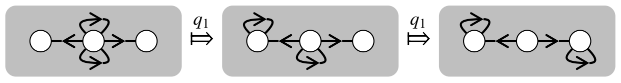


Fig. 7. (Counter-)example to sequential independence

Example 2.16 (Sequential Independence). Using the rules of our running example, two direct derivations that are not sequential-independent are given in Figure 7 where the involved graph morphisms act as the identity on the numbers of the nodes. Intuitively, this is due to the fact that the second derivation deletes node 1 which is used (preserved) by the first one. Hence they cannot be applied in reverse order. An example of two sequential-independent rewriting steps are the following ones.



The relevant objects of the rules of Definition 2.15, namely K, R, L' and K', can all be seen as subobjects of B. This is used in the following succinct characterisation of sequential independence.

Lemma 2.17 (Sequential Independence). Let the following be a pair of direct derivations as in Definition 2.15.

$$\begin{tikzcd}
L \arrow[d, "m"'] & K \arrow[l, "\alpha"'] \arrow[r, "\beta"] \arrow[d, "d"] \arrow[dr, "k"] & R \arrow[d, "r"] & L' \arrow[d, "l'"'] & K' \arrow[l, "\alpha'"'] \arrow[r, "\beta'"] \arrow[d, "d'"] \arrow[dl, "k'"'] & R' \arrow[d, "n'"] \\
A & D \arrow[l, "\varepsilon"] \arrow[r, "\eta"'] & B \arrow[r, equal] & B & D' \arrow[l, "\varepsilon'"] \arrow[r, "\eta'"'] & C'
\end{tikzcd}$$

They are sequential-independent if and only if $[r] \sqcap [l'] \sqsubseteq [k] \sqcap [k']$ holds in $\mathsf{Sub}(B)$.

Proposition 2.18 (Sequential Commutativity). Let q_1 and q_2 be a pair of linear rules, and let $\mathcal{X}_1 \colon A \vDash\langle q_1,m_1\rangle\Rightarrow B_1$ and $\mathcal{X}_2 \colon B_1 \vDash\langle q_2,n_2\rangle\Rightarrow C$ be a pair of sequential-independent direct derivations. Then there exists a pair of sequential-independent direct derivations $\mathcal{X}'_2 \colon A \vDash q_2\Rightarrow B_2$ and $\mathcal{X}'_1 \colon B_2 \vDash q_1\Rightarrow C$.

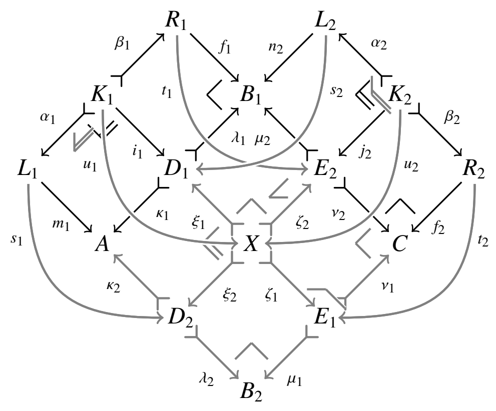


Fig. 8. Construction sketch for switching couples

Construction. The construction is essentially the same as the one for double pushout rewriting (see (Ehrig, 1979) and (Heindel, 2009, Proposition 3.6)); the main technical tools are Lemma 2.7 and Lemma 2.8. The construction is summarised in Figure 8. First X is constructed as the pullback of D_1 and E_2 over B_1; then there exist uniquely determined mediating morphisms $u_1 \colon K_1 \to X$ and $u_2 \colon K_2 \to X$ that make the diagram commute. Further, D_2 is the RPC of D_1 w.r.t. X where these objects are considered as subobjects of A; now there is a morphism $s_1 \colon L_1 \to D_2$ that satisfies $m_1 = \kappa_2 \circ s_1$. Next, E_1 is the pushout of X and R_2 over K_2; there is a morphism $v_1 \colon E_1 \to C$ that makes the diagram commute and yields a pushout square. Finally, B_2 is the pushout of D_2 and E_1 over X. In the end, applying Lemma 2.7 and Lemma 2.8, and using the morphisms in Figure 8, we have the following two sequential-independent direct derivations.

$$\begin{tikzcd}
L_2 \arrow[d, "\kappa_1\circ s_2"'] & K_2 \arrow[l, tail, "\alpha_2"'] \arrow[r, tail, "\beta_2"] \arrow[d, "\xi_2\circ u_2"] \arrow[dl, phantom, "\equiv"] & R_2 \arrow[dr, "\mu_1\circ t_2"'] & & L_1 \arrow[dl, "\lambda_2\circ s_1"] & K_1 \arrow[l, tail, "\alpha_1"'] \arrow[r, tail, "\beta_1"] \arrow[d, "\zeta_1\circ u_1"] \arrow[dl, phantom, "\equiv"] & R_1 \arrow[d, "\nu_2\circ t_1"] \\
A & D_2 \arrow[l, tail, "\kappa_2"] \arrow[rr, tail, "\lambda_2"'] \arrow[ur, phantom, "\ulcorner"] & & B_2 & & E_1 \arrow[ll, tail, "\mu_1"] \arrow[r, tail, "\nu_1"'] & C \arrow[ul, phantom, "\ulcorner"]
\end{tikzcd}$$

□

Definition 2.19 (Switching Couple). Let $\mathcal{XZ}\colon A \Longmapsto C$ and $\mathcal{Z}'\mathcal{X}'\colon A \Longmapsto C$ be two pairs of sequential-independent direct derivations; then $\langle \mathcal{XZ}, \mathcal{Z}'\mathcal{X}'\rangle$ is a *switching couple* if $\mathcal{Z}'\mathcal{X}'$ is obtained from $\mathcal{XZ}$ by the construction for Proposition 2.18.

Remark 2.20. In the proposed construction all new objects are determined by universal constructions, and therefore they are unique up to a unique "canonical" commuting isomorphism. Furthermore, if the construction is applied again to the resulting pair of sequential-independent direct derivations, the resulting derivation is isomorphic to the original one (in the sense of Definition 2.24). This fact has a lengthy direct proof but is also a consequence of Proposition 5.3 and Proposition 5.6. Note also that there can also exist "non-canonical" switchings of pairs of direct derivations, i.e., sometimes two rules can be applied in the opposite order but with different matches than the ones that are used above.

2.3. *Grammars and Concurrent Computations*

A grammar is a set of rules with a start object. This is in analogy to Petri nets: rules play the role of transitions, the start object is the counterpart of the initial marking. More precisely, as described in detail in (Baldan et al., 2007), the token game of a Petri net with place set P can be modelled by SPO rewriting in the slice category $\mathbf{S}\!\downarrow\!P$, where $\mathbf{S}$ is the category of (finite) sets and functions: multisets are encoded as functions with co-domain P where the multiplicity of $p \in P$ is the size of the inverse image of $\{p\}$. Abstracting away from sets, we work in the slice category $\mathbf{C}\!\downarrow\!T$ for a given "place" object T, also called *type object*. The definition of grammars in slice categories will be exploited in Section 4 for defining simulation morphisms between grammars.
Notation: We fix the following convention for rules $q \in \mathscr{R}_{\mathbf{C}\downarrow T}$: we assume $q = l_q \xleftarrow{\alpha_q} k_q \xrightarrow{\beta_q} r_q$ and $|q|_T = L_q \xleftarrow{\alpha_q} K_q \xrightarrow{\beta_q} R_q \in \mathscr{R}_{\mathbf{C}}$, where the latter is the obvious untyped version of q.

If $\mathbf{C}$ is adhesive and RPCs are stable under pullback, then every slice category $\mathbf{C}\!\downarrow\!T$ has the same properties (cf. Proposition 7.9 and (Lack and Sobociński, 2005)). Grammars in the slice category are called *typed* grammars.

Definition 2.21 (Typed Grammars). Let $T \in \mathbf{C}$ be an object, which will be referred to as *type object*. A *T-typed grammar G* is a pair $G = \langle Q, s\colon S \to T\rangle$ where $Q \subseteq \mathscr{R}_{\mathbf{C}\downarrow T}$ is a set of linear, consuming $\mathbf{C}\!\downarrow\!T$-rules, and $s\colon S \to T \in \mathbf{C}\!\downarrow\!T$ is the *start object*; it is *mono-typed* if $s\colon S \rightarrowtail T$ is monic and the left and right hand sides l_q and r_q of all rules $q \in Q$ are monic.

The *G-rewriting relation* over $\mathbf{C}\!\downarrow\!T$-objects, denoted by $\overset{G}{\Longmapsto}$, contains all pairs $a, b \in \mathbf{C}\!\downarrow\!T$

such that $a \models_{q}\Rightarrow b$ holds for some $q \in Q$; further an object $a \in \mathbf{C}{\downarrow}T$ is *reachable in* G if $s \models_{G}\Rightarrow^{*} a$, where $\models_{G}\Rightarrow^{*}$ is the transitive-reflexive closure of $\models_{G}\Rightarrow$.

Example 2.22 (Typed Grammar). We obtain a typed version of our running example by fixing the *type graph* T that is shown on the bottom left in Figure 9 where all messages are (implicitly) mapped to the arc with the triangular tip and all network links to the other edge. Hence the typing of the rules is given implicitly by the uniquely determined morphisms that preserve the two different kinds of arrow tips. The typed versions of the two example derivations are given in Figure 10.

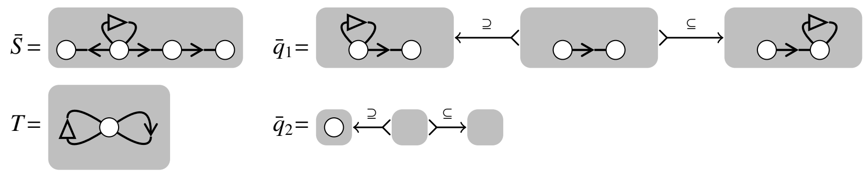


Fig. 9. The typed version of the running graph transformation example

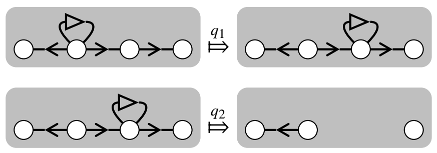
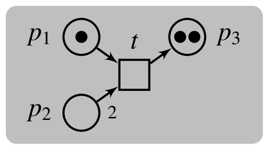


Fig. 10. The typed versions of the rewriting steps in Example 2.14

Example 2.23 (Petri Nets and Typed Set Rewriting). We give a simple example to illustrate the relation between Petri nets and rewriting of typed sets. Consider the following Petri net with an initial marking.



The corresponding grammar has the set of places $P = \{p_1, p_2, p_3\}$ as type object. The initial marking can be seen as a function from $\{(1, p_1), (1, p_3), (2, p_3)\}$ that maps each pair (i, p) to the second component p. In general, each subset $M \subseteq \mathbb{N} \times P$ yields a function $m\colon M \to P$ by composition with the projection $\pi_2\colon \mathbb{N} \times P \to P$ and thus can be seen as an object of $\mathbf{Set}{\downarrow}P$. Similarly, the following pair of inclusions

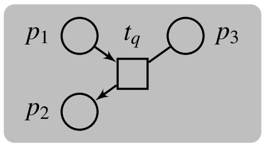
$$q_t = \{(1, p_1), (1, p_2), (2, p_2)\} \overset{\supseteq}{\leftarrow} \varnothing \overset{\subseteq}{\rightarrow} \{(1, p_3)\}$$

can be seen as a rule q_t in $\mathbf{Set}{\downarrow}P$ where K_{q_t} is the empty set; this rule is the counterpart of the transition t.

A rule in $\mathbf{Set}{\downarrow}P$ can have a non-empty set in the middle: in this case it corresponds to a Petri net transition with *read arcs*, which allow the transitions to test for the presence of tokens without consuming them. For example, the rule

$$q = \{(1,p_1),(1,p_3)\} \xleftarrow{\supseteq} \{(1,p_3)\} \xrightarrow{\subseteq} \{(1,p_3),(1,p_2)\}$$

would be represented as



where we have a read arc between t_q and p_3, which indicates that t_q can fire only if there is a token in p_3. Note that in general, given a set of places P, rules in $\mathbf{Set}{\downarrow}P$ are "more concrete" than net transitions over P, in the sense that to each (non-safe) marking enabling a transition there could correspond several different matches to which the rule is applicable. However, as discussed in Remark 7.8, there is an exact correspondence between safe nets and "safe" typed set rewriting as well as between their unfoldings.

A derivation in a grammar is just a sequence of "composable" direct derivations that only use rules of the grammar.

Definition 2.24 (Derivation). Let $G = \langle Q, s\colon S \to T\rangle$ be a T-typed grammar; a *G-derivation* is defined inductively as follows: $\epsilon_G\colon s \Longmapsto s$ is a G-derivation (of length 0), called *the empty derivation*; whenever $\tilde{\mathfrak{X}}\colon s \Longmapsto a$ is a G-derivation (of length n), $q \in Q$ is a rule and $\mathfrak{Z}\colon a \overset{q}{\Longmapsto} b$ is a direct derivation, then $\tilde{\mathfrak{X}}\mathfrak{Z}\colon s \Longmapsto b$ is a G-derivation (of length $n+1$).

Let $\tilde{\mathfrak{X}}\colon s \Longmapsto a$ and $\tilde{\mathfrak{X}}'\colon s \Longmapsto a'$ be two G-derivations of length $n > 0$ where each direct derivation is of the form:

$$\mathfrak{X}_i = \begin{tikzcd} l_i \arrow[d, tail, "m_i"'] & k_i \arrow[l, tail, "\alpha_i"'] \arrow[r, tail, "\beta_i"] \arrow[d, tail, "j_i"] \arrow[dl, phantom, "\llcorner"] & r_i \arrow[d, tail, "n_i"] \\ a_{i-1} & d_i \arrow[l, tail, "\varepsilon_i"] \arrow[r, tail, "\eta_i"'] \arrow[ur, phantom, "\ulcorner"] & a_i \end{tikzcd} \quad \text{and} \quad \mathfrak{X}'_i = \begin{tikzcd} l'_i \arrow[d, tail, "m'_i"'] & k'_i \arrow[l, tail, "\alpha'_i"'] \arrow[r, tail, "\beta'_i"] \arrow[d, tail, "j'_i"] \arrow[dl, phantom, "\llcorner"] & r'_i \arrow[d, tail, "n'_i"] \\ a'_{i-1} & d'_i \arrow[l, tail, "\varepsilon'_i"] \arrow[r, tail, "\eta'_i"'] \arrow[ur, phantom, "\ulcorner"] & a'_i \end{tikzcd}$$

for all $i \in \{1,\ldots,n\}$. Then $\tilde{\mathfrak{X}}$ and $\tilde{\mathfrak{X}}'$ are *isomorphic*, written $\tilde{\mathfrak{X}} \cong \tilde{\mathfrak{X}}'$, if there exist two families $\{\varphi_i\colon d_i \to d'_i\}_{i\in\{1,\ldots n\}}$ and $\{\psi_i\colon a_i \to a'_i\}_{i\in\{0,\ldots n\}}$ of isomorphisms (in $\mathbf{C}{\downarrow}T$) such that the diagram

$$\begin{tikzcd} & l_i \arrow[d, tail, "m_i"] \arrow[ddl, tail, bend right, "m'_i"'] & k_i \arrow[l, tail, "\alpha_i"'] \arrow[r, tail, "\beta_i"] \arrow[d, tail, "j_i"] \arrow[dl, phantom, "\llcorner"] & r_i \arrow[d, tail, "n_i"'] \arrow[ddr, tail, bend left, "n'_i"] & \\ & a_{i-1} \arrow[dl, "\psi_{i-1}"'] & d_i \arrow[l, tail, "\varepsilon_i"] \arrow[r, tail, "\eta_i"'] \arrow[d, "\varphi_i"] \arrow[ur, phantom, "\ulcorner"] & a_i \arrow[dr, "\psi_i"] & \\ a'_{i-1} & & d'_i \arrow[ll, tail, "\varepsilon'_i"] \arrow[rr, tail, "\eta'_i"'] & & a'_i \end{tikzcd}$$

commutes for all $i \in \{1,\ldots,n\}$ and $\psi_0 = \mathrm{id}_s$.

Isomorphic derivations are usually identified. Moreover, from a true concurrency point of view two derivations that can be obtained from each other by switching pairs of sequential-independent direct derivations should be considered the "same" concurrent computation. This is formalised as follows.

Definition 2.25 (Switch-Equivalence). Let $G = \langle Q, s\rangle$ be a grammar and let $\tilde{\mathcal{X}}, \tilde{\mathcal{Z}} \colon s \Longmapsto a$ be two derivations of length $n > 1$; they are *switchings of each other*, written $\tilde{\mathcal{X}} \sim_{\mathrm{sw}} \tilde{\mathcal{Z}}$, if $\langle \mathcal{X}_i\mathcal{X}_{i+1}, \mathcal{Z}_i\mathcal{Z}_{i+1}\rangle$ is a switching couple for some $i \in \{1, \ldots, n-1\}$ and $\mathcal{X}_j = \mathcal{Z}_j$ for all $j \in \{1, \ldots, n\} \setminus \{i, i+1\}$. *Switch equivalence*, denoted by $\approx_{\mathrm{sw}}$, is the reflexive transitive closure $(\sim_{\mathrm{sw}} \cup \cong)^*$ where $\cong$ is the isomorphism relation of G-derivations.

In the rest of the paper we restrict ourselves to *finite* grammars, similar to (Braatz, Ehrig, Gabriel and Golas, 2010).

Definition 2.26 (Finite Grammar). Let $G = \langle Q, s \colon S \to T\rangle$ be a grammar; then G is *finite* if the start object and all left and right hand sides are finite, i.e., $\mathrm{Sub}(S)$ is finite and $\mathrm{Sub}(L_q)$, $\mathrm{Sub}(R_q)$ are finite for all $q \in Q$. Moreover there are only finitely many isomorphisms between left hand sides of rules, i.e., the set $\{i \colon l_q \to l_{q'} \mid q' \in Q,\ i \text{ iso}\}$ is finite for each $q \in Q$.

Lemma 2.27 (Finiteness Lemma). Every reachable object in a finite grammar is finite.

Proof. The proof is by induction on the length of derivations. The base case is trivial. Thus let $G = \langle Q, s \colon S \to T\rangle$ be a finite grammar and let $a \colon A \to T$ be a finite reachable object in G. Let $q = (l \leftarrow\!\alpha\!\prec k \succ\!\beta\!\rightarrow r) \in Q$ be a rule and let

$$\begin{tikzcd}
l \arrow[d, tail, "m"'] & k \arrow[l, tail, "\alpha"'] \arrow[r, tail, "\beta"] \arrow[d, tail, "h"] \arrow[dl, phantom, "\llcorner", very near start] & r \arrow[d, tail, "n"] \\
a & d \arrow[l, tail, "\varepsilon"] \arrow[r, tail, "\eta"'] \arrow[l, phantom, "m \multimap m\circ\alpha", shift left=3] & c \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd}$$

be a direct derivation. Clearly d is finite because so is a. Let $[\mu] \in \mathrm{Sub}(c)$ be any subobject; now we have $\mu = \mu \sqcap (\eta \sqcup n) = (\mu \sqcap \eta) \sqcup (\mu \sqcap n)$. Thus, each subobject of c is the union of a pair of subobjects of d and r; hence – since d and r are finite – c is finite, whence the desired result follows. $\square$

As a consequence of this lemma, of Proposition 2.10, and of the existence of RPCs in finite lattices, we have that for each rule q and every match m of q into a reachable object, the pushout of q and m exists in $\mathrm{Par}(\mathbf{C})$: thus (as it is usual for the SPO approach) rewriting is possible at any match.

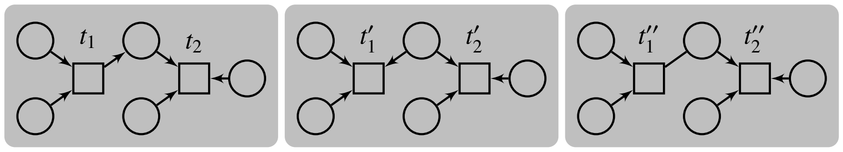
3. Occurrence Grammars and their Properties

In this section, we introduce the semantic domain that will be used to model (sets of) concurrent computations, namely *occurrence grammars*. Among the rules of an occurrence grammar suitable dependencies can be defined, including causality, conflicts and disabling, which are required to be acyclic. Occurrence grammars will be a central ingredient of the concurrent semantics of grammars: each single concurrent computation will correspond to a certain *deterministic occurrence grammar* (with additional information, which will be made precise in Definition 5.1). Furthermore, occurrence grammars turn out to enjoy a useful property: reachable objects can be characterised statically in terms of suitable dependency relations (causality and asymmetric conflict) among their rules. As a consequence, we shall not have to solve reachability problems when constructing the unfolding of a grammar in Section 6. In other words, an algorithm can be defined which builds unfoldings in a static manner without executing the rules dynamically.

3.1. *Occurrence Grammars*

We introduce now *occurrence grammars* formally. The definition will be based on two dependency relations among rules that have been introduced by the authors in (Baldan et al., 2006). The possible dependencies between two rules q and q' can roughly be described as follows: q *causes* q' if q produces something needed by q' to become activated, and q *can be disabled* by q' if q' destroys something on which q depends.

Example 3.1 (Causality and Disabling in a Petri Net). Consider the following three Petri nets.



The role of rules in grammars is played by transitions. In the first net, t_1 causes t_2, in the second net t'_1 can be disabled by t'_2 and vice versa. Finally, in the third net, we have a read arc, i.e., t''_1 only checks for the presence of a token in the place in the middle without consuming it whenever it should fire; as a result, only t''_2 can disable t''_1 while t''_1 cannot disable t''_2.

Definition 3.2 (Causality, Conflict). Let $G = \langle Q, s \colon S \to T \rangle$ be a mono-typed grammar; hence for each rule $q \in Q$, its components l_q, k_q and r_q represent subobjects $[l_q], [k_q]$ and $[r_q] \in \mathrm{Sub}(T)$. A pair of rules $q, q' \in Q$ may be related in any of the following ways.

$<$ **:** *q directly causes q'*,	written $q < q'$,	if $[r_q] \sqcap [l_{q'}] \not\sqsubseteq [k_q]$
$\ll$ **:** *q can be disabled by q'*,	written $q \ll q'$,	if $[l_q] \sqcap [l_{q'}] \not\sqsubseteq [k_{q'}]$

Further, the *asymmetric conflict* relation is defined as $\nearrow := <^+ \cup (\ll \setminus \mathrm{id}_Q)$, where $<^+$ is the transitive closure of $<$ and $\ll \setminus \mathrm{id}_Q$ the irreflexive version of $\ll$; moreover

— the *direct causes of q*,	are given by $\llcorner q \lrcorner = \{q' \in Q \mid q' < q\}$, and
— the *(complete) causes of q*,	are given by $\lfloor q \rfloor = \{q' \in Q \mid q' <^* q\}$.

Any subobject $[a] \in \mathrm{Sub}(T)$ may be related to a rule $q \in Q$ in a similar way:

$<$ **:** *q directly causes* $[a]$,	written $q < a$, if $[r_q] \sqcap [a] \not\sqsubseteq [k_q]$, and
$<_{\mathrm{co}}$ **:** $[a]$ is *(partly) consumed by q'*,	written $a <_{\mathrm{co}} q'$, if $[a] \sqcap [l_{q'}] \not\sqsubseteq [k_{q'}]$.

Finally, we have the following sets:

— the *consumers of* $[a]$,	are $\ulcorner a \urcorner = \{q' \in Q \mid a <_{\mathrm{co}} q'\}$ and
— the *(complete) causes of* $[a]$,	are $\lfloor a \rfloor = \{q' \in Q \mid \exists q \in Q.\, q' <^* q < a\}$.

Now we are ready to define *occurrence grammars*, which are a generalization of occurrence nets. We shall see later that in an occurrence grammar with type object T, all rule applications operate on subobjects of T, which is the counterpart of safety for occurrence nets (see also Proposition 3.5).

Definition 3.3 (Occurrence Grammar). An *occurrence grammar* is a mono-typed grammar $O = \langle Q, s\colon S \rightarrowtail T\rangle$ with a countable set of rules Q such that

1 the type object is the union of all right hand sides, i.e., $[\mathrm{id}_T] = [s] \sqcup \bigsqcup_{q\in Q}[r_q]$,
2 the transitive-reflexive closure $<^*$ of causality $<$ is a partial order,
3 for each rule $q \in Q$, $\lfloor q\rfloor$ is finite, and $\nearrow|_{\lfloor q\rfloor} := \nearrow \cap (\lfloor q\rfloor \times \lfloor q\rfloor)$ is acyclic,
4 the start object has no causes, i.e., $\lfloor s\rfloor = \emptyset$,
5 there are no backward conflicts, i.e., $[r_q] \sqcap [r_{q'}] \sqsubseteq [k_q] \sqcup [k_{q'}]$ for all $q \neq q' \in Q$,
6 left-hand sides are properly produced, i.e., $[l_q] \sqsubseteq [s] \sqcup \bigsqcup_{p'\in\lfloor q\rfloor}[r_{p'}]$ for all $q \in Q$.

The occurrence grammar $O = \langle Q, s\colon S \rightarrowtail T\rangle$ is *deterministic* if there are also no forward conflicts, i.e., $[l_q] \sqcap [l_{q'}] \sqsubseteq [k_q] \sqcup [k_{q'}]$ for all $q \neq q' \in Q$.

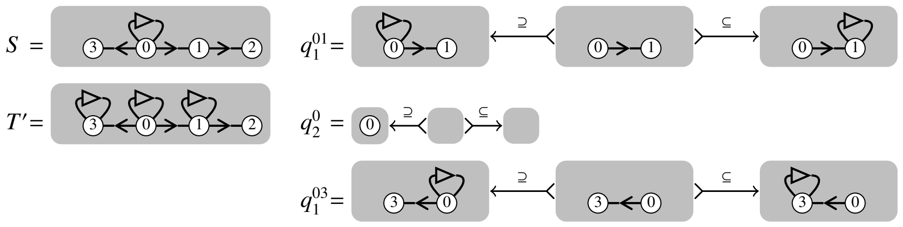


Fig. 11. Example occurrence grammar

Example 3.4. Consider the occurrence grammar in Figure 11, in which each rule captures a possible event in our running example, namely passing the message one step to the left or the right and the failure of the node 0. Again, the morphisms into the type object T' are implicitly expressed by the numbering of the items. In this example rule q_1^{01} can be disabled by q_2^0, i.e., $q_1^{01} \ll q_2^0$ and rules q_1^{01}, q_1^{03} can disable each other. Note that this occurrence grammar would become deterministic if we removed the rule q_1^{03} together with the loop at node 3 in T'. The loop has to be removed because the type graph must be the union of the start object and all right hand sides of rules.

3.2. *Properties of Occurrence Grammars*

In the theory of Petri nets, a characteristic property of occurrence nets is their safety, which means that every reachable marking is a set of places, rather than a proper multiset. The analogous result for occurrence grammars reads as follows.

Proposition 3.5 (Safety). Let $O = \langle Q, S \overset{s}{\rightarrowtail} T\rangle$ be an occurrence grammar and let $a \in \mathbf{C} \downarrow T$ be a reachable object, i.e., $s \vDash_O\Rightarrow^* a$. Then a is monic.

Proof. The proof is by induction on the length of derivations. As auxiliary properties, we shall show that for each derivation $\tilde{\mathcal{Z}}\colon\ s \Longmapsto a$

—each cause of a is a rule that has been applied in $\tilde{\mathcal{Z}}$, which we express by writing $\lfloor a\rfloor \lhd \tilde{\mathcal{Z}}$;
—each rule that is applied in $\tilde{\mathcal{Z}}$ is preceded by all its causes, for which we shall say that $\tilde{\mathcal{Z}}$ is *downward closed*;
—each consumer of a does not occur in $\tilde{\mathcal{Z}}$, written $\tilde{\mathcal{Z}} \lhd \ulcorner a\urcorner$.

The base case is trivial. Hence let $\tilde{\mathcal{X}}\colon\ s \Longmapsto a$, be a derivation, let $q = (l \leftarrowtail k \rightarrowtail r) \in Q$ be a rule such that $l \sqsubseteq a$, and let $\mathcal{X}\colon\ a \overset{q}{\Longmapsto} b$ be the following direct derivation.

$$\begin{tikzcd}
l \arrow[d, tail, "m"'] & k \arrow[l, tail, "\alpha"'] \arrow[r, tail, "\beta"] \arrow[d, tail, "h"] \arrow[dl, phantom, very near start, "\llcorner"] & r \arrow[d, tail, "n"] \\
a & d \arrow[l, tail, "\varepsilon", "m \multimap\!\bullet\, m \circ \alpha"'] \arrow[r, tail, "\eta"'] & b \arrow[ul, phantom, very near start, "\ulcorner"]
\end{tikzcd}$$

First, we observe that, since O is consuming, q is a consumer of a and thus it is a "new" rule, i.e., q has not been applied in $\tilde{\mathcal{X}}$ because $\tilde{\mathcal{X}} \lhd \ulcorner a \urcorner$ is part of the induction hypothesis.

To show that b is a mono we can use the fact that adhesive categories have effective unions (see Proposition 2.5) and prove that $[k] = [d] \sqcap [r]$ in $\mathsf{Sub}(T)$. For this it suffices to show that $[d] \sqcap [r] \sqsubseteq [k]$, which follows from the fact that $q \notin \lfloor a \rfloor$ (which in turn is a consequence of $\lfloor a \rfloor \lhd \tilde{\mathcal{X}}$).

Now it remains to verify that the auxiliary properties hold for $\tilde{\mathcal{X}}\mathcal{X}$:

—$\lfloor b \rfloor \lhd \tilde{\mathcal{X}}\mathcal{X}$.
Let $q'' \in \lfloor b \rfloor$, which means that that there is a rule $q' = (l' \leftarrowtail k' \rightarrowtail r') \in Q$ such that $q'' <^* q'$ and $r' \sqcap b \not\sqsubseteq k'$; we have to show that q'' occurs in $\tilde{\mathcal{X}}\mathcal{X}$. Now we derive

$$r' \sqcap b = r' \sqcap (d \sqcup r) = (r' \sqcap d) \sqcup (r' \sqcap r).$$

Hence either $q' \in \lfloor d \rfloor \subseteq \lfloor a \rfloor$ and $q'' <^* q'$ occurs in $\tilde{\mathcal{X}}$ (by the induction hypothesis) or $q = q'$ (since $q \not< q'$ and $r \sqcap r' \sqsubseteq k \sqcup k'$ by the definition of occurrence grammar if $q \neq q'$). In the latter case, i.e., if $q = q'$, we have $q'' <^* q \in \lfloor b \rfloor$ and thus q'' occurs in $\mathcal{X}$ (if $q'' = q'$) or in $\tilde{\mathcal{X}}$ (since $\lfloor a \rfloor \lhd \tilde{\mathcal{X}}$ and $l \sqsubseteq a$).

—$\tilde{\mathcal{X}}\mathcal{X}$ is downward closed.
Let $q' = (l' \leftarrowtail k' \rightarrowtail r') \in Q$ be a rule such that $q' < q$, i.e., $r' \sqcap l \not\sqsubseteq k'$. Using $\lfloor a \rfloor \lhd \tilde{\mathcal{X}}$ and $l \sqsubseteq a$, we conclude that q' occurs in $\tilde{\mathcal{X}}$, as desired.

—no consumer of b has been applied before, i.e., $\tilde{\mathcal{X}}\mathcal{X} \lhd \ulcorner b \urcorner$.
For this let $q' = (l' \leftarrowtail k' \rightarrowtail r') \in \ulcorner b \urcorner$, i.e., $l' \sqcap b \not\sqsubseteq k'$. Now we have $l' \sqcap b = l' \sqcap (d \sqcup r) = (l' \sqcap d) \sqcup (l' \sqcap r)$. If $l' \sqcap d \not\sqsubseteq k'$ then q' is a consumer of a and thus q' does not occur in $\tilde{\mathcal{X}}$; moreover $q' \neq q$ since q is not a consumer of d and we have shown that q' has not been applied in $\tilde{\mathcal{X}}\mathcal{X}$ in case that $l' \sqcap d \not\sqsubseteq k'$ holds. Otherwise, if $l' \sqcap d \sqsubseteq k'$ and $l' \sqcap r \not\sqsubseteq k'$, we at least know that $q \neq q'$ since one easily verifies that $l \sqcap r = k$ using the fact that asymmetric conflict is acyclic (see (Heindel, 2009, Corollary D.2)); in this second case, it remains to show that q' has not been applied in $\tilde{\mathcal{X}}$. As a last case distinction, assume first that $q < q'$. This however implies that q' also has not been applied in $\tilde{\mathcal{X}}$ by downward closedness of $\tilde{\mathcal{X}}\mathcal{X}$ (and the fact that q has not been applied in $\tilde{\mathcal{X}}$ as it is a consumer of a). If instead, $q \not< q'$, i.e., $r \sqcap l' \sqsubseteq k$, then $r \sqcap l' = r \sqcap l' \sqcap k = l' \sqcap k$ and thus (using $l' \sqcap r \not\sqsubseteq k'$) the rule q' is a consumer of k and thus a consumer of a. This implies that q' has not been applied in $\tilde{\mathcal{X}}$ by the induction hypothesis.

□

By the result above, reachable objects of occurrence grammars can be seen as subobjects of the type object. We next show that for occurrence grammars, instead of considering reachable objects, we can concentrate on the statically characterised *concurrent subobjects*, as they are exactly the ones contained in reachable objects. This will be used for the definition of the unfolding algorithm.

Definition 3.6 (Concurrent Subobject). Let $O = \langle Q, S \overset{s}{\rightarrowtail} T\rangle$ be an occurrence grammar. A subobject $[a] \in \mathrm{Sub}(T)$ is called a *concurrent subobject of O* if (i) $\lfloor a \rfloor$ is finite, (ii) $\lfloor a \rfloor \cap \ulcorner a \urcorner = \varnothing$, and (iii) $\nearrow|_{\lfloor a \rfloor}$ is acyclic.

Intuitively, $[a]$ is concurrent when its set of causes is finite and conflict-free (condition (i) and (iii), respectively) and there are no causal dependencies between subobjects of $[a]$ (condition (ii)).

Proposition 3.7 (Static Coverability). Let $O = \langle Q, S \overset{s}{\rightarrowtail} T\rangle$ be an occurrence grammar, and $[a] \in \mathrm{Sub}(T)$ be a subobject. Then $[a]$ is concurrent if and only if there is some reachable object b such that $[a] \sqsubseteq [b]$.

To obtain a reachable object that covers a concurrent object $[a]$, one just has to apply all its causes in any order compatible with asymmetric conflict. This fact establishes a bijective correspondence between reachable subobjects and suitable finite subsets of rules, called *configurations*.

Definition 3.8 (Configuration). Let $O = \langle Q, s\colon S \rightarrowtail T\rangle$ be an occurrence grammar. A *configuration* is a set of rules $C \subseteq Q$ such that

(i) asymmetric conflict on C does not contain cycles, i.e., $\nearrow|_C$ is acyclic;
(ii) each rule in C has finitely many predecessors w.r.t. asymmetric conflict, i.e., for all $q \in Q$, the set $\{q' \in C \mid q' \nearrow q\}$ is finite;
(iii) the configuration is downward closed w.r.t. causality, i.e., $\lfloor q \rfloor \subseteq C$ holds for all $q \in C$.

Each (finite) configuration gives rise to derivations; independently of the order of rule application, we obtain the same reachable subobject.

Proposition 3.9 (Reachable Objects). Let $O = \langle Q, s\colon S \rightarrowtail T\rangle$ be an occurrence grammar and let $C \subseteq Q$ be a finite configuration. Then there is a subobject $[a] \in \mathrm{Sub}(T)$ that is derivable ($s \vDash_O\Rightarrow^* a$) using each rule of C exactly once. Further the subobject $[a]$ can be described as follows:

$$[a] = \bigsqcup \Big\{ [x] \in \mathrm{Sub}(T) \,\Big|\, \big(\lfloor x \rfloor \subseteq C\big) \text{ and } \big(\ulcorner x \urcorner \cap C = \varnothing\big) \Big\} \tag{3}$$

Proof sketch (see also (Heindel, 2009, Proposition D.8)). The proof is by induction on the size of the configuration C with a trivial base case $C = \varnothing$. For the induction step, we choose a rule $q \in C$ that is maximal w.r.t. asymmetric conflict and show that its left hand side is contained in the reachable subobject that corresponds to the smaller configuration $C \setminus \{q\}$. It remains to apply the rule and verify that the obtained subobject is described by Equation (3). □

Therefore, a finite configuration is essentially the same as a concurrent computation of an occurrence grammar. This will become clearer in Section 5 where the process semantics of computations in arbitrary grammars is discussed, and where the following fact will be exploited.

Proposition 3.10. Let $O = \langle Q, S \overset{s}{\rightarrowtail} T\rangle$ be a deterministic occurrence grammar. Then the set Q is a configuration.

4. Categories of Grammars

Intuitively, the semantics of a grammar in terms of occurrence grammars is obtained by using occurrence grammars as representatives of (sets of) concurrent computations of the given grammar. What is still missing is a way to relate computations in the "generated" occurrence grammars to computations in the original grammar. Therefore we introduce a suitable notion of morphism between grammars that will play the role of this link to the semantic domain. Grammar morphisms will also play a crucial role in the main theorem about unfoldings, which characterises them as the canonical way – universal in category theoretic terms – to capture all the concurrent computations of a grammar by means of a single occurrence grammar.

The rough idea of grammar morphisms, which is already present in the literature on Petri nets and graph transformation (Winskel, 1987a; Baldan, 2000), is a simulation relation between its source an target. Thus, every computation in the source system is mapped to a computation of the target system. Another desirable property is that a notion of morphism defined in our abstract setting should specialise to the corresponding notions proposed for systems such as Petri nets and graph grammars.

The morphisms we introduce below will satisfy the first requirement, i.e., a grammar morphism will describe how the target grammar can simulate the source grammar. Concerning the second property, as explained more detailed in Remark 7.8, the proposed notion of morphism is "more concrete": for example, when $\mathbf{C}$ is the category of graphs, a graph grammar morphism of (Baldan et al., 2007) might be induced by several different ones according to our definition. However, this greater explicitness allows us to characterise the unfolding as a coreflection without restricting to the so-called *semi-weighted* grammars (Meseguer et al., 1997; Baldan, 2000; Baldan et al., 2007).

In analogy to work on Petri nets and their unfolding, where morphisms are monoid homomorphisms preserving the net structure, a morphism between two grammars typed over T and T', respectively, will be a suitable functor $\mathcal{F}\colon \mathbf{C}{\downarrow}T \to \mathbf{C}{\downarrow}T'$ that preserves the rules and the start object. We dub the class of functors that we shall consider *retyping operations*.

Definition 4.1 (Retyping Operation). A *retyping operation* $\mathcal{F}\colon \mathbf{C}{\downarrow}T \to \mathbf{C}{\downarrow}T'$ is a functor $\mathcal{F}\colon \mathbf{C}{\downarrow}T \to \mathbf{C}{\downarrow}T'$ such that there exists a natural transformation $\varphi\colon (|\textvisiblespace|_{T'} \circ \mathcal{F}) \xrightarrow{\cdot} |\textvisiblespace|_T$ that is *cartesian*, i.e., for each arrow $\psi\colon a \to b$ between objects $A \xrightarrow{a} T$ and $B \xrightarrow{b} T$ in $\mathbf{C}{\downarrow}T$, the span $A \xleftarrow{\varphi_a} \mathcal{F}(A) \xrightarrow{|\mathcal{F}(\psi)|} \mathcal{F}(B)$ is a pullback of $A \xrightarrow{|\psi|} B \xleftarrow{\varphi_b} \mathcal{F}(B)$ (where $\mathcal{F}(A)$ and $\mathcal{F}(B)$ are the domain of $\mathcal{F}(a)$ and $\mathcal{F}(b)$ respectively) and thus yields a pullback square $\begin{smallmatrix} B & \leftarrow & \mathcal{F}(B) \\ \uparrow & \lrcorner & \uparrow \\ A & \leftarrow & \mathcal{F}(A) \end{smallmatrix}$.

$$\begin{tikzcd}
& B \arrow[ddl, bend left, "b"] & & \mathcal{F}(B) \arrow[ll, "\varphi_b"'] \arrow[ddr, "\mathcal{F}(b)"] & \\
A \arrow[ur, "\psi"] \arrow[dd, "a"] & & \mathcal{F}(A) \arrow[ll, "\varphi_a"] \arrow[ur, "\mathcal{F}(\psi)"'] \arrow[ddrr, bend right, "\mathcal{F}(a)"] \arrow[ul, phantom, "\lrcorner", very near start] & & \\
& & & & \\
T & & & & T'
\end{tikzcd}$$

Every morphism $f\colon T \to T'$ in $\mathbf{C}$ induces a *(canonical) retyping operation* $f \circ \textvisiblespace\colon \mathbf{C}{\downarrow}T \to \mathbf{C}{\downarrow}T'$, which post-composes any $\mathbf{C}{\downarrow}T$-object with f and acts as the identity on morphisms.

This definition is closely related to the *pullback-retyping* used in (Baldan et al., 2007). In fact, as illustrated to the right, a retyping operation $\mathcal{F}\colon \mathbf{C}{\downarrow}T \to \mathbf{C}{\downarrow}T'$ with some cartesian natural transformation $\varphi\colon (|\_|_{T'} \circ \mathcal{F}) \dot{\to} |\_|_T$ acts by pulling back along φ_{id_T} followed by composition with $\mathcal{F}(\mathrm{id}_T)$, which is retyping along the span $T \leftarrow \mathcal{F}(T) \rightarrow T'$ in the terminology of (Baldan et al., 2007).

$$\begin{tikzcd}
 & B & & \mathcal{F}(B) \arrow[ll, "\varphi_b"'] \arrow[dl, bend right, gray] \arrow[ddr, bend left, "\mathcal{F}(b)"] & \\
A \arrow[ur, "\psi"] \arrow[d, "a"'] & & \mathcal{F}(A) \arrow[ll, "\varphi_a"] \arrow[ur, "\mathcal{F}(\psi)"'] \arrow[d, gray] \arrow[drr, bend right, "\mathcal{F}(a)"] & & \\
T & & \mathcal{F}(T) \arrow[ll, gray, "\varphi_{\mathrm{id}_T}"] \arrow[rr, gray, "\mathcal{F}(\mathrm{id}_T)"'] & & T'
\arrow[from=1-2, to=3-1, bend left, "b"]
\end{tikzcd}$$

Example 4.2 (Retyping Operation). We extend our example with a distinction between private and public nodes, depicted as ◎ and ○, respectively, and a new type of broadband connections, depicted using double lines $\Longrightarrow$. One possible way to map computations in the new system "back" to the old one is to ignore private nodes and to "implement" broad-band links by parallel "normal" links. The corresponding retyping operation is based on the span in Figure 12. The left

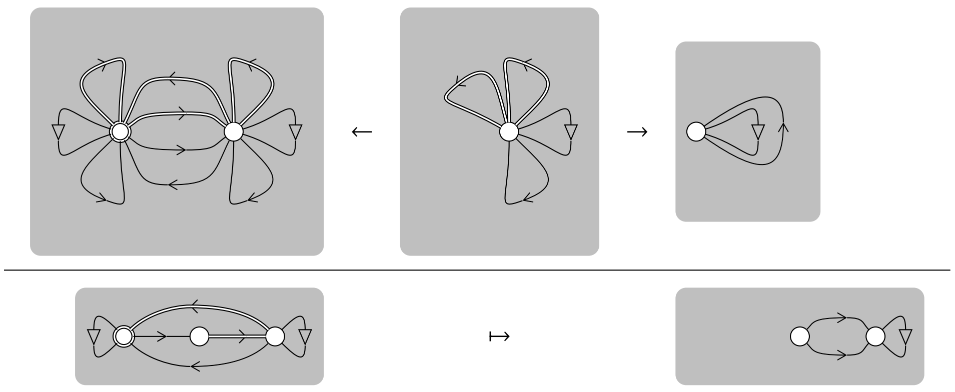


Fig. 12. An example span and its retyping operation applied to a state

leg maps the two double loops to the double loop on the public node and it acts as the inclusion on all other entities; the right leg instead maps the loops such that the tips are preserved, i.e., all loops in the middle are mapped to the outer loop on the right except for the loop with the triangular tip. Pulling back along the left leg followed by (post-)composition with the right leg now corresponds to deletion of private nodes and replacement of broadband links with a pair of parallel links, as illustrated in the bottom part of Figure 12: first the pullback action deletes the left node, creates two duplicates of the edge between the other two nodes, and preserves the rightmost loop; then, post-composition with the right hand morphism of the span makes the duplicates "normal" links and "preserves" the message.

Roughly, grammar morphisms are retyping operations that preserve the structure of grammars; however it is also possible to "ignore" some rules if they do not induce any action in the target system. The formal definition is as follows.

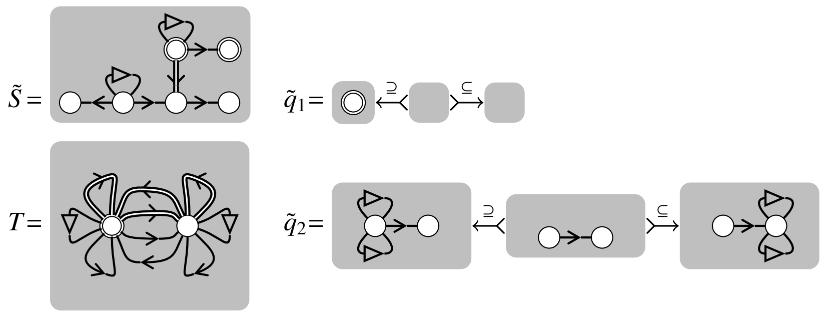
Definition 4.3 (Grammar Morphism). Let $G = \langle Q, s\colon S \to T\rangle$ and $G' = \langle Q', s'\colon S' \to T'\rangle$ be two typed grammars in $\mathbf{C}$. Then a *grammar morphism* from G to G', denoted by $\mathcal{F}\colon G \to G'$, is a retyping operation $\mathcal{F}\colon \mathbf{C}{\downarrow}T \to \mathbf{C}{\downarrow}T'$ such that

(i) the start object is preserved, i.e., $\mathcal{F}(s) = s'$, and

(ii) for any rule $q \in Q$, the *image* $\mathcal{F}(q) := \mathcal{F}(l_q) \xleftarrow{\mathcal{F}(\alpha_q)} \mathcal{F}(k_q) \xrightarrow{\mathcal{F}(\beta_q)} \mathcal{F}(r_q)$ is either a rule in G', i.e., $\mathcal{F}(q) \in Q'$, or an identity span, i.e., $\mathcal{F}(q) = \mathcal{F}(l_q) \xleftarrow{\text{id}} \mathcal{F}(l_q) \xrightarrow{\text{id}} \mathcal{F}(l_q)$.

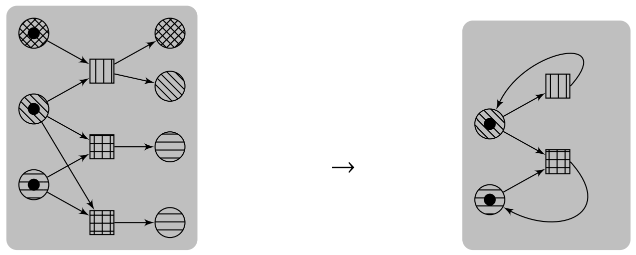
Grammar morphisms can be seen as generalizations of the Petri net morphisms introduced in (Winskel, 1987a). For Petri nets they have been proved to enjoy useful properties: they ensure the existence of products, which can be interpreted as asynchronous compositions, and of some coproducts, modelling nondeterministic choice (Winskel, 1987b). An application of products in unfolding-based diagnosis techniques can be found in (Baldan et al., 2010).

Example 4.4 (Grammar Morphism). To illustrate some crucial points of the definition, consider the following grammar with the usual implicit typing that arises by those morphisms that preserve the kind of entities.

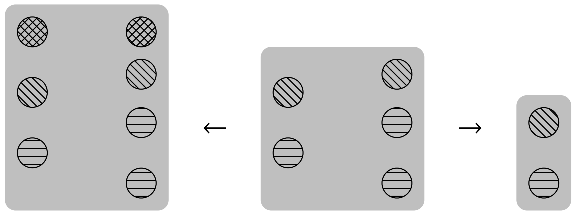


Now, we check whether the retyping operation of Example 4.2 is actually a grammar morphism from the grammar above into the typed version of our running example, presented as Example 2.22. First, the start object is preserved because the private nodes are deleted with all adjacent edges. The rule $\tilde{q}_1$ is mapped to the identity span on the empty graph, which is allowed (and is in analogy to process calculi where silent transitions might be “ignored”). Instead, the rule $\tilde{q}_2$ is left unchanged. Hence, the retyping operation based on the span of Figure 12 is not a grammar morphism since our running example grammar does not include any rule that transmits two messages at the same time.

Example 4.5 (Petri Net Morphism). In order to illustrate the connection to Petri net morphisms, we present here a morphism between two grammars encoding Petri nets, as described in Example 2.23. Consider the two nets given below.



There is a grammar morphism from the left-hand net to the right-hand net, based on the span depicted below.



Note that the net on the left contains several duplicates of places and transitions that occur only once in the net on the right. Moreover it contains two new places that are deleted by the morphism.

This notion of morphism is closely related to, but more concrete than the net morphisms of (Winskel, 1987a), which consist of partial mappings between sets of transitions and multi-relations between sets of places. In fact, as multisets over a set of places P can be seen as objects in the slice category $\mathbf{Set}{\downarrow}P$, multirelations can be represented by spans between the sets of places.

We next prove the *Simulation Lemma:* it shows that grammar morphisms map direct derivations in the domain to corresponding ones in the co-domain, which are either again direct derivations in the target grammar or "identity steps". Hence grammar morphisms preserve reachability.

Lemma 4.6 (Simulation Lemma). Let $\mathcal{F}\colon G \to G'$ be a morphism between two grammars where $G = \langle Q, s\colon S \to T\rangle$ and $G' = \langle Q', s'\colon S' \to T'\rangle$. Then, if $\mathcal{F}(q)$ is not an identity span, for any direct derivation $\mathcal{X}\colon a \vDash\!\langle q,m\rangle\!\Rightarrow b$ in G, there is a corresponding direct derivation $\mathcal{X}'\colon \mathcal{F}(a) \vDash\!\langle \mathcal{F}(q),\mathcal{F}(m)\rangle\!\Rightarrow \mathcal{F}(b)$ in G'.

Proof sketch (see also (Heindel, 2009, Lemma 6.4)). Let $\mathcal{X}\colon (A\text{–}a\!\to T) \vDash\!\langle q,m\rangle\!\Rightarrow (B\text{–}b\!\to T)$ be a direct derivation that uses a match $m\colon l \rightarrowtail a$ and a rule $q = l \leftarrow\!\alpha\!- k -\!\beta\!\to r \in Q$. By definition, $\mathcal{X}$ is a diagram of the form $\begin{tikzcd} l \arrow[d, tail] & \cdot \arrow[l] \arrow[r] \arrow[d, tail] \arrow[dl, phantom, "\llcorner"] & r \arrow[d, tail] \\ a & \cdot \arrow[l] \arrow[r] \arrow[ur, phantom, "\ulcorner"] & b \end{tikzcd}$. As pushouts and pullbacks in $\mathbf{C}{\downarrow}T$ are constructed in $\mathbf{C}$, it is enough to consider the underlying $\mathbf{C}$-diagram $\begin{tikzcd} L \arrow[d, tail] & \cdot \arrow[l] \arrow[r] \arrow[d, tail] \arrow[dl, phantom, "\llcorner"] & R \arrow[d, tail] \\ A & \cdot \arrow[l] \arrow[r] \arrow[ur, phantom, "\ulcorner"] & B \end{tikzcd}$. Now choose any cartesian natural transformation $\varphi\colon (|\llcorner\!\lrcorner|_{T'} \circ \mathcal{F}) \stackrel{\cdot}{\to} |\llcorner\!\lrcorner|_T$; it provides not only arrows $\varphi_a\colon \mathcal{F}(A) \to A$ and $\varphi_b\colon \mathcal{F}(B) \to B$ into the "tips" of the two squares, but actually a pair of "fitting" pullback cubes over $\begin{tikzcd} L \arrow[d, tail] & \cdot \arrow[l] \arrow[r] \arrow[d, tail] \arrow[dl, phantom, "\llcorner"] & R \arrow[d, tail] \\ A & \cdot \arrow[l] \arrow[r] \arrow[ur, phantom, "\ulcorner"] & B \end{tikzcd}$. The top face of the resulting double cube is $\begin{tikzcd} \mathcal{F}(L) \arrow[d, tail] & \cdot \arrow[l] \arrow[r] \arrow[d, tail] \arrow[dl, phantom, "\llcorner"] & \mathcal{F}(R) \arrow[d, tail] \\ \mathcal{F}(A) & \cdot \arrow[l] \arrow[r] \arrow[ur, phantom, "\ulcorner"] & \mathcal{F}(B) \end{tikzcd}$, because RPC squares and pushouts along monos are stable under pullback. $\square$

As a consequence of the Simulation Lemma, grammar morphisms can be extended to map derivations in their source grammar to derivations in the target grammar. This is made precise in the following definition.

Definition 4.7 (Image of Derivations). Let $G = \langle Q, s\colon S \to T\rangle$ and $G' = \langle Q', s'\colon S' \to T'\rangle$ be two typed grammars and let $\mathcal{F}\colon G \to G'$ be a grammar morphism. Further let the left of the following diagrams be a direct derivation in $\mathbf{C}{\downarrow}T$.

$$\mathcal{X} = \begin{tikzcd} l \arrow[d, tail, "m"'] & k \arrow[l, tail, "\alpha"'] \arrow[r, tail, "\beta"] \arrow[d, tail, "j"] \arrow[dl, phantom, "\llcorner"] & r \arrow[d, tail, "n"] \\ a & d \arrow[l, tail, "\varepsilon"] \arrow[r, tail, "\eta"'] \arrow[ur, phantom, "\ulcorner"] & b \end{tikzcd} \qquad \mathcal{F}(\mathcal{X}) = \begin{tikzcd} \mathcal{F}(l) \arrow[d, tail, "\mathcal{F}(m)"'] & \mathcal{F}(k) \arrow[l, tail, "\mathcal{F}(\alpha)"'] \arrow[r, tail, "\mathcal{F}(\beta)"] \arrow[d, tail, "\mathcal{F}(j)"] \arrow[dl, phantom, "\llcorner"] & \mathcal{F}(r) \arrow[d, tail, "\mathcal{F}(n)"] \\ \mathcal{F}(a) & \mathcal{F}(d) \arrow[l, tail, "\mathcal{F}(\varepsilon)"] \arrow[r, tail, "\mathcal{F}(\eta)"'] \arrow[ur, phantom, "\ulcorner"] & \mathcal{F}(b) \end{tikzcd}$$

Moreover assume that $\mathcal{F}(\alpha)$ is not an identity span. Then the *image of* $\mathcal{X}$ *under* $\mathcal{F}$, written $\mathcal{F}(\mathcal{X})$, is the direct derivation that is shown above on the right.

Grammar morphisms and the Simulation Lemma will play a crucial role in the proof of the main theorem about unfoldings; they also yield an elegant presentation of our theory of processes, which generalises the work of (Goltz and Reisig, 1983).

5. Concurrent Computations as Processes

In this section, we show that switch-equivalence classes of derivations are in one-to-one correspondence with so-called *processes*. We first introduce processes (of a given grammar); then we show that any such process can be seen as a representative of a full switch-equivalence class of typed derivations, all of which are "linearizations" of the process. Vice versa, given a derivation, a colimit-based construction allows us to derive the corresponding process. These two constructions are (essentially) inverse to each other, which is the main result about processes.

We shall now define the notion of G-process, i.e., a truly concurrent computation of a specific grammar G that is represented by an occurrence grammar.

Definition 5.1 (Processes). Let $G = \langle Q, s \colon S \to T\rangle$ be a T-typed grammar. Then a *G-process* is a pair $\mathcal{P} = \langle O, \mathcal{F} \colon O \to G\rangle$ where

1 $O = \langle Q', s' \colon S' \rightarrowtail T'\rangle$ is a deterministic occurrence grammar over T' where Q' is a finite set and
2 $\mathcal{F}$ is a canonical retyping operation, i.e., $\mathcal{F} = g \circ \llcorner\!\lrcorner$, for some $g \colon T' \to T$.

Let $\mathcal{P}_1 = \langle O_1, \mathcal{F}_1\rangle$ and $\mathcal{P}_2 = \langle O_2, \mathcal{F}_2\rangle$ be two G-processes. A *process isomorphism* $\mathcal{I} \colon \mathcal{P}_1 \to \mathcal{P}_2$ is an isomorphism from O_1 to O_2 in the slice category over G, i.e., $\mathcal{I} \colon O_1 \to O_2$ is an isomorphism of grammars such that $\mathcal{F}_1 = \mathcal{F}_2 \circ \mathcal{I}$.

Intuitively, by Proposition 3.10 the occurrence grammar O represents a concurrent computation and the morphism $\mathcal{F}$ provides a link back to the grammar G that specifies how each computation in O can be retyped over the type object of G.

Given a G-process $\mathcal{P}$, we can obtain a corresponding G-derivation by taking a "linearization" of the rules in O in any order compatible with asymmetric conflict (see also Proposition 3.9): we apply the rules in the chosen order and retype the obtained derivation over the type object of G.

Definition 5.2 (Derivations of a Process). Let $\mathcal{P} = \langle O, \mathcal{F}\rangle$ be a G-process, where $O = \langle Q, s'\rangle$. Let $\mathcal{X}_1 \cdots \mathcal{X}_n \colon s' \Longmapsto a$ be an O-derivation of length $n = |Q|$; then $\mathcal{F}(\mathcal{X}_1) \cdots \mathcal{F}(\mathcal{X}_n)$ is a *$\mathcal{P}$-derivation*. The set of all $\mathcal{P}$-derivations is denoted by $\mathrm{Drv}(\mathcal{P})$.

The next proposition shows that all derivations of a given process are "equivalent" from a true concurrency point of view. Hence Drv induces a mapping from (isomorphism classes of) processes to switch-equivalence classes of derivations.

Proposition 5.3. Let $\mathcal{P}$ and $\mathcal{P}'$ be processes such that $\mathcal{P} \cong \mathcal{P}'$. Then for all $\tilde{\mathcal{X}} \in \mathrm{Drv}(\mathcal{P})$ and $\tilde{\mathcal{Z}} \in \mathrm{Drv}(\mathcal{P}')$ it holds that $\tilde{\mathcal{X}} \approx_{\mathrm{sw}} \tilde{\mathcal{Z}}$.

Proof. Without loss of generality we assume that $\mathcal{P} = \mathcal{P}'$ where $\mathcal{P} = \langle O, \mathcal{F}\rangle$. Moreover it is enough to show that every pair of O-derivations $\tilde{\mathcal{X}}, \tilde{\mathcal{Z}}$ is switch-equivalent, i.e., $\tilde{\mathcal{X}} \approx_{\mathrm{sw}} \tilde{\mathcal{Z}}$ (where $O = \langle Q, s \colon S \rightarrowtail T\rangle$ is the relevant occurrence grammar).

Let $\pi[\tilde{\mathcal{X}}, \tilde{\mathcal{Z}}]$ be the permutation of $\{1, \ldots, n\}$, where $n = |Q|$ is the length of $\tilde{\mathcal{X}} = \mathcal{X}_1 \ldots \mathcal{X}_n$

and $\tilde{\mathcal{Z}} = \mathcal{Z}_1 \dots \mathcal{Z}_n$, such that $\mathcal{X}_i$ and $\mathcal{Z}_{\pi[\tilde{\mathcal{X}},\tilde{\mathcal{Z}}](i)}$ apply the same rule (at a possibly different match). Further, let

$$\left|\pi[\tilde{\mathcal{X}},\tilde{\mathcal{Z}}]\right| = \left|\left\{(i,j) \mid i < j \wedge \pi[\tilde{\mathcal{X}},\tilde{\mathcal{Z}}](j) < \pi[\tilde{\mathcal{X}},\tilde{\mathcal{Z}}](i)\right\}\right|,$$

that is, $\left|\pi[\tilde{\mathcal{X}},\tilde{\mathcal{Z}}]\right|$ is the number of pairs of positions that are switched by the permutation. The proof proceeds by induction on $\left|\pi[\tilde{\mathcal{X}},\tilde{\mathcal{Z}}]\right|$.

If $\left|\pi[\tilde{\mathcal{X}},\tilde{\mathcal{Z}}]\right| = 0$ then $\pi[\tilde{\mathcal{X}},\tilde{\mathcal{Z}}]$ is the identity, and it is straightforward to show that $\tilde{\mathcal{Z}} \cong \tilde{\mathcal{X}}$ (as any two derivations that apply the rules in the same order are isomorphic because the matches of rules are uniquely determined by inclusion witnesses).

If instead $\left|\pi[\tilde{\mathcal{X}},\tilde{\mathcal{Z}}]\right| = h > 0$, let $i = min\left\{j \mid \pi[\tilde{\mathcal{X}},\tilde{\mathcal{Z}}](j+1) < \pi[\tilde{\mathcal{X}},\tilde{\mathcal{Z}}](j)\right\}$, i.e., i is the first position in $\tilde{\mathcal{X}}$ which is switched in $\tilde{\mathcal{Z}}$ with the next position: it is easy to check that such a position exists. Let $q = (l \leftarrowtail k \rightarrowtail r)$ and $q' = (l' \leftarrowtail k' \rightarrowtail r')$ be the rules applied in direct derivations $\mathcal{X}_i$ and $\mathcal{X}_{i+1}$, respectively. Since q' is applied before q in $\tilde{\mathcal{Z}}$, from the auxiliary properties in the proof of Proposition 3.5 we have that $q \not< q'$ and $q' \notin \ulcorner r \urcorner$, and thus we can conclude that $\mathcal{X}_i\mathcal{X}_{i+1}$ is a pair of sequential-independent direct derivations by Lemma 2.17. Now, let $\tilde{\mathcal{X}}'$ be the derivation obtained from $\tilde{\mathcal{X}}$ by switching the direct derivations $\mathcal{X}_i$ and $\mathcal{X}_{i+1}$, using the construction of Proposition 2.18. We obviously have $\tilde{\mathcal{X}}' \in \mathrm{Drv}(\mathcal{P})$ and $\tilde{\mathcal{X}}' \sim_{\mathrm{sw}} \tilde{\mathcal{X}}$ by construction. Furthermore $\left|\pi[\tilde{\mathcal{X}}',\tilde{\mathcal{Z}}]\right| = h - 1$, and thus we have $\tilde{\mathcal{X}}' \approx_{\mathrm{sw}} \tilde{\mathcal{Z}}$ by induction hypothesis, which allows us to conclude that $\tilde{\mathcal{X}} \approx_{\mathrm{sw}} \tilde{\mathcal{Z}}$. □

Vice versa, given any derivation in a grammar G, we can generate a corresponding process by taking a colimit over the derivation (considered as a $\mathbf{C}{\downarrow}T$ diagram).

$$\begin{tikzcd}
s \arrow[r, Rightarrow, "\models q_1" , "\mathcal{X}_1"'] \arrow[drr, bend right, "s'"'] & a_1 \arrow[r, Rightarrow, "\models q_2", "\mathcal{X}_2"'] \arrow[dr, bend right, "a'_1"'] & \cdots \arrow[r, Rightarrow, "\models q_n", "\mathcal{X}_n"'] & a_n \arrow[dl, bend left, "a'_n"] \\
 & & t' &
\end{tikzcd} \quad \Big\} \text{colimit}$$

The domain of the colimit object is the type object of the occurrence grammar underlying the process. The rules that are applied in the direct derivations become the rules of the process. The morphism back to the type object of G is just the colimit object in $\mathbf{C}{\downarrow}T$. To make this formal, we define processes of derivations by induction on the length of derivations.

Definition 5.4 (Process of a Derivation). Let $G = \langle Q, s \colon S \to T\rangle$ be a T-typed grammar and let $\tilde{\mathcal{X}} \colon s \Longmapsto a$ be a G-derivation.

We define the *process of* $\tilde{\mathcal{X}}$ by induction on the length of the derivation $\tilde{\mathcal{X}}$; in our inductive definition, we keep track of the *end object* of each process.

($n = 0$) The process of $\epsilon_G \colon s \Longmapsto s$ is $\langle\langle\varnothing, \mathrm{id}_S\rangle, s \circ \lrcorner\rangle$; its end object is id_S.

($n \rightsquigarrow n+1$) Let $\tilde{\mathcal{X}}\mathcal{Z} \colon s \Longmapsto a_{n+1}$ be a G-derivation of length $n+1$ where $\mathcal{Z} \colon a_n \models q \Rightarrow a_{n+1}$ is a direct derivation for some $q \in Q$; its untyped image is depicted below.

$$|\mathcal{Z}|_T = \begin{tikzcd}
L \arrow[d, tail, "m"'] & K \arrow[l, tail, "\alpha"'] \arrow[r, tail, "\beta"] \arrow[d, tail, "d"] \arrow[dl, phantom, "\llcorner"] & R \arrow[d, tail, "n"] \\
A_n & D \arrow[l, tail, "\varepsilon", "m \multimap m\circ\alpha"'] \arrow[r, tail, "\eta"'] \arrow[ur, phantom, "\ulcorner"] & A_{n+1}
\end{tikzcd} .$$

Moreover let

$$\mathcal{P}_{\tilde{\mathcal{X}}} = \langle\langle Q, s_n \colon S \to T_n\rangle, t_n \circ \llcorner\lrcorner\rangle$$

be the process of $\tilde{\mathcal{X}}$ with end object $\bar{a}_n \colon A_n \to T_n$. Construct the following pushout and mediating morphism to T.

$$\begin{tikzcd}
 & A_{n+1} \arrow[dr, tail, "\bar{a}_{n+1}"] \arrow[drr, bend left, "a_{n+1}"] & & \\
D \arrow[ur, tail, "\eta"] \arrow[dr, tail, "\bar{a}_n \circ \varepsilon"'] & & T_{n+1} \arrow[r, dashed, "t_{n+1}"] & T \\
 & T_n \arrow[ur, tail, "i"'] \arrow[urr, bend right, "t_n"'] & &
\end{tikzcd}$$

(where the outer square commutes since $t_n \circ \bar{a}_n = a_n$ and rule q is typed over T). Now define rule q' as

$$q' = (i \circ \bar{a}_n \circ m) \overset{\alpha}{\leftarrow} (\bar{a}_{n+1} \circ n \circ \beta) \overset{\beta}{\rightarrow} (\bar{a}_{n+1} \circ n).$$

Finally,

$$\langle\langle i \circ (Q) \cup \{q'\}, i \circ s_n\rangle, t_{n+1} \circ \llcorner\lrcorner\rangle$$

is the process of $\tilde{\mathcal{X}}\mathcal{Z}$; its end object is $\bar{a}_{n+1}$.

The process of a derivation $\tilde{\mathcal{X}}$ is denoted by $\mathrm{Prc}(\tilde{\mathcal{X}})$.

It can be shown that the above definition is well-given, i.e., the described construction actually produces a process.

Lemma 5.5. Let $\tilde{\mathcal{X}}$ be a G-derivation. Then $\mathrm{Prc}(\tilde{\mathcal{X}})$ is a G-process.

The next proposition shows that if we start from two switch-equivalent derivations, the construction described in Definition 5.4 produces isomorphic processes. Hence Prc can be seen as a function from switch-equivalence classes of derivations to isomorphism classes of processes.

Proposition 5.6. Let $\tilde{\mathcal{X}}$ and $\tilde{\mathcal{Z}}$ be typed G-derivations such that $\tilde{\mathcal{X}} \approx_{\mathrm{sw}} \tilde{\mathcal{Z}}$. Then $\mathrm{Prc}(\tilde{\mathcal{X}}) \cong \mathrm{Prc}(\tilde{\mathcal{Z}})$.

Proof. The guiding idea is that colimits of switch-equivalent derivations (considered as diagrams) are isomorphic. Formally, the proof proceeds by induction on the number of switchings of pairs of sequential-independent direct derivations that are needed to transform $\tilde{\mathcal{X}}$ into $\tilde{\mathcal{Z}}$. Now, let $G = \langle Q, s \colon S \to T\rangle$ be a grammar with switch-equivalent derivations $\tilde{\mathcal{X}}$ and $\tilde{\mathcal{Z}}$.

For the base case, it is straighforward to show that isomorphic derivations have the same colimit. For the induction step we first switch two direct derivations in $\tilde{\mathcal{X}}$ to obtain $\tilde{\mathcal{Y}}$ such that $\tilde{\mathcal{Y}}$ is "one switching closer" to $\tilde{\mathcal{Z}}$. Then, we apply the induction hypothesis to obtain $\mathrm{Prc}(\tilde{\mathcal{Y}}) \cong \mathrm{Prc}(\tilde{\mathcal{Z}})$; therefore it is enough to show that $\mathrm{Prc}(\tilde{\mathcal{X}}) \cong \mathrm{Prc}(\tilde{\mathcal{Y}})$ where $\tilde{\mathcal{X}} \sim_{\mathrm{sw}} \tilde{\mathcal{Y}}$ are switchings of each other.

We split $\tilde{\mathcal{X}}$ and $\tilde{\mathcal{Y}}$ into a common prefix $\tilde{\mathcal{X}}'$, a switching couple $\langle \mathcal{X}_1\mathcal{Y}_2, \mathcal{Y}_2'\mathcal{X}_1'\rangle$, and a common suffix $\tilde{\mathcal{Y}}'$ such that $\tilde{\mathcal{X}} = \tilde{\mathcal{X}}'\mathcal{X}_1\mathcal{Y}_2\tilde{\mathcal{Y}}'$ and $\tilde{\mathcal{Y}} = \tilde{\mathcal{X}}'\mathcal{Y}_2'\mathcal{X}_1'\tilde{\mathcal{Y}}'$. The process construction from Definition 5.4 applied to $\tilde{\mathcal{X}}'$ yields the process $\mathrm{Prc}(\tilde{\mathcal{X}}') = \langle O', \mathcal{F}'\rangle$ and also an end object $\bar{a}_n$. We retype $\bar{a}_n$ over T to obtain $s' := \mathcal{F}'(\bar{a}_n)$. Now s' serves as a "new" start object, i.e., $\mathcal{X}_1\mathcal{Y}_2\mathcal{Y}'$ and $\mathcal{Y}_2'\mathcal{X}_1'\mathcal{Y}'$ are switch-equivalent derivations in $G' := \langle Q, s' \colon S' \to T\rangle$. Now it suffices to show that the process constructions for $\mathcal{X}_1\mathcal{Y}_2$ and $\mathcal{Y}_2'\mathcal{X}_1'$ yield isomorphic processes. The reason is that then also $\mathcal{X}_1\mathcal{Y}_2\mathcal{Y}'$ and $\mathcal{Y}_2'\mathcal{X}_1'\mathcal{Y}'$ will have isomorhic processes (as they begin with the "same" process after two direct derivations) and moreover, we can "glue" the process for $\tilde{\mathcal{X}}'$ in front by taking the pushout of the end object $\bar{a}_n$ and the start object of $\mathrm{Prc}(\mathcal{X}_1\mathcal{Y}_2\mathcal{Y}') \cong \mathrm{Prc}(\mathcal{Y}_2'\mathcal{X}_1'\mathcal{Y}')$.

To prove that $\mathcal{X}_1\mathcal{Y}_2$ and $\mathcal{Y}_2'\mathcal{X}_1'$ have isomorphic processes $\mathrm{Prc}(\mathcal{X}_1\mathcal{Y}_2) \cong \mathrm{Prc}(\mathcal{Y}_2'\mathcal{X}_1')$, consider the diagram in Figure 8. Assume that the "top" derivation is (the untyped version of) $\mathcal{X}_1\mathcal{Y}_2$ and that $\mathcal{Y}_2'\mathcal{X}_1'$ is the (untyped) constructed switching. To see that the respective processes are the "same", i.e., isomorphic, take the pushout of A and C over X; this yields a "common" type object of a process, which in fact is isomorphic to both $\mathrm{Prc}(\mathcal{X}_1\mathcal{Y}_2)$ and $\mathrm{Prc}(\mathcal{Y}_2'\mathcal{X}_1')$. □

The main result of this section makes precise that Prc and Drv can be seen as mutually inverse functions between switch-equivalence classes of derivations and isomorphism classes of processes.

Theorem 5.7 (Pseudo-Inverses Prc **and** Drv**).** Let **C** be an arbitrary adhesive category, let $T \in \mathbf{C}$ be an object, and let G be a finite T-typed grammar (see Definitions 2.26 and 2.21); let $\tilde{\mathcal{X}}$ be a typed G-derivation and let $\mathcal{P}$ be a G-process. Then:

1. $\tilde{\mathcal{Z}} \in \mathrm{Drv}\left(\mathrm{Prc}(\tilde{\mathcal{X}})\right)$ implies $\tilde{\mathcal{Z}} \approx_{\mathrm{sw}} \tilde{\mathcal{X}}$
2. $\mathcal{P}' \in \mathrm{Prc}\left(\mathrm{Drv}(\mathcal{P})\right)$ implies $\mathcal{P}' \cong \mathcal{P}$

Proof. For the first part, consider a derivation $\tilde{\mathcal{X}}$. It is a derivation of the process $\mathrm{Prc}(\tilde{\mathcal{X}})$ as the latter is "just" a colimt of the derivation; the desired now follows from Proposition 5.3.

For the second part, we first show that any process $\mathcal{P}$ can be "reconstructed" from an arbitrary $\mathcal{P}$-derivation. For this, we can show that any configuration C of an occurrence grammar induces a (sub-)occurrence grammar consisting only of the rules in C and of the involved subobjects (by induction on the size of configurations). Then we apply Proposition 3.10 and see that all rules induce a (sub-)occurrence grammar that is isomorphic to the one of $\mathcal{P}$. The desired then follows from Proposition 5.3 and Proposition 5.6. □

Example 5.8 (Processes and Derivations). As an example consider the process in Figure 13. Two corresponding derivations, which are not explicitly depicted here, can be obtained by considering the rule sequences $q_1^{01}, q_1^{12}, q_2^0$ and $q_1^{01}, q_2^0, q_1^{12}$, respectively. The corresponding derivations are switch-equivalent in the sense of Definition 2.25.

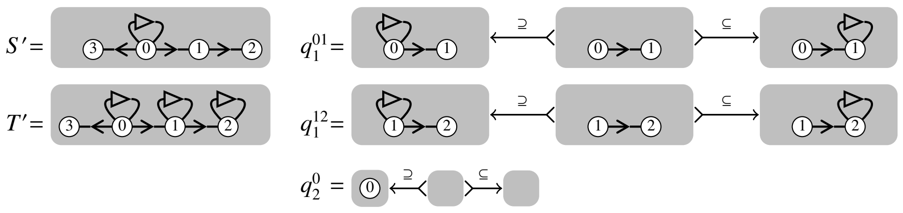


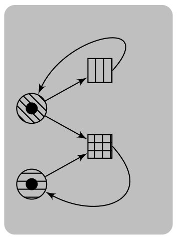
Fig. 13. A process that represents two switch-equivalent derivations

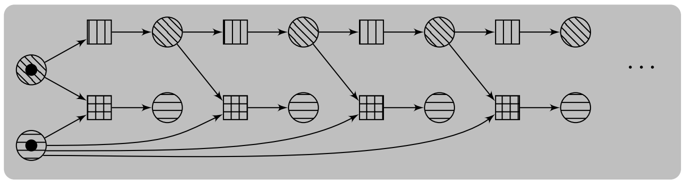
6. The Unfolding Construction

The unfolding of a system records all computations in a single "branching" structure which fully describes the concurrent behaviour of the system itself, including all possible rule occurrences and their mutual dependencies. While processes, which represent single, specific computations of systems are based on deterministic occurrence grammars, in order to capture a class of different computations, unfoldings are usually general (or non-deterministic) occurrence grammars.

Every typed grammar can be *unfolded* by recording all possible sequences of rewriting steps originating from the start object. This is in analogy to the constructions for Petri Nets and graph grammars. One effect, which becomes particularly clear in the case of Petri nets, is the unravelling of causal cycles.

Example 6.1 (Unfolding of Petri nets). Consider the Petri net that is shown on the left below.





(A prefix of) its unfolding is sketched on the right. For each transition in the original net we have several rule occurrences on the right. In the unfolding we also have an infinite computation, which consists of a sequence of infinitely many occurrences of the upper transition of the original net.

The result of the general construction that we shall present is a *(non-deterministic) occurrence grammar* that gives a partial order representation of all possible events and concurrent computations. Finite initial parts of the (full) unfolding of the grammar – so-called *prefixes* or truncations – give a compact representation of the behaviour of the grammar up to a certain *causal depth*.

The idea of the unfolding procedure for a given grammar G is to construct a chain of growing occurrence grammars U_n. Each U_n represents all computations up to causal depth n where the depth of a concurrent computation is the length of a maximally parallel execution of the computation. Finally the full unfolding U_G will arise as the "union" of the chain $\{U_n \text{ “}\subseteq\text{” } U_{n+1}\}_{n\in\mathbb{N}}$. This is a concrete algorithmic description of the unfolding. As shown in the next section, the unfolding can be characterised in a succint and elegant way as the right adjoint functor to the inclusion of the category of occurrence grammars into the category of grammars.

Definition 6.2 (Unfolding Algorithm). Let $G = \langle Q, S \xrightarrow{s} T\rangle$ be a finite grammar. We shall construct a chain U_0"$\subseteq$" U_1"$\subseteq$" ..."$\subseteq$" U_n ... of occurrence grammars $U_n = \langle Q_n, S \xrightarrow{s_n} T_n\rangle$ that come equipped with *folding morphism*s $\mathcal{F}_n \colon U_n \to G$ mapping rule occurrences in each n-th unfolding U_n to the original grammar G; further each $\mathcal{F}_n$ will be induced by a *folding arrow* $T_n \xrightarrow{\lambda_n} T$, i.e., $\mathcal{F}_n = \lambda_n \circ \llcorner\!\lrcorner$ (see Definition 4.1).

Base case. The 0-th unfolding U_0 contains the start object of the grammar G and no rules, i.e., $U_0 = \langle \varnothing, S \rightarrowtail^{\mathrm{id}} S\rangle$, thus $T_0 = S$. The folding arrow is $\lambda_0 = s \colon T_0 \to T$, which induces $\mathcal{F}_0 = \lambda_0 \circ \llcorner\!\lrcorner$.

Induction step. Going from U_n to U_{n+1} consists in adding the next level of causal depth. The central operation of this step can be described as the *non-consuming* application of all rules with all possible (new) matches to T_n "in parallel" – here the non-consuming rule application of a rule $q = L \xleftarrow{\alpha} K \xrightarrow{\beta} R$ at a match $m\colon L \rightarrowtail T$ is the application of $q^+ := K \xleftarrow{\mathrm{id}} K \xrightarrow{\beta} R$ at $m \circ \alpha\colon K \rightarrowtail T$ (see Figure 1(c)).

A *match* or an *occurrence* of a rule $q \in Q$ in the n-th unfolding $\langle Q_n, S \xrightarrow{s_n} T_n\rangle$ via the folding $\mathcal{F}_n$ is a mono $\nu\colon L_q \rightarrowtail T_n$ such that the corresponding subobject $[\nu] \in \mathrm{Sub}(T_n)$ is concurrent and the equation $\lambda_n \circ \nu = l_q$ holds; such a match ν is a *new match* if it is not an occurrence of q that is already present in Q_n, i.e., there is no rule $q' \in Q_n$ such that $\nu = l_{q'}$ and $q = \mathcal{F}_n(q')$ is the image of q' via $\mathcal{F}_n$.

Let $\{\nu_i\colon L_{q_i} \rightarrowtail T_n\}_{i \in I_n}$ be the family of all new matches where the index set $I_n = \{1, \ldots, m\}$ is finite as G is finite and also $\mathrm{Sub}(T_n)$ is finite. Now consider the diagram in Figure 14, which consists of the new matches ν_i and the rule morphisms $\alpha_{q_i}, \beta_{q_i}$ for $i \in I_n$. Take the colimit of

$$\begin{tikzcd}
 & L_{q_m} \arrow[dl, tail, bend right, "\nu_m"'] & K_{q_m} \arrow[l, tail, "\alpha_{q_m}"'] \arrow[r, "\beta_{q_m}"] & R_{q_m} \\
 & \vdots & & \\
T_n & L_{q_1} \arrow[l, tail, "\nu_1"'] & K_{q_1} \arrow[l, tail, "\alpha_{q_1}"'] \arrow[r, "\beta_{q_1}"] & R_{q_1}
\end{tikzcd}$$

Fig. 14. The diagram of new matches

the diagram of new matches in $\mathbf{C}$, which can be obtained by stepwise construction of pushouts of monos (see Figure 15). We obtain morphisms t_n, k_i, r_i into the colimit object T_{n+1} for $i \in I_n$. Furthermore, every object in the colimit diagram is typed over T as we have the folding arrow $\lambda_n\colon T_n \to T$ and each $q_i \in Q$ is a rule of G. Hence, we obtain the folding arrow $\lambda_{n+1}\colon T_{n+1} \to T$ as a mediating arrow.

$$\begin{tikzcd}
 & L_{q_m} \arrow[dl, tail, bend right, "\nu_m"'] & K_{q_m} \arrow[l, tail, "\alpha_{q_m}"'] \arrow[r, "\beta_{q_m}"] \arrow[ddr, bend right, "k_m"] & R_{q_m} \arrow[dd, bend left, "r_m"] & \\
 & \vdots & & & \\
T_n \arrow[rrr, bend right=40, "t_n"'] & L_{q_1} \arrow[l, tail, "\nu_1"'] & K_{q_1} \arrow[l, tail, "\alpha_{q_1}"'] \arrow[r, "\beta_{q_1}"] \arrow[r, bend right, "k_1"'] & R_{q_1} & \mathbf{T_{n+1}} \arrow[r, dashed, "\lambda_{n+1}"] & T \\
\end{tikzcd}$$

$$\begin{tikzcd}
R_{q_1} \arrow[r, "r_1"] & \mathbf{T_{n+1}} & T_n \arrow[l, "t_n"'] & K_{q_1} \arrow[ll, bend left, "k_1"]
\end{tikzcd}$$

Fig. 15. The colimit construction of the diagram of new matches

Now the new rule occurrences form the set $Q'_{n+1} := \{(t_n \circ \nu_i) \xleftarrow{\alpha_{q_i}} k_i \xrightarrow{\beta_{q_i}} r_i \mid i \in I_n\}$ and are at depth level $n+1$; further the complete set Q_{n+1} of rules of U_{n+1} is $Q_{n+1} = Q'_{n+1} \cup t_n \circ (Q_n)$ where $t_n \circ (Q_n) = \{t_n \circ (q) \mid q \in Q_n\}$.

To complete the object part of the (n+1)-th unfolding, we just need to define $U_{n+1} := \langle Q_{n+1}, S \xrightarrow{t_n \circ s_n} T_{n+1}\rangle$. Further the folding morphism $\mathcal{F}_{n+1}\colon U_{n+1} \to G$ is given by $\mathcal{F}_{n+1} := \lambda_{n+1} \circ \llcorner\!\lrcorner$, which is induced by the folding arrow $T_{n+1} \xrightarrow{\lambda_{n+1}} T$.

Lemma 6.3 (Soundness of finite unfoldings). For a given finite grammar G, each n-th unfolding U_n is an occurrence grammar.

Proof. The verification of each property is a fairly straightforward induction with a trivial base case; for the induction step we verify the properties of occurrence grammars as follows where we use the fact that the colimit in Figure 15 can be constructed as an "iterated" pushout.

The type object is the union of all right hand sides since adhesive categories have effective unions (as described in Proposition 2.5). Causality remains acyclic, since the newly added rules are not the cause of any other rule. The causes of each rule obviously form a finite set (as there are only finitely many rules). The start object remains without causes and no backward conflicts are introduced as the colimit is an "iterated" pushout. Finally we can use Proposition 3.7 to show that the (new) left-hand sides are properly produced. □

Summarizing, we have inductively defined a growing sequence of occurrence grammars U_0"$\subseteq$" U_1"$\subseteq$" $\cdots$"$\subseteq$" $U_n \ldots_\infty$, where each U_n has components $\langle Q_n, s_n \colon S \rightarrowtail T_n \rangle$, folding morphisms $\lambda_n \circ \llcorner\!\lrcorner \colon U_n \to G$, and "inclusion" morphisms $t_n \circ \llcorner\!\lrcorner \colon U_n \rightarrowtail U_{n+1}$.

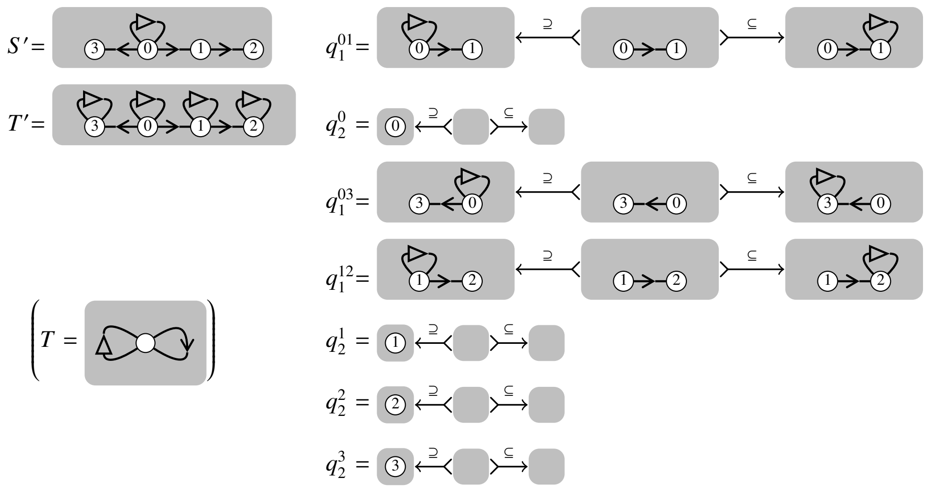


Fig. 16. Unfolding of the running example

Example 6.4. Figure 16 presents the complete unfolding of the typed version of our running example grammar, which stops at a certain causal depth. Note that, if the start object consisted of a ring of nodes, the unfolding construction would never terminate.

The morphism from the type graph T' to the original type graph is again left implicit; it is the one that preserves the kind of edges. The unfolding contains three occurrences of the rule q_1 and four occurrences of the rule q_2. Note that all rule occurrences except for q_1^{12} belong to the level of depth 0 of the unfolding, while q_1^{12} has depth 1, because it causally depends on q_1^{01}.

7. ω-Adhesive Categories and the Coreflection Result

In this section we propose ω-adhesive categories as a framework in which the unfolding construction can be completed and can be characterised as the right adjoint to the inclusion functor from the full sub-category of occurrence grammars into the category of all finite grammars.

As mentioned in Section 6, the unfolding U_G of a grammar G will be a single occurrence grammar that represents the complete chain of truncations generated by the algorithm of Definition 6.2. The colimits that we shall use to construct U_G and to prove the coreflection result are *Van Kampen* (VK) *fans*: they are the ω-chain counterpart of Van Kampen squares, the latter being the central concept in the definition of adhesive categories in (Lack and Sobociński, 2005).

$$\begin{tikzcd}
A_1 \arrow[r, "u_1"] \arrow[rr, bend right, "a_1"'] & A_2 \arrow[r, "u_2"] \arrow[d, "a_2"'] & A_3 \arrow[r, "u_3"] \arrow[dl, "a_3"] & \cdots \\
 & A & \omega\text{-fan} &
\end{tikzcd}$$

$$\begin{tikzcd}
B_1 \arrow[r, "v_1"] \arrow[d, "x_1"] \arrow[drr, "b_1"'] & B_2 \arrow[r, "v_2"] \arrow[d, "x_2"] \arrow[dr, "b_2"] & B_3 \arrow[r, "v_3"] \arrow[d, "x_3"] \arrow[dl, "b_3"] & \cdots \\
 & & B \arrow[dd, "x"] & \\
A_1 \arrow[r, "u_1"] \arrow[drr, "a_1"'] & A_2 \arrow[r, "u_2"] \arrow[dr, "a_2"'] & A_3 \arrow[r, "u_3"] \arrow[d, "a_3"] & \cdots \\
 & & A & \omega\text{-fan prism}
\end{tikzcd}$$

Definition 7.1 (ω-Adhesive Categories). An *ω-fan* is an ω-chain diagram $\mathcal{A} = \{A_n \xrightarrow{u_n} A_{n+1}\}_{n\in\mathbb{N}}$ with a cocone $\alpha = \{A_n \xrightarrow{a_n} A\}_{n\in\mathbb{N}}$ (see the left one of the displayed diagrams); it is a *colimit ω-fan* if α is a colimit of $\mathcal{A}$, and it is a *Van Kampen fan* if in each ω-fan prism over it, as illustrated in the right of the displayed diagrams, having pullback squares $\begin{smallmatrix} B_i & \to & B_{i+1} \\ \downarrow & \lrcorner & \downarrow \\ A_i & \to & A_{i+1} \end{smallmatrix}$ as back faces, the top face is a colimit ω-fan if and only if all lateral trapezia $\begin{smallmatrix} B_i & \to & B \\ & \searrow A_i & \to A \end{smallmatrix}$ are pullbacks.

Now a category is *ω-adhesive* if it is adhesive, and moreover

— it has colimits of monic ω-chains $\{A_n \overset{u_n}{\rightarrowtail} A_{n+1}\}_{n\in\mathbb{N}}$, and

— colimits of monic ω-chains give rise to Van Kampen fans.

From now on, we assume $\mathbf{C}$ to be ω-adhesive. To ensure soundness of the full unfolding construction in Definition 7.3, we need the following lemma, which can be shown in analogy to Lemma 2.3 of (Lack and Sobociński, 2005) (see also (Heindel, 2009, Lemma B.6)).

Lemma 7.2 (Monic VK-fans). Let $\mathbf{C}$ be any category, let $\{A_n \overset{u_n}{\rightarrowtail} A_{n+1}\}_{n\in\mathbb{N}}$ be a monic ω-chain paired with a cocone $\{A_n \xrightarrow{a_n} A\}_{n\in\mathbb{N}}$ such that they together form a Van Kampen fan. Then each $a_i\colon A_n \rightarrowtail A$ is monic.

Definition 7.3 (Full Unfolding). Let $G = \langle Q, s\colon S \to T\rangle$ be a finite grammar, and let the family $\{U_n \overset{t_n \circ \llcorner\!\lrcorner}{\rightarrowtail} U_{n+1}\}_{n\in\mathbb{N}}$ be a chain that is constructed according to Definition 6.2, where $U_n = \langle Q_n, s_n\colon S \rightarrowtail T_n\rangle$ and $t_n\colon T_n \rightarrowtail T_{n+1}$ for each $n \in \mathbb{N}$. To define the *full unfolding U_G*, let $\iota = \{i_n\colon T_n \rightarrowtail T^{U}\}_{n\in\mathbb{N}}$ be the colimit of the ω-chain diagram $\mathcal{T} = \{T_n \overset{t_n}{\rightarrowtail} T_{n+1}\}_{n\in\mathbb{N}}$, and put $U_G := \langle \bigcup_{n\in\mathbb{N}} i_n \circ (Q_n), i_0\colon S \rightarrowtail T^{U}\rangle$.

Finally, to define the *folding morphism* $\mathcal{F}\colon U_G \to G$, let $\lambda_n\colon T_n \to T$ be as in Definition 6.2 for each $n \in \mathbb{N}$. By the universal property of the colimit ι, there is a unique arrow $\lambda\colon T^{U} \to T$ that satisfies $\lambda \circ i_n = \lambda_n$ for all $n \in \mathbb{N}$; now put $\mathcal{F} := \lambda \circ \llcorner\!\lrcorner$.

Lemma 7.4 (Soundness of the Unfolding Construction). The unfolding of a grammar is an occurrence grammar.

Proof. The only non-trivial point concerns the type object, which is required to be the union of all the right hand sides and the start object; this follows from Lemma 7.2. All other properties are inherited from the finite "prefixes" of the full unfolding. □

Proposition 7.5 (Completeness of the Unfolding). Let G be a grammar and $\lambda \circ \llcorner\!\lrcorner\colon U_G \to G$

be the folding morphism from the full unfolding U_G. Then each derivation in G has a unique counterpart in U_G, i.e., for each G-derivation $\mathcal{X}_1 \cdots \mathcal{X}_n$ there is unique U_G-derivation $\mathcal{X}'_1 \cdots \mathcal{X}'_n$ such that $\mathcal{X}_i = \lambda \circ (\mathcal{X}'_i)$ for all $i \in \mathbb{N}$.

Proof. Let $G = \langle Q, s\colon S \to T\rangle$ be a finite grammar. The proof is by induction on the length of derivations. The base case is trivial. Now let $\tilde{\mathcal{X}}\colon s \Longmapsto a$ be a derivation, $q = (l \xleftarrow{\alpha} k \xrightarrow{\beta} r) \in Q$ be a rule and let $\mu\colon l \to a$ be a match for q, which gives rise to a direct derivation $\mathcal{X}\colon a \overset{\langle q,\mu\rangle}{\Longmapsto} b$ in G. Now we have to establish the existence of a unique counterpart of $\tilde{\mathcal{X}}\mathcal{X}$ in U_G.

By the induction hypothesis, the G-derivation $\mathcal{X}$ has a unique counterpart $\tilde{\mathcal{X}}'\colon s' \Longmapsto a'$ in U_G. Now $a' \circ \mu$ is a concurrent subobject and by finiteness of G and the construction of the unfolding there is a unique rule $q' = (l' \leftarrowtail k' \rightarrowtail r')$ of the unfolding such that $l' = a' \circ \mu$ and $\lambda \circ l' = l$; hence applying q' to a' (and chosing the right representatives of subobjects based on $\mathcal{X}$) yields a derivation $\mathcal{X}'$ such that $\mathcal{X} = \lambda \circ (\mathcal{X}')$. Now $\tilde{\mathcal{X}}'\mathcal{X}'$ is the unique counterpart of $\tilde{\mathcal{X}}\mathcal{X}$. □

This proposition is sufficient for many applications, but it does not rule out that U_G might contain superfluous information. The coreflection result ensures that the unfolding with the folding morphism $\mathcal{F}\colon U_G \to G$ is the "minimal" or – more precisely – *universal* choice of an occurrence grammar O and a morphism $\mathcal{H}\colon O \to G$.

Theorem 7.6 (Coreflection). Let $\mathbf{C}$ be an adhesive category in which all existing RPCS are pullback stable (see Definition 2.11), let $T \in \mathbf{C}$, and let G be a finite T-typed grammar (see Definitions 2.26 and 2.21).

Let $\mathcal{F}\colon U_G \to G$ be the folding morphism from the unfolding U_G. Then for each occurrence grammar O and morphism $\mathcal{H}\colon O \to G$ there is a unique morphism $\mathcal{V}\colon O \to U_G$ such that $\mathcal{H} = \mathcal{F} \circ \mathcal{V}$.

$$\begin{tikzcd} U_G \arrow[r, "\mathcal{F}"] & G \\ O \arrow[u, dashed, "\mathcal{V}"] \arrow[ur, "\mathcal{H}"'] & \end{tikzcd}$$

Proof. For the existence and uniqueness of $\mathcal{V}\colon O \to U_G$ where $O = \langle Q', s'\colon S' \to T'\rangle$ we shall use that T' arises as the colimit $\iota' = \{j_n\colon T'_n \rightarrowtail T'\}_{n\in\mathbb{N}}$ of some ω-chain $\{T'_n \xrightarrow{t'_n} T'_{n+1}\}_{n\in\mathbb{N}}$ where $T'_0 = S'$ and each T'_{n+1} arises by gluing the right hand side of a rule to T'_n; more precisely, there is an (injective) enumeration $\{q'_n\}_{n\in\mathbb{N}\setminus\{0\}}$ of Q' that respects causality, i.e., $q'_n < q'_m$ implies $n < m$, such that $j_n = s' \sqcup \bigsqcup_{0<i\leq n} r_{q'_i}$ where $r_{q'_n}$ is the right hand side of q'_n and $j_0 = s'$ (more details can be found in (Heindel, 2009, Lemma D.10)).

Given such a "colimit decomposition" of O, we recursively define an image of j_n in U_G, which we shall call $\mathcal{V}(j_n)$, such that $\mathcal{F}(\mathcal{V}(j_n)) = \mathcal{H}(j_n)$. This growing sequence of images will determine the image of $\mathrm{id}_{T'}$ in the unfolding, which will give the central piece of data for the definition of $\mathcal{V}$.

Now consider $\mathcal{H}(q'_n) = (l \xleftarrow{\alpha} k \xrightarrow{\beta} r) =: q$ in G. We have two cases: Either q is a proper rule or not. In the first case, there is a unique rule $q' = (l' \xleftarrow{\alpha'} k' \xrightarrow{\beta'} r') \in Q^U$ in the unfolding such that $\mathcal{V}(j_{n-1}) \circ |\mathcal{H}(\nu_n)| = l'$ and $\mathcal{F}(q') = q$. Now the idea is that $\mathcal{V}(j_n)$ is determined by $\mathcal{V}(j_{n-1})$, the rule q' and $\mathcal{H}$. Formally, let $j_{n-1} \xrightarrow{t'_{n-1}} j_n \xleftarrow{\rho_n} r_{q'_n}$ be the pushout of $j_{n-1} \xleftarrow{\nu_n \circ \alpha'_n} k_{q'_n} \xrightarrow{\beta'_n} r_{q'_n}$ in $\mathbf{C}{\downarrow}T'$. Now there is a unique $\mathcal{V}(j_n)$ such that $\mathcal{V}(j_{n-1}) \xrightarrow{|\mathcal{H}(t'_{n-1})|} \mathcal{V}(j_n) \xleftarrow{|\mathcal{H}(\rho_n)|} r'$ is the pushout of $\mathcal{V}(j_{n-1}) \xleftarrow{|\mathcal{H}(\nu_n)|\circ\alpha'} k' \xrightarrow{\beta'} r'$ in $\mathbf{C}{\downarrow}T^U$ since $\mathcal{H}$ preserves pushouts along monos. In case that $\mathcal{H}(q'_n) = q$ is an identity span, we can use the same construction as $\mathcal{V}(j_{n-1}) \circ |\mathcal{H}(\nu_n)| =: l'$ is still a concurrent subobject, and $\mathcal{V}(j_{n-1}) \xrightarrow{|\mathcal{H}(t'_{n-1})|} \mathcal{V}(j_n) \xleftarrow{|\mathcal{H}(\rho_n)|} l'$ is again the pushout of $\mathcal{V}(j_{n-1}) \xleftarrow{|\mathcal{H}(\nu_n)|\circ\mathrm{id}} l' \xrightarrow{\mathrm{id}} l'$ (where $\mathcal{H}(t'_{n-1})$ is of course an isomorphism). Also $\mathcal{F}(\mathcal{V}(j_n)) = \mathcal{H}(j_n)$ as $\mathcal{H}$ preserves pushouts along monos.

Finally, the value of $\mathcal{V}(\mathrm{id}_{T'})$ is the unique mediating morphism from $\mathcal{H}(\iota')$. More precisely, reasoning in $\mathbf{C}{\downarrow}T$, the slice category of the type object of G, the object $\mathcal{H}(\mathrm{id}_{T'})$ with the co-projections $\mathcal{H}(\iota') = \{\mathcal{H}(j_n)\}_{n\in\mathbb{N}}$ is the colimit of the ω-chain $\{\mathcal{H}(t'_n)\}_{n\in\mathbb{N}}$; here we have used the VK-fan property and that $\mathcal{H}$ is a retyping operation. Moreover the familiy $\{\mathcal{V}(j_n)\}_{n\in\mathbb{N}}$ yields a co-cone to λ (again considered as an object of $\mathbf{C}{\downarrow}T$). Hence there is a unique mediating morphism from $u\colon \mathcal{H}(\mathrm{id}_{T'}) \to \lambda$ in $\mathbf{C}{\downarrow}T$ such that $u \circ \mathcal{H}(j_n) = \mathcal{V}(j_n)$; now $|u|$ is the value of $\mathcal{V}(\mathrm{id}_{T'})$.

In order to turn $\mathcal{V}$ into a functor, note that any object $f \in \mathbf{C}{\downarrow}T'$ can be considered as a morphism $f'\colon F \to \mathrm{id}_{T'}$, which we map to $\mathcal{V}(f) := |\mathcal{V}(\mathrm{id}_{T'})| \circ |\mathcal{H}(f')|$; further for each morphism $\phi\colon f \to g$, $\mathcal{V}(\phi)$ is given by $|\mathcal{H}(\phi)|$. However $\mathcal{V}$ is actually a retyping operation, as every cartesian transformation $\vartheta\colon |\llcorner\!\lrcorner|_T \circ \mathcal{H} \to |\llcorner\!\lrcorner|_{T'}$ yields a cartesian transformation $\vartheta'\colon |\llcorner\!\lrcorner|_{T^U} \circ \mathcal{V} \to |\llcorner\!\lrcorner|_{T'}$ where $\vartheta'(u) = \vartheta(\lambda \circ u)$.

The proof for uniqueness of $\mathcal{V}$ is similar. Let $\mathcal{W}\colon O \to U_G$ be a grammar morphism that satisfies $\mathcal{F} \circ \mathcal{W} = \mathcal{H}$. Now $\mathcal{W}$ must coincide with $\mathcal{V}$ on s' and on each $r_{q'_n}$, i.e., $\mathcal{W}(s') = \mathcal{V}(s')$ and $\mathcal{W}(r_{q'_n}) = \mathcal{V}(r_{q'_n})$ for all n. The equality $\mathcal{W}(s') = \mathcal{V}(s')$ follows directly from the definition of grammar morphism. For each rule $q'_n \in Q'$, the equality $\mathcal{W}(r_{q'_n}) = \mathcal{V}(r_{q'_n})$ is shown by induction on the length of the shortest derivation that applies q'_n. Here we use that the left hand side $l_{q'_n}$ is properly produced by right hand sides of rules with shorter shortest derivations. By induction hypothesis, $\mathcal{V}$ and $\mathcal{W}$ coincide on these right hand sides. If $\mathcal{H}(q'_n)$ is not a rule in G then we have $\mathcal{W}(r_{q'_n}) = \mathcal{V}(r_{q'_n})$ by the definition of grammar morphism. Otherwise, $\mathcal{V}(l_{q'_n}) = \mathcal{W}(l_{q'_n}) =: \bar{l}$ is a concurrent subobject with a unique rule $q' = (l' \leftarrowtail k' \rightarrowtail r') \in Q^U$ such that $\mathcal{F}q' = \mathcal{H}(q)$ and $l' = \bar{l}$. This implies that $\mathcal{V}(q'_n) = q' = \mathcal{W}(q'_n)$ and in particular $\mathcal{W}(r_{q'_n}) = \mathcal{V}(r_{q'_n})$.

As a consequence of the previous considerations with the help of a colimit decomposition of O, we show that $\mathcal{W}$ and $\mathcal{V}$ must also coincide on $\mathrm{id}_{T'}$ as they preserve colimits of ω-chains of monos. The equality $\mathcal{W} = \mathcal{V}$ is shown by using again the observation that each $f \in \mathbf{C}{\downarrow}T'$ can be seen as a morphism to the final object $\mathrm{id}_{T'}$ and that for each morphism $\phi\colon f \to g$ we have $|\mathcal{W}(\phi)| = |\mathcal{H}(\phi)| = |\mathcal{V}(\phi)|$ (see also (Heindel, 2009, Lemma E.12, Proposition E.15)). $\square$

This theorem directly implies that the unfolding construction extends to a functor from the category of finite grammars to that of occurrence grammars, which in turn means that the category of occurrence grammars is a coreflective subcategory of the category of finite grammars.

Example 7.7 (Coreflection). Figure 17 shows our example net N and its unfolding U_N, given earlier. The occurrence net O of Example 4.5 can be mapped to N. Then, by Theorem 7.6, there exists a unique morphism from O to U_N making the triangle commuting. This illustrates the fact that the unfolding is in a sense the most general representation of the original net in the domain of occurrence nets (or occurrence grammars).

Remark 7.8 (Relation to Petri Nets and their Unfolding). We mentioned before that typed grammars in the category **Set** are closely related to Petri nets, although objects and morphisms are, in a sense, more concrete. More precisely, there is a natural choice of a functor from the category of typed grammars over **Set** to the category of Petri nets (with read arcs) (see, e.g., (Winskel, 1987a; Baldan, 2000)). In fact each typed set $c\colon C \to P$ yields a P-multiset $m_c\colon P \to \mathbb{N}$ which maps each $p \in P$ to the size of its inverse image, i.e., $|c^{-1}(p)|$. Given a typed set grammar, the type object becomes the set of places, the start object $s\colon S \to P$ gives a multiset in the described way, each rule is the name of a transition with the obvious pre-set, post-set and

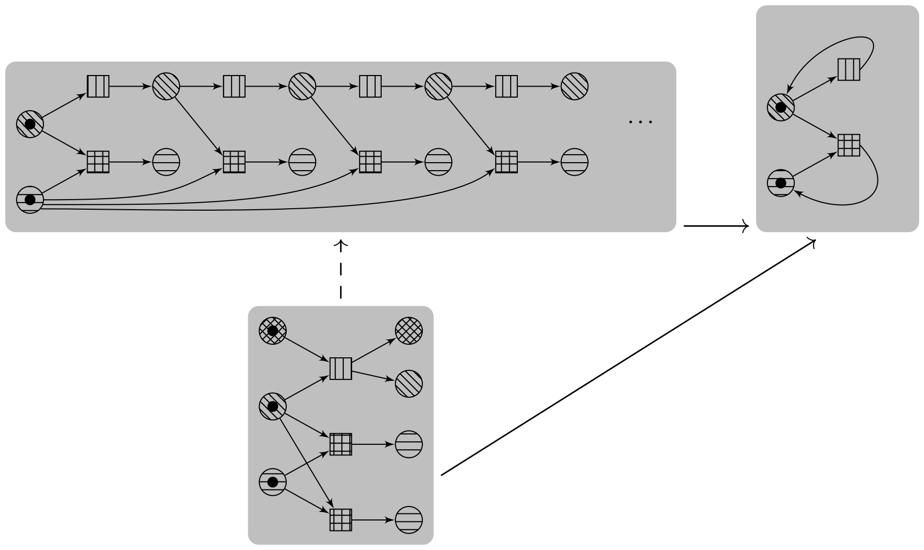


Fig. 17. The coreflection result, illustrated.

read-arcs. Each grammar morphism $\mathcal{F}$, which is a retyping operation from **Set**$\downarrow P$ to **Set**$\downarrow P'$, is mapped to the multi-relation that maps each $(p, p') \in P \times P'$ to $|\mathcal{F}(p)^{-1}(p')|$ where $\mathcal{F}(p)$ is the object of **Set**$\downarrow P'$ to which the inclusion $p \subseteq P$ is mapped by $\mathcal{F}$. Call this functor the *multiset abstraction*.

The multiset abstraction is surjective on objects and full, but neither injective on objects nor faithful; however, any two grammars that are mapped to the same net are isomorphic and also two grammar morphisms that are mapped to the same net-morphism are naturally isomorphic. Thus, the functor has an "inverse functor", which is essentially surjective and "essentially full". Thus (using the axiom of choice) we have an embedding of Petri nets to typed set rewriting and the multiset-abstraction is "almost" its inverse. Finally, at least for the case of safe Petri nets, the usual unfolding functor that maps into the categoy of occurrence nets factors as the embedding to SPO **Set**-rewriting, followed by the unfolding as described in the present paper and the multi-set abstraction.

It is worth stressing that assuming that category **C** is ω-adhesive we could have relaxed the assumption on grammar G from finite to countable.

As for examples of ω-adhesive categories: any elementary topos is ω-adhesive if and only if it has countable sums (as a consequence of Theorem 9.8 and Theorem B.11 of (Heindel, 2009)). Hence the category of sets is ω-adhesive, and more generally any Grothendieck topos is. Instead, the category **S** of *finite* sets – the "primordial" elementary topos – is a counterexample, i.e., it is not ω-adhesive. Further examples arise via the following constructions.

Proposition 7.9 (Closure of ω-Adhesivity). Let **C** and **D** be ω-adhesive categories. Then the following categories are again ω-adhesive:

— the product category **C**×**D**;

— the slice category $\mathbf{C}{\downarrow}T$ for any $T \in \mathbf{C}$;
— the co-slice category $I{\downarrow}\mathbf{C}$ for any $I \in \mathbf{C}$;
— the functor category $[\mathbf{X}, \mathbf{C}]$ for any category $\mathbf{X}$;
— the Artin-Wraith glueing $\mathbf{C}{\downarrow}\mathcal{F}$, i.e., the comma category $\mathbf{C}{\downarrow}\mathcal{F}$ for any functor $\mathcal{F}\colon \mathbf{D} \to \mathbf{C}$ that preserves pullbacks.

Proof sketch. Pullbacks and the relevant colimits are constructed componentwise. □

In addition we know that the slice construction preserves stability of pseudo-complementation. Note also that all examples of "graph categories" mentioned in the introduction (undirected and directed graphs, hypergraphs, graphs with scopes, graphs with second-order edges, etc.) are ω-adhesive and RPCs are stable under pullback.

8. Conclusion

The central contribution of this paper is the generalization of several standard concepts, constructions and results about concurrent computations of concrete systems like Petri nets and graph transformation systems to the abstract setting of rule-based, single pushout systems over ω-adhesive categories. In particular, we considered the construction of a process from a deterministic computation, and the construction of the full unfolding of a grammar; we showed that processes are in one-to-one correspondence with switch-equivalent classes of derivations, and that the unfolding construction "unravelling" a grammar into an occurrence grammar can be characterised as a coreflection.

Concerning the choice of the rewriting approach, we preferred to stick to the SPO approach in this paper just to keep the presentation simpler. In fact, the alternative DPO approach was actually adopted in (Baldan et al., 2006) for the abstract definition of processes. Also the unfolding construction could have been presented for DPO rewriting along the lines followed for inhibitor nets in (Baldan, Busi, Corradini and Pinna, 2004) and for DPO graph transformations in (Baldan, 2000), at the price of some additional complications. Processes and unfolding-like structures have also been studied in the setting of subobject transformation systems (Corradini, Hermann and Sobociński, 2008), incorporating negative application conditions (Hermann, Corradini, Ehrig and König, 2010) as well.

As mentioned in Section 2.1, there exist several variations of adhesive categories that arise by weakening the conditions of Definition 2.1. A natural question to ask is whether the results of the present paper could be generalised, for instance, to partial map adhesive categories (Heindel, 2010). This is in principle feasible (see (Heindel, 2009)), but it would be necessary to impose additional conditions that hold in the relevant example categories, but are ad hoc. Hence we decided to refrain from greater generality in order to obtain a simpler presentation. But we would like to state that the presented constructions and results can also be applied to the non-adhesive categories **sGraph** and **Top** of Example 2.2 (Heindel, 2009).

We have introduced a new notion of grammar morphisms where the retyping is given by a functor. This allows us to treat also non-semi-weighted grammars, i.e., grammars where the start graph or the right-hand sides of rules might not be monos. Otherwise technical complications arise because of the presence of "too much symmetry" in the structure which is being unfolded and hence the uniqueness of arrow $\mathcal{V}$ in Proposition 7.5 cannot be guaranteed. Another solution to the symmetry problem has been proposed in (Hayman and Winskel, 2008), for the case of Petri nets and with a different notion of morphisms.

The unfolding represents all computations as well as all reachable objects of the original grammar in a single acyclic branching structure. Hence, as observed in (McMillan, 1993; Baldan, Corradini and König, 2008), it can serve as the basis for partial order verification techniques. For instance, we plan to generalise the notion of finite complete prefix to the abstract framework of the present paper. Another direction is to adapt the model-based diagnosis techniques of (Benveniste et al., 2003; Baldan et al., 2010); the latter depend on the preservation of products of grammars by the unfolding functor, which is ensured by the coreflection result. To this aim, in future work we shall further investigate the existence of products and pullbacks in our category of grammar morphisms.

References

Baldan, P. (2000). *Modelling Concurrent Computations: from Contextual Petri Nets to Graph Grammars*, PhD thesis, Dipartimento di Informatica, Università di Pisa.

Baldan, P., Busi, N., Corradini, A. and Pinna, G. M. (2004). Domain and event structure semantics for Petri nets with read and inhibitor arcs, *Theor. Comput. Sci.* **323**(1-3): 129–189.

Baldan, P., Chatain, T., Haar, S. and König, B. (2010). Unfolding-based diagnosis of systems with an evolving topology, *Information and Computation* **208**(10): 1169–1192.

Baldan, P., Corradini, A., Heindel, T., König, B. and Sobociński, P. (2006). Processes for adhesive rewriting systems, *in* L. Aceto and A. Ingólfsdóttir (eds), *FoSSaCS*, Vol. 3921 of *LNCS*, Springer, pp. 202–216.

Baldan, P., Corradini, A., Heindel, T., König, B. and Sobocinski, P. (2009). Unfolding grammars in adhesive categories, *in* A. Kurz, M. Lenisa and A. Tarlecki (eds), *CALCO*, Vol. 5728 of *LNCS*, Springer, pp. 350–366.

Baldan, P., Corradini, A. and König, B. (2008). A framework for the verification of infinite-state graph transformation systems, *Information and Computation* **206**: 869–907.

Baldan, P., Corradini, A. and Montanari, U. (2001). Contextual Petri nets, asymmetric event structures, and processes, *Information and Computation* **171**(1): 1–49.

Baldan, P., Corradini, A., Montanari, U. and Ribeiro, L. (2007). Unfolding Semantics of Graph Transformation, *Information and Computation* **205**: 733–782.

Benveniste, A., Fabre, E., Haar, S. and Jard, C. (2003). Diagnosis of asynchronous discrete event systems, a net unfolding approach, *IEEE Transactions on Automatic Control* **48**(5): 714–727.

Birkhoff, G. (1967). *Lattice Theory*, American Mathematical Society.

Braatz, B., Ehrig, H., Gabriel, K. and Golas, U. (2010). Finitary $\mathcal{M}$-adhesive categories, *in* Ehrig, Rensink, Rozenberg and Schürr (2010), pp. 234–249.

Cockett, R. and Guo, X. (2007). Join restriction categories and the importance of being adhesive. Unpublished manuscript, slides from CT 07.

Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L. and Rozenberg, G. (eds) (2006). *Graph Transformations, Third International Conference, ICGT 2006, Natal, Rio Grande do Norte, Brazil, September 17-23, 2006, Proceedings*, Vol. 4178 of *LNCS*, Springer.

Corradini, A., Heindel, T., Hermann, F. and König, B. (2006). Sesqui-pushout rewriting, *in* Corradini, Ehrig, Montanari, Ribeiro and Rozenberg (2006), pp. 30–45.

Corradini, A., Hermann, F. and Sobociński, P. (2008). Subobject transformation systems, *Applied Categorical Structures* **16**(3): 389–419.

Corradini, A., Montanari, U. and Rossi, F. (1996). Graph processes, *Fundamenta Informaticae* **26**: 241–265.

Dyckhoff, R. and Tholen, W. (1987). Exponentiable morphisms, partial products and pullback complements, *Journal of Pure and Applied Algebra* **49**: 103–116.

Ehrig, H. (1979). Introduction to the algebraic theory of graph grammars, *in* V. Claus, H. Ehrig and G. Rozenberg (eds), *Graph-Grammars and Their Application to Computer Science and Biology*, Vol. 73 of *LNCS*, Springer, pp. 1–69.

Ehrig, H., Ehrig, K., Prange, U. and Taentzer, G. (2006). *Fundamentals of Algebraic Graph Transformation*, EATCS Monographs in Theor. Comp. Science, Springer.

Ehrig, H., Golas, U. and Hermann, F. (2010). Categorical Frameworks for Graph Transformation and HLR Systems based on the DPO Approach, *Bulletin of the EATCS* **102**: 111–121.

Ehrig, H., Habel, A., Kreowski, H.-J. and Parisi-Presicce, F. (1991). Parallelism and Concurrency in High-Level Replacement Systems, *Mathematical Structures in Computer Science* **1**: 361–404.

Ehrig, H., Pfender, M. and Schneider, H. (1973). Graph-grammars: an algebraic approach, *Proc. of IEEE Conf. on Automata and Switching Theory*, pp. 167–180.

Ehrig, H., Rensink, A., Rozenberg, G. and Schürr, A. (eds) (2010). *Graph Transformations - 5th International Conference, ICGT 2010, Enschede, The Netherlands, September 27 - - October 2, 2010. Proceedings*, Vol. 6372 of *LNCS*, Springer.

Goltz, U. and Reisig, W. (1983). The Non-sequential Behaviour of Petri Nets, *Information and Computation* **57**: 125–147.

Habel, A., Müller, J. and Plump, D. (2001). Double-pushout graph transformation revisited., *Mathematical Structures in Computer Science* **11**(5): 637–688.

Hayman, J. and Winskel, G. (2008). The unfolding of general Petri nets, *Proc. of FSTTCS '08*, number 2 in *Leibniz International Proceedings in Informatics*.

Heindel, T. (2009). *A Category Theoretical Approach to the Concurrent Semantics of Rewriting: Adhesive Categories and Related Concepts*, PhD thesis, Universität Duisburg-Essen.

Heindel, T. (2010). Hereditary Pushouts Reconsidered, *in* Ehrig, Rensink, Rozenberg and Schürr (2010), pp. 250–265.

Heindel, T. and Sobociński, P. (2009). Van Kampen colimits as bicolimits in Span, *Proc. of CALCO '09*, Vol. 5728 of *LNCS*, Springer, pp. 335–349.

Hermann, F., Corradini, A., Ehrig, H. and König, B. (2010). Efficient analysis of permutation equivalence of graph derivations based on Petri nets, *Proc. of GT-VMT '10 (Workshop on Graph Transformation and Visual Modeling Techniques)*, Vol. 29 of *Electronic Communications of the EASST*.

Johnstone, P. (2002). *Sketches of an Elephant*, Vol. 1, Oxford Science Publications.

Johnstone, P., Lack, S. and Sobociński, P. (2007). Quasitoposes, quasiadhesive categories and Artin glueing, *Proc. CALCO'07*, Vol. 4626 of *LNCS*, Springer, pp. 312–326.

Lack, S. and Sobociński, P. (2005). Adhesive and quasiadhesive categories, *Theoretical Informatics and Applications* **39**(2): 511–546.

Lack, S. and Sobociński, P. (2006). Toposes are adhesive, *in* Corradini, Ehrig, Montanari, Ribeiro and Rozenberg (2006), pp. 184–198.

Löwe, M. (1993). Algebraic approach to single-pushout graph transformation, *Theoretical Computer Science* **109**: 181–224.

Löwe, M. (2010). Graph rewriting in span-categories, *in* Ehrig, Rensink, Rozenberg and Schürr (2010).

McMillan, K. (1993). *Symbolic Model Checking*, Kluwer.

Meseguer, J., Montanari, U. and Sassone, V. (1997). On the semantics of Place/Transition Petri nets, *Mathematical Structures in Computer Science* **7**(4): 359–397.

Robinson, E. and Rosolini, G. (1988). Categories of partial maps, *Information and Computation* **79**(2): 95–130.

Vogler, W., Semenov, A. and Yakovlev, A. (1998). Unfolding and finite prefix for nets with read arcs, *in* D. Sangiorgi and R. de Simone (eds), *Proc. of CONCUR '98*, Vol. 1466 of *LNCS*, Springer, pp. 501–516.

Winskel, G. (1987a). Event structures, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, Vol. 255 of *LNCS*, Springer, pp. 325–392.

Winskel, G. (1987b). Petri nets, algebras, morphisms, and compositionality, *Information and Computation* **72**(3): 197–238.