On categorical graph grammars integrating structural transformations and operations on labels*

H.J. Schneider

Lehrstuhl für Programmiersprachen der Universität Erlangen–Nürnberg, Martensstraße, 3, W-8520 Erlangen, Germany

Abstract

Schneider, H.J., On categorical graph grammars integrating structural transformations and operations on labels, Theoretical Computer Science 109 (1993) 257–274.

Graph-theoretic structures are an obvious means to reason about systems of asynchronous processes. Their dynamic behaviour can be simulated by applying productions of a graph grammar. The present paper is motivated by looking for a formal method that is able to describe the behaviour of systems of processes that share data structures. We generalize the categorical graph-grammar approach by labelling the graphs with elements of a suitable category rather than with those of an alphabet. Thus, operations can be performed on the labels while the graphs are rewritten. After presenting the fundamental definitions and some properties, we demonstrate the usefulness of the approach by modelling some well-known Petri nets as well as a generalized net the places of which are labelled with graphs. Finally, we show that known theoretical techniques are applicable to the generalized framework by exemplary discussing parallel independence of derivation steps.

1. Introduction

Graph-theoretic structures are an obvious means to reason about a lot of things in computer science. Whereas the traditional graph theory is mainly concerned with static properties of graphs, computer science applications also include dynamically changing the structure. Different types of graph grammars have been developed to describe the dynamic behaviour of graph structures. Each step corresponds to

Correspondence to: H.J. Schneider, Lehrstuhl für Programmiersprachen der Universität Erlangen–Nürnberg, Martensstraße 3, W-8520 Erlangen, Germany.
* This work was partly supported by Deutsche Forschungsgemeinschaft (SFB 182 – Project B1).

applying a production, the left-hand side of which defines the applicability condition, i.e., a subgraph and its labelling. By applying the production, we replace an occurrence of its left-hand side by the right-hand side to alter the structure of the graph. In the present paper, we continue a line of research in which some authors have already tried to overcome the limitations of working with label-preserving graph morphisms. A first approach to the problem of relabelling was given by Ehrig et al. [6] in 1981. We generalize an idea due to Parisi-Presicce et al. [11]; these authors imposed a simple structure on the set of labels to allow variables in the productions. The approach can easily be extended: Labelling is done in categories rather than in an alphabet. In this way, operations can be performed on the labels while the graphs are rewritten.

The usefulness of our approach is demonstrated by a quite natural modelling of condition/event nets and place/transition nets in the new framework. Although in this paper we model only the token game, it is clear that the graph-grammar approach is also suited to model dynamic systems of processes which can grow and shrink.[1] As far as we know, Wileden [17] was the first to point out that firing in a Petri net can be interpreted in the framework of graph grammars. His approach as well as Reisig's [14] use arithmetics to manage the number of tokens on the places. Kreowski [8] added a bunch of special edges to each place indicating the number of tokens and the remaining capacity.

In Section 2, we present the fundamental definitions and prove the properties we need. In Section 3, we apply the approach to two well-known types of Petri nets and show that our framework also covers a generalization allowing places to be labelled with graphs. This case is motivated by studying systems of processes which share data structures. Finally, we exemplary characterize the parallel independence of derivation steps in the generalized framework.

2. Labelled graph grammars

2.1. Definitions

We use the categorical graph-grammar approach as it is outlined, e.g., in Ehrig's introductory paper [2]. This approach allows us to take full advantage of some theoretical results. Ehrig et al. [3], Kreowski [9] and Kreowski and Wilharm [10] have extensively studied the problem of applying productions concurrently or in parallel.

Definition 2.1. A **C**-production is a pair of morphisms $p=({}'B \xleftarrow{{}'p} K \xrightarrow{p'} B')$ in a suitable category **C**.

The objects ${}'B$ and B' are called the left-hand side and the right-hand side of the production, respectively. K is the so-called gluing object; in the special case of a graph

[1] An example describing a parallel sorting algorithm by a graph grammar will be given in a forthcoming paper.

production, i.e., **C** is the category of graphs, it defines the boundary connecting the left-hand side (after applying the production: the right-hand side) to the context.

Definition 2.2. Given a **C**-production p, we say that G' is derivable from $'G$ with respect to p: $'G \overset{p}{\Rightarrow} G'$ if and only if there exist an object D and a morphism $K \to D$ such that both parts of Diagram 1 are pushout diagrams.

D is called the context object. If **C** is the category of graphs, we may interpret it as the context that remains unchanged in applying the production.

Usually, the category **C** is assumed to have pushouts; in this case, the diagram can uniquely be completed if the production and the morphism $K \to D$ are given. On the other hand, however, G' is not unambiguously defined by the production and the morphism $'B \to 'G$. It is possible that different context objects exist (and, thus, different G') or that there is no context object at all. The category of graphs is an example where this may happen. In our original paper on graph grammars, we have already given a gluing condition ensuring the existence of a context object, and we could show that it is unambiguous if $'p$ is injective [7]. This criterion has led Raoult [13] to a new approach replacing $'p$ by an inverse (partial) graph morphism $'p^{-1}$ and then using only one pushout construction. Van den Broek [1] compared the generative power of both approaches.

The definition also makes sense if we assume that only the required pushout objects exist. Of course, we are not allowed to use all theorems in this case without carefully taking into account the consequences of weakening the assumption.[2]

We now consider categories of labelled graphs.

Definition 2.3. A graph G is a quadruple $G = (E, V, s, t)$, with E and V being finite sets and $s, t : E \to V$ being mappings.

E and V denote the set of edges and nodes (vertices), respectively. $s(e)$ denotes the source node of an edge e and $t(e)$ its target node. Considering more than one graph, we

$$\begin{tikzcd}
'B \arrow[d] & K \arrow[l, "'p"'] \arrow[r, "p'"] \arrow[d] & B' \arrow[d] \\
'G & D \arrow[l] \arrow[r] & G'
\end{tikzcd}$$

Diagram 1.

[2] The category S-graph mentioned below is an example of such a category.

distinguish their constituents E, V, s and t by indices or by apostrophes referring to the denotations of the graphs, e.g., E_G is the set of edges of graph G, l_{EG} is the labelling of E_G, etc.

Definition 2.4. Let G and H be graphs. A graph morphism $f:(E_G, V_G, s_G, t_G)\rightarrow(E_H, V_H, s_H, t_H)$ is a pair $f=(f_E: E_G\rightarrow E_H,\ f_V: V_G\rightarrow V_H)$ of set morphisms such that $f_V s_G = s_H f_E \wedge f_V t_G = t_H f_E$.

Limits and colimits in the category of graphs can separately be constructed for nodes and edges in the category of sets.

In what follows, we generally assume that $\mathbf{L}=(\mathbf{L}_E, \mathbf{L}_V)$ is a pair of (small) categories the object sets of which are denoted by L_E (edge labels) and L_V (node labels), respectively.

Definition 2.5. An $\mathbf{L}$-labelled graph is a sextuple $\hat{G}=(E, V, s, t, l_E, l_V)$, where $G=(E, V, s, t)$ is a graph and $l_E: E\rightarrow L_E$, $l_V: V\rightarrow L_V$ are mappings.

G is called the underlying graph.

Usually, morphisms in the category of labelled graphs preserve labels. Parisi-Presicce et al. [11] have extended this approach by imposing a simple structure on the set of labels: Graph morphisms must be compatible with this structure. Their approach is a special case of the following definition.

Definition 2.6. Let $\hat{G}$ and $\hat{H}$ be $\mathbf{L}$-labelled graphs. An $\mathbf{L}$-graph morphism $\hat{f}: \hat{G}\rightarrow\hat{H}$ is defined by
- a graph morphism $f: G\rightarrow H$ between the underlying graphs,
- an $\mathbf{L}_E$-morphism $\bar{f}_e: l_{EG}(e)\rightarrow l_{EH}(f_E(e))$ for each e in E_G,
- an $\mathbf{L}_V$-morphism $\bar{f}_v: l_{VG}(v)\rightarrow l_{VH}(f_V(v))$ for each v in V_G.

Sometimes, we omit the indices E and V to simplify our formulae; they hold both for nodes and for edges. Furthermore, we do not need edge labels in our discussion of Petri nets, i.e., all edges are labelled by the same object.

2.2. *Colimits in the category of* L*-graphs*

The set of $\mathbf{L}$-graphs together with $\mathbf{L}$-graph morphisms is a category. The composition of two $\mathbf{L}$-graph morphisms

$$\hat{f}: \hat{G}_1\rightarrow\hat{G}_2 \quad\text{and}\quad \hat{g}: \hat{G}_2\rightarrow\hat{G}_3$$

is given by the composition $g\cdot f$ of the underlying graph morphisms, together with the labelling morphisms $\bar{g}_{f(x)}\cdot\bar{f}_x$, where x denotes a node or an edge of G_1. The identities in **Graph** turn to identities in **L-Graph** if we use the identities of $\mathbf{L}$ to map the labels.

Lemma 2.7. *The initial object of* **L-Graph** *is the empty graph.*

Proof. The initial object is characterized by the fact that there is exactly one morphism from it to any other object of the category. The only possibility to map the empty graph into a nonempty one is the empty mapping. □

Lemma 2.8. L-Graph *has coproducts.*

Proof. Let $\hat{G}'$ and $\hat{G}''$ be two L-graphs and G be the coproduct of the underlying graphs G' and G'' in **Graph**. It is well known that this coproduct as well as other limits and colimits can be constructed for nodes and edges separately. V_G and E_G are labelled by using the coproduct property in **Set**: i.e., in the category of sets, there is exactly one morphism $l_{VG}: V_G \to L_V$ such that the triangles in Diagram 2 commute, and the same holds true for the edges. (Please note that this lemma does not assume existence of coproducts in **L**.) To show the coproduct property in **L-Graph**, we have to consider any two morphisms $V_{G'} \to V_H$ and $V_{G''} \to V_H$ and to prove the existence of a unique morphism $f_V: V_G \to V_H$ factorizing the given morphisms. This holds true for the underlying graphs, and the coproduct property of V_G yields uniqueness of $l_{VG} = l_{VH} \cdot f_V$. This means that f is a morphism in **L-Graph**. □

Lemma 2.9. *If* **L** *has all finite colimits, then* **L-Graph** *has co-equalizers.*

Proof. Let $\hat{f}, \hat{g}: \hat{G}' \to \hat{G}''$ be a pair of parallel L-graph morphisms. We have to construct a morphism $\hat{q}$ in such a way that for any $\hat{h}$ with $\hat{h} \cdot \hat{f} = \hat{h} \cdot \hat{g}$, there exists a unique $\hat{h}'$ making Diagram 3 commute: We first construct $q: G'' \to G$ as the co-equalizer of the

$$\begin{tikzcd}
V_{G'} \arrow[r, "g_V"] \arrow[dr, "l_{VG'}"'] & V_G \arrow[d, "l_{VG}"] & V_{G''} \arrow[l, "h_V"'] \arrow[dl, "l_{VG''}"] \\
 & L_V &
\end{tikzcd}$$

Diagram 2.

$$\begin{tikzcd}
\hat{G}' \arrow[r, shift left, "\hat{f}"] \arrow[r, shift right, "\hat{g}"'] & \hat{G}'' \arrow[r, "\hat{q}"] \arrow[dr, "\hat{h}"'] & \hat{G} \arrow[d, "\hat{h}'"] \\
 & & \hat{H}
\end{tikzcd}$$

Diagram 3.

underlying graph morphisms f, g, i.e., separately for nodes and edges in the category of sets. In a second step, we decide on how to label G. We have to define l_{VG} and l_{EG} and, furthermore, morphisms

$$\bar{q}_{v''}: l_{VG''}(v'') \to l_{VG}(q_V(v''))$$

$$\bar{q}_{e''}: l_{EG''}(e'') \to l_{EG}(q_E(e''))$$

for all nodes v'' and edges e'' of graph G''. In addition, we have to choose the l_{VG} and l_{EG} such that existence of unique morphisms

$$\bar{h}'_v: l_{VG}(v) \to l_{VH}(h'_V(v))$$

$$\bar{h}'_e: l_{EG}(e) \to l_{EH}(h'_E(e))$$

is ensured. Therefore, a node v of G must be labelled with the colimit of the diagram that consists of all morphisms

$$\{\bar{f}_{v'}: l_{VG'}(v') \to l_{VG''}(v'') \mid q(v'') = v\} \cup \{\bar{g}_{v'}: l_{VG'}(v') \to l_{VG''}(v'') \mid q(v'') = v\}.$$

This colimit exists because of the assumption. Edge labels are constructed analogously. □

In general, the diagram of which we have to construct the colimit may be rather complicated even if the underlying graph morphisms are injective. An example may illustrate this: We assume that

$$f(x'_1) = x''_1,$$

$$g(x'_1) = f(x'_2) = x''_2,$$

and

$$g(x'_2) = x''_3.$$

Then, construction of the co-equalizer yields

$$q(x''_1) = q(x''_2) = q(x''_3) = x$$

and we have to label x such that $l_G(x)$ together with $\bar{q}_{x''_1}, \bar{q}_{x''_2}$ and $\bar{q}_{x''_3}$ is the colimit of Diagram 4 consisting of $\bar{f}_{x'_1}, \bar{g}_{x'_1}, \bar{f}_{x'_2}$ and $\bar{g}_{x'_2}$. This difficulty arises from labelling the graph with objects of a category. In the special case mentioned by Parisi-Presicce et al. [11], $l_G(x)$ is given by the greatest lower bound of all $l_{G''}(x'')$ with $q(x'') = x$.

Theorem 2.10. *If the labelling category* **L** *has all finite colimits, then* **L-Graph** *has all finite colimits, too.*

Proof. It is well known that a category that has initial object, coproducts and co-equalizers has all finite colimits. □

$$\begin{tikzcd}
l_{G'}(x'_1) \arrow[r, "\bar f_{x'_1}"] \arrow[ddr, "\bar g_{x'_1}"] & l_{G''}(x''_1) \arrow[ddr, "\bar q_{x''_1}"] & \\
 & = & \\
l_{G'}(x'_2) \arrow[r, "\bar f_{x'_2}"] \arrow[ddr, "\bar g_{x'_2}"] & l_{G''}(x''_2) \arrow[r, "\bar q_{x''_2}"] & l_G(x) \\
 & = & \\
 & l_{G''}(x''_3) \arrow[uur, "\bar q_{x''_3}"] &
\end{tikzcd}$$

Diagram 4.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, "q"] \\
C \arrow[r, "p"] & D
\end{tikzcd}$$

Diagram 5.

$$\begin{tikzcd}
l_A(a) \arrow[r, "\bar f_a"] \arrow[d, "\bar g_a"'] & l_B(f(a)) \arrow[d, "\bar q_{f(a)}"] \\
l_C(g(a)) \arrow[r, "\bar p_{g(a)}"] & l_D(d)
\end{tikzcd}$$

Diagram 6.

The situation is simpler if we restrict consideration to those pushouts that are of special importance in many applications:

Theorem 2.11. *Let* **L** *be a category that has pushouts. If $\hat f:\hat A\to\hat B$ and $\hat g:\hat A\to\hat C$ are* **L-Graph** *morphisms the underlying graph morphisms of which are injective, then the pushout $\hat q\cdot\hat f=\hat p\cdot\hat g$ exists in* **L-Graph**.

Proof. First, we construct the pushout[3] (Diagram 5) in **Graph**. It is well known that this can separately be done for nodes and edges in the category of sets and that p and q are injective, too. In a second step, we label D in a suitable way:

(a) If $d\in qf[A]=pg[A]$, then $l_D(d)$ is the pushout object in Diagram 6, where $a\in A$ is the unique element with $d=q(f(a))=p(g(a))$.

(b) If $d\notin q[B]\ \wedge\ d\in p[C]$, there is a unique $c\in C\setminus g[A]$ such that $d=p(c)$. We define

$$l_D(d):=l_C(c),\quad \bar p_d:=i_{l_C(c)}$$

(c) Analogously, if $d\in q[B]\ \wedge\ d\notin p[C]$, we use the unique b such that $d=q(b)$ and

$$l_D(d):=l_B(b),\quad \bar q_d:=i_{l_B(b)}$$

This definition turns $q\cdot f=p\cdot q$ into a pushout diagram in the category of **L**-graphs. (i_x denotes identity on x up to isomorphisms.) We have $\hat q\cdot\hat f=\hat p\cdot\hat q$, and for all $\hat p',\hat q'$ with

[3] $q\cdot f=p\cdot q$ is a pushout if and only if for all p',q' with $q'\cdot f=p'\cdot q$ there exists a unique h such that $q'=h\cdot q$ and $p'=h\cdot p$.

$\hat{q}' \cdot \hat{f} = \hat{p}' \cdot \hat{g}$, we can construct a unique $\hat{s}$ satisfying $\hat{q}' = \hat{s} \cdot \hat{q}$ and $\hat{p}' = \hat{s} \cdot \hat{p}$. The underlying graph morphism s is uniquely defined in the category of graphs. By construction of l_D, there is only one way to make s a morphism in **L-Graph**. If we restrict consideration to injective graph morphisms, $\hat{q} \cdot \hat{f} = \hat{p} \cdot \hat{g}$ is a pushout in **L-Graph** if and only if $q \cdot f = p \cdot g$ is a pushout in **Graph** and labelling is defined as above. □

Of course, the dual constructions also hold.

3. Examples

We start with comparing our approach to playing the token game in a Petri net. Such a net consists of two disjoint node sets: the places and the transitions. Edges connect input places to transitions and transitions to output places. Whereas this underlying bipartite graph defines the static structure of the system, its dynamic behaviour is controlled by labelling the places. A transition is called enabled if its input places are labelled appropriately. Then, the transition can fire; this step changes the labelling of both the input places and the output places. We want to construct a graph grammar the productions of which correspond to firing the transitions of the net. More precisely we have Claims 3.1 and 3.2.

Claim 3.1. *A transition in the net is activated if and only if the corresponding production is applicable to the net.*

Claim 3.2. *Firing a transition in the net corresponds to a derivation step in the grammar and vice versa.*

As long as we are only interested in simulating the token game, the structure of the net is not changed and we can concentrate upon choosing an appropriate labelling category.

3.1. Places with at most one token

Let us first consider a rather simple example. We assume the places of the Petri net to be marked or not, i.e., at most one token can be put on a place. A transition is allowed to "fire" if all predecessors are marked (m) and all successors are not ($\bar{m}$). A characteristic situation is given in Fig. 1: the left-hand side and the right-hand side of this figure describe the same piece of a large net before the transition fires and after it has done.

In Fig. 2, we see this step in the graph-grammar framework. The pieces we have shown in Fig. 1 become the left-hand side $'B$ and the right-hand side B' of a graph production. The gluing graph K is a discrete graph consisting of all places involved in this step. The morphisms $'p$ and p' are given by numbering the places in a suitable way:

$$'p(i) = {'i}, \qquad p'(i) = i'.$$

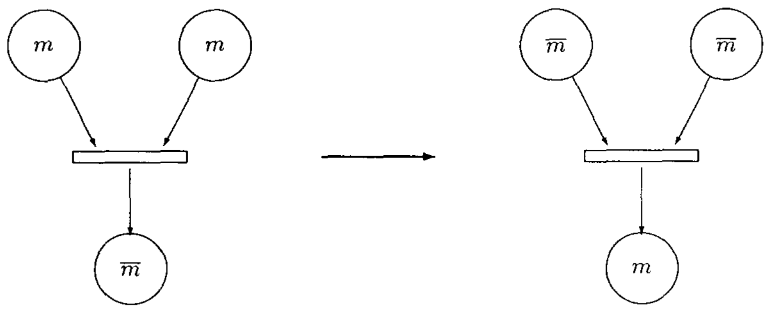

Fig. 1. A simple Petri net transition.

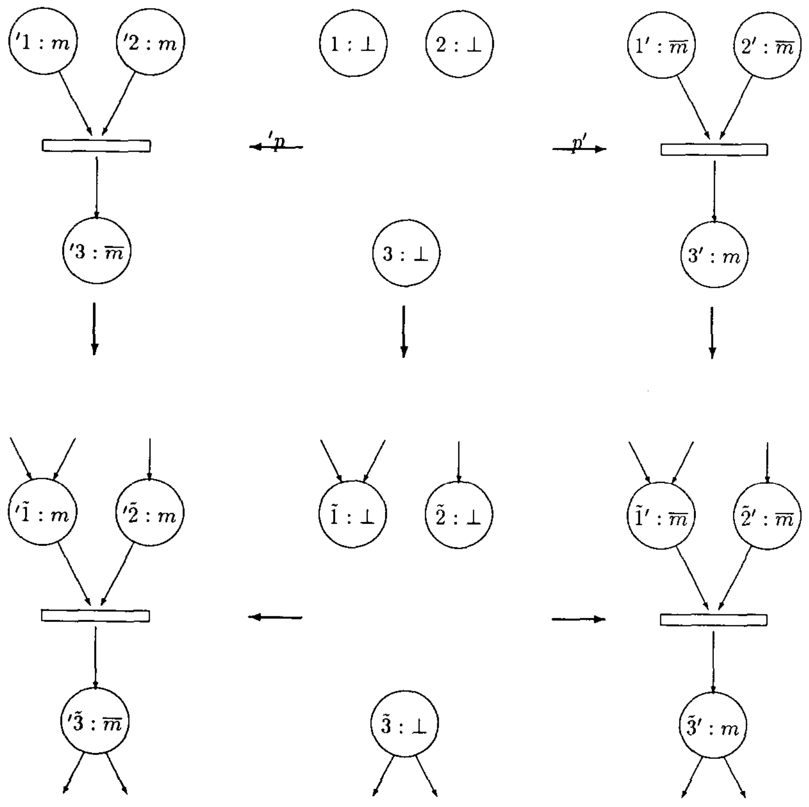

Fig. 2. Graph grammar interpretation of the simple transition.

In the sequel, we shall omit these numbers in order to avoid overloading the figures; the morphisms are then given by the relative positions of the places.[4]

[4] For the same reason, we do not consider the labelling of the transition and of the edges. Putting different labels on the transitions yields an unambiguous correspondence between productions and transitions.

Next, we have to discuss how to label the gluing graph K appropriately. We cannot use label-preserving morphisms because corresponding places are labelled differently. We choose a very simple category **S** to label the places: The objects are $m, \bar{m}$ and an additional bottom element $\perp$; the nontrivial morphisms are $\perp \to m$ and $\perp \to \bar{m}$. These morphisms allow us to label the places of the gluing graph by the bottom element, which is compatible with both m and $\bar{m}$. Thus, $'p$ and p' are **S**-graph morphisms.[5]

Let us now consider the applicability of such a production p to a Petri net $'G$. As we have mentioned in the discussion following Definition 2.2, we have to look for an **S**-graph morphism mapping the left-hand side $'B$ into $'G$ and then to find a morphism $K \to D$ making the left-hand part of the derivability diagram a pushout. Both steps place some restrictions on the structure of $'G$, more precisely on the places that are adjacent to the transition considered:

- All places mentioned in the production must also be present in $'G$, i.e., the morphism must not identify some places. We can satisfy this condition by restricting the category to morphisms the underlying graph morphisms of which are injective.
- In $'G$, no additional places may be adjacent to this transition. This is ensured by the fact that there is no node in the gluing graph the image of which is the transition; the gluing condition we have already mentioned [7] says that such a node cannot be connected to edges of $'G$ that are not an image of an edge in $'B$.

The last point we have to mention is how to label the gluing nodes in the context graph D. If we consider only the left-hand part of the derivability diagram, we can use either the bottom element or the label of the corresponding place in $'B$. But in the latter case, there does not exist a pushout object G' on the right-hand side! Thus, we have to choose the "minimal" context object.

3.2. Places with more than one token

A generalization of the simple Petri net leads to nets the places of which can contain more than one token. We leave out the special case of undistinguishable tokens, which usually are depicted by points, and immediately pass over to distinguishable ones. In this case, the places as well as the edges of the net are labelled with multisets, i.e., some elements may occur more than once. A transition is enabled if the label of each predecessor place includes the label of the edge connecting this predecessor to the transition. Firing removes the elements of the edge label from the place and adds the labels of the outgoing edges to those of the successor places.

[5] Please, note that there is no **S**-graph morphism $'B \to B'$ although the underlying unlabelled graphs are isomorphic. Furthermore, it is to be mentioned that by construction $'p$ and p' certainly are monomorphisms, but fail to be coretractions. This means that categorical proofs must not use existence of a left inverse.

Since the idea of grammatical derivation step is to remove the left-hand side of a production from the given graph, we have to reorganize labelling: We do not attach the tokens that are to be removed or added to the edges. On the left-hand side of the production, the predecessor places contain the tokens that are to be removed; analogously, the successor places on the right-hand side contain the tokens that are added in applying the production. The upper part of Fig. 3 shows a production removing a and b from one place and b from another place, while an a is added to the third place.[6]

We can straightforwardly embed this type of Petri nets into our graph-grammar framework by using a labelling category **M** the objects of which are multisets. In choosing the morphisms of this category, we could try to generalize the concept of the structured alphabet: A unique morphism $A \rightarrow B$ exists if and only if $A \subseteq B$, where $\subseteq$ denotes the multiset inclusion. But this approach does not work, because the pushout construction corresponds to labelling the places in the pushout object with

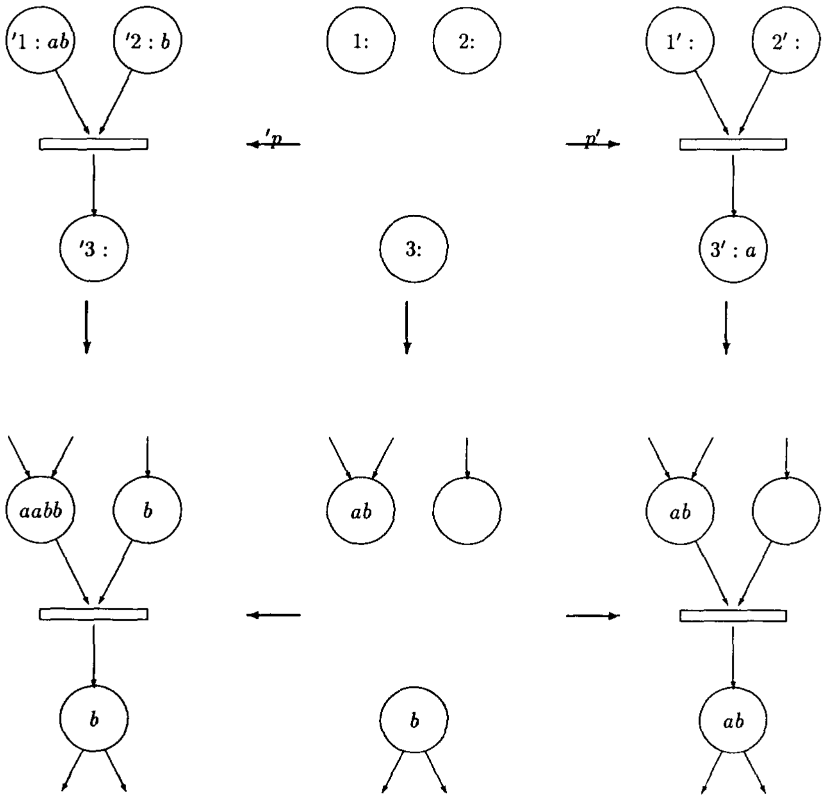

Fig. 3. A Petri net with multiple tokens.

[6] As in the previous example, primed numbers are used to indicate the morphisms, e.g., in the left-hand side of the production, '1 denotes the place that is mapped onto the first place mentioned above.

the usual set union, whereas we need the disjoint union. Appropriate morphisms are the mappings of multisets that preserve the "type" of the tokens, i.e., an a can only be mapped to an a, etc. As in the previous case, we have to restrict discussion to monomorphisms in order to establish the equivalence between derivation steps and firing transitions. In the category **M-Graph**, the monomorphisms are given by injective graph morphisms, together with injective mappings of the token sets.

The category of multisets is also able to handle nets with places of limited capacity. The idea is based on viewing free positions as special tokens moving backwards through transitions. Free positions are not computed from occupying tokens: generalizing our first model based on the category of S-graphs, the correspondence between occurrences and free positions must correctly be set up in the rules. Special symbols, marked by a bar, are used to indicate free positions. If the capacity is separately defined for each symbol x, we mark a free position by a corresponding $\bar{x}$. Otherwise, only one symbol, e.g., $\bar{m}$, is additionally required. Of course, it is possible to use places with limited capacity and places without this restriction in the same grammar.

3.3. *Places labelled with graphs*

Now, we leave Petri nets: The generalized definition of **L**-graphs and **L**-graph morphisms allows us to consider nets the places of which are labelled with graphs. For reason of simplification, however, we restrict the labelling category to the usual category of labelled graphs, i.e., the graph morphisms of the labelling category preserve labels.[7] We call this category **G-Graph**.

The example of Fig. 4 is a detail of a system of processes that share a common memory containing structured data. For a moment, let us think of a system of producers and consumers. If the elements put into the buffer by the producer can be removed in an arbitrary order, the multiple-token approach is adequate. This does no

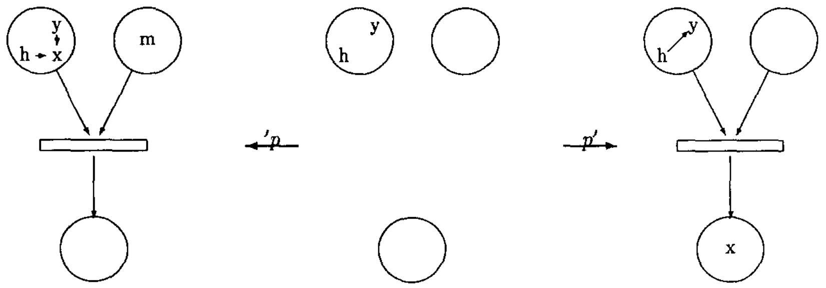

Fig. 4. A metaproduction in a net labelled with graphs.

[7] The more general case would also be of interest. Then, we get a close relationship to Pratt's [12] hierarchical graphs.

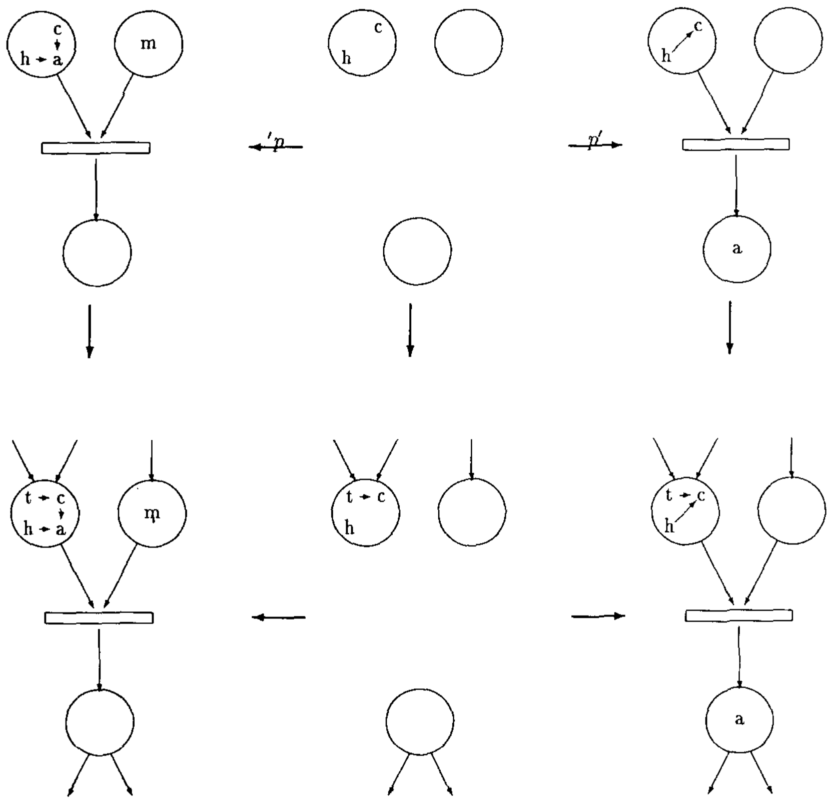

Fig. 5. A transition in a net labelled with graphs.

longer hold true if we require the buffer to observe a special strategy, e.g., to be a FIFO-queue. In our example, the leftmost predecessor place of the transition is the buffer place. It contains a graph representing the queue; a special node, labelled with *h*, points to the first element of the queue and *t* to the last element. The second predecessor place is part of the consumer process. When it is labelled with *m*, this process is ready to consume the next element of the queue. After this step, the former head *x* of the queue is on the successor place in the consumer process, whereas *h* points to the next element of the queue in the buffer.

The production of Fig. 4 is a metaproduction in the sense of van Wijngaarden et al. [16]; it defines an infinite set of productions the elements of which we can get by substituting concrete labels for the variables *x* and *y*. In Fig. 5, we have substituted *a* for *x* and *c* for *y*, and we apply the resulting production to a concrete net case. As in our previous examples, the context is indicated by some edges only. Before the derivation step is applied, the buffer contains *a* and *c* with *a* being the head; afterwards, *a* has been deleted and *h* points to *c*. The unchanged part of the queue belongs to the context graph.

Remark. At the first glance, this production seems to be restricted to buffers that contain at least two elements. An appropriate convention, however, allows us to avoid this restriction: we suppose the buffer contents to be terminated by a special element π. This element may be substituted for y, but not for x.

4. Parallel independence of productions

4.1. The parallel independence theorem

Studying the behaviour of systems of processes, we are interested in some properties of derivation sequences that reflect asynchronous behaviour. Typical problems are characterizing parallel and sequential independence, amalgamating parallel productions or replacing a derivation sequence by one production. We demonstrate the applicability of our approach by considering parallel independence of derivation steps: If two productions are applicable to a given graph G, the result may depend on the order of application, or even the second production is no longer applicable after the first has been applied. Of course there is no problem if the left-hand sides of both productions are mapped into disjoint parts of G. But Ehrig [2] observed that this condition is too strong.

Definition 4.1. Two derivation steps (Diagram 7) are called parallel-independent if and only if there exist morphisms $\alpha_1 : {}'B_1 \to D_2$ and $\alpha_2 : {}'B_2 \to D_1$ with $g_1 = {}'\tilde{p}_2 \cdot \alpha_1 \wedge g_2 = {}'\tilde{p}_1 \cdot \alpha_2$.

This characterization can be interpreted as follows: If we apply p_1 first, we are able to apply p_2 subsequently by embedding its left-hand side into H_1 (instead of G) by the morphism $\tilde{p}'_1 \cdot \alpha_2$. The result does not depend on the order of applying the productions if they behave well. Ehrig and Kreowski [5] studied the special case of sets. Their proofs take full advantage of set-theoretic concepts. Especially, the morphisms under consideration must be injective mappings, and the proofs are based on analyzing the elements one by one. The proof cannot be generalized to arbitrary categories because it uses the so-called triple-pushout condition that is not satisfied in each case. In

$$\begin{tikzcd}
{}'B_i \arrow[d, "g_i"'] & K_i \arrow[l, "{}'p_i"'] \arrow[r, "p'_i"] \arrow[d] & B'_i \arrow[d] \\
G & D_i \arrow[l, "{}'\tilde{p}_i"] \arrow[r, "\tilde{p}'_i"'] & H_i
\end{tikzcd}
\qquad i = 1,2$$

Diagram 7.

$$\begin{tikzcd}
A \arrow[r] \arrow[d] \arrow[rd, phantom, "(1)"] & B \arrow[d] \\
C \arrow[r] & D
\end{tikzcd}$$

Diagram 8.

$$\begin{tikzcd}
& K_2 \arrow[r] \arrow[d] \arrow[rd, phantom, "(1)"] & B_2 \arrow[d] \\
K_1 \arrow[r] \arrow[d] \arrow[rd, phantom, "(2)"] & D_0 \arrow[r] \arrow[d] \arrow[rd, phantom, "(3)"] & D_1 \arrow[d] \\
B_1 \arrow[r] & D_2 \arrow[r] & G_1
\end{tikzcd}$$

Diagram 9.

a recent paper, Ehrig et al. [4] restricted discussion to categories that allow to prove the parallel independence theorem (PIT).

Definition 4.2. We call a category **C** a PIT-category with respect to a class M of distinguished morphisms if the following conditions are satisfied:

(a) If morphisms $B \leftarrow A \rightarrow C$ are in M, there exists a pushout diagram (1) as in Diagram 8 and $B \rightarrow D \leftarrow C$ are also in M.

(b) If morphisms $B \rightarrow D \leftarrow C$ are morphisms in M, there exists a pullback diagram (1) and $B \leftarrow A \rightarrow C$ are also in M.

(c) If in Diagram 9 all morphisms are in M, the diagrams $(1+3)$ and $(2+3)$ are pushouts, and diagram (3) is a pullback, then (1), (2) and (3) are pushouts.

In such a category, it is possible to prove the parallel independence theorem:

Theorem 4.3. *If in a PIT-category two derivation steps*

$$G_1 \overset{p_1}{\Rightarrow} H_1 \quad \wedge \quad G_1 \overset{p_2}{\Rightarrow} H_2$$

are parallel-independent and 'p_i, p_i', g_i $(i = 1, 2)$ *are in* M, *then there exists a* G_2 *such that*

$$H_1 \overset{p_2}{\Rightarrow} G_2 \quad \wedge \quad H_2 \overset{p_1}{\Rightarrow} G_2.$$

The proof, that can be carried out without leaving the categorical framework, is analogous to that in the category of sets [2].

4.2. Application of the theorem

If we want to apply the theorem to the examples of Section 3, we have to show that these categories satisfy the PIT condition. In the category of unlabelled graphs, a commutative diagram is a pushout if and only if both the node and the edge component are pushouts in the category of sets, and the same holds true for pullbacks.

This means that **Graph** is a PIT-category with respect to injections as Ehrig and Kreowski [5] have proved. Therefore, we can restrict our discussion to consider the influence of labelling.

Theorem 4.4. G-Graph *is a PIT-category with respect to the following class* M *of morphisms*: $\hat{f}$ *is in* M *if and only if the underlying graph morphism* f *is injective and all* $\bar{f}_x$ *are injective graph morphisms.*

Proof. Conditions a and b of Definition 4.2 are satisfied because of the construction in Theorem 2.11 and the dual one: In case a of Theorem 2.11, $\bar{p}$ and $\bar{q}$ inherit injectivity of $\bar{f}$ and $\bar{g}$, respectively; the other cases are trivial. Now, we consider Diagram 9 and assume x to be a node (or an edge) of G_1. We have to study three cases:
- x has pre-images in B_1 as well as in B_2.
- x has a pre-image in B_1, but not in B_2 or vice versa.
- x has neither a pre-image in B_1 nor in B_2.

First, let x have a pre-image in B_1 as well as in B_2. The pushout properties of $(2+3)$ and $(1+3)$ ensure existence of pre-images in K_1 and K_2, and, therefore, in D_0, D_1 and D_2. As we have already mentioned, the triple-pushout condition holds in the category of unlabelled graphs. Thus, we can separately apply the construction in Theorem 2.11 to each square, and we get the same diagram with the graphs being replaced by the labels of x and of its pre-images. This completes the proof of this case because we have assumed to be now in the category of graphs with label-preserving morphisms. Next, let x have a pre-image in B_1, but not in B_2. There are two subcases: x has a pre-image in D_1 or it has not. In the second subcase, the labels of x and its pre-images are isomorphic; this follows from the construction in Theorem 2.11 and the dual construction because $(2+3)$ is a pushout and (3) is a pullback. In the first subcase, the pushout property of $(2+3)$ ensures that there is a pre-image in K_1 and, therefore, in D_0. At the level of labels, i.e., the graphs in the diagram being replaced by x and its pre-images, we have the situation of the factorization lemma for pushouts that was proved by Ehrig and Kreowski [5]. Thus, diagrams (2) and (3) are pushouts. In the last case, x must have pre-images in both D_1 and D_2; therefore, in D_0, too. Since $D_2 \rightarrow G_1$ is part of the pushout diagram $(1+3)$, the labels of x and its pre-image in D_2 are isomorphic according to Theorem 2.11. The same holds true for $D_1 \rightarrow G_1$, and, trivially, the pullback is also a pushout. This completes the proof. □

In this category, the parallel independence theorem says that we can apply productions concurrently even if they share places. But we have to ensure that the contents of these places can be divided among the different productions in a suitable way. More precisely, we must be able to embed the left-hand side of one production into the context graph of the second; this context also comprises the unaffected part of the data structure that is contained in the common places.

Corollary 4.5. M-Graph *is a PIT-category with respect to label-preserving multiset injections.*

Proof. M-Graph is a full subcategory of **G-Graph**, for we get the objects of **M-Graph** by requiring the labels to be discrete graphs. Therefore, we can immediately transfer the results we have just got in the general case. □

We illustrate this case by considering the example of Fig. 3. If the predecessor place that is originally labelled with $\{a, a, b, b\}$ is also the predecessor place of another transition that consumes a or b or both elements, firing of this second transition is independent of the depicted step. The result is not affected by the order of executing these derivation steps. We can say that they may be applied in parallel.

The situation in **S-Graph** is a little bit more complicated because S certainly has all pullbacks, but not all pushouts. The pushouts of interest, however, satisfy the triple-pushout condition. We can show Theorem 4.3 if we restrict the assumption c of Definition 4.2:

... if the diagrams $(1+3)$ and $(2+3)$ are pushouts that may occur in derivation steps ...

No further difficulties arise in choosing M: since there is atmost one morphism between two labels, we can use injective graph morphisms together with any label morphism.

Because of the simple structure of the set of label morphisms, there is only one nontrivial case in proving c of Definition 4.2. Let x be a node that has pre-images in B_1 as well as in B_2, but with different labels, e.g., the pre-image in B_1 is labelled with m and that in B_2 with $\perp$. Now, we use the fact that the derivation steps assume all gluing nodes and their images in the context graph to be labelled with the bottom element. Therefore, the pre-images of x in K_1, K_2, B_2, D_0 and D_1 are labelled with $\perp$ and those in B_1 and D_2 with m. Then, Theorem 2.11 yields all squares to be pushouts. □

In the category **S-Graph**, the parallel independence theorem leads to the well-known fact that two enabled transitions can fire in parallel only if they have no predecessor or successor place in common. If they have, the morphisms α_i cannot exist since in the context graphs D_i all places adjacent to the firing transitions must be labelled with $\perp$ and there is no morphism $m \to \perp$ or $\bar{m} \to \perp$.

5. Conclusion

The research we have reported on is part of a larger project concerned with different multiprocessor environments. We need a formal basis to discuss mapping logical process structures into different hardware structures, and we are thoroughly convinced that the graph-grammar approach is better suited than others. In a previous paper [15], we considered another aspect of systems of asynchronous processes. We translated the statements of the program into graph productions and defined the "traces" the system can run through by the set of derivable graphs. In that paper, we did not lay emphasis on synchronization mechanisms. Of course, both concepts can be combined. Another special aspect we are also interested in is application of our results to database theory.

Acknowledgment

The author thanks H. Ehrig, B. Hindel, M. Löwe, G. Schied (who especially made valuable contributions to **G-Graph**) and the referees for carefully reading previous versions of this paper and for their helpful suggestions.

References

[1] P.M. van den Broek, Comparison of two graph-rewrite systems, *Theoret. Comput. Sci.* **61** (1988) 67–81.

[2] H. Ehrig, Introduction to the algebraic theory of graph grammars (a survey), in: *Proc. 1st Graph Grammar Workshop*, Lecture Notes in Computer Science, Vol. 73 (Springer, Berlin, 1979) 1–69.

[3] H. Ehrig, P. Boehm, U. Hummert and M. Löwe, Distributed parallelism of graph transformations, in: *Proc. 13th Internat. Workshop on Graph-theoretic Concepts in Computer Science*, Lecture Notes in Computer Science, Vol. 314 (Springer, Berlin, 1988) 1–19.

[4] H. Ehrig, A. Habel, H.J. Kreowski and F. Parisi-Presicce, Parallelism and concurrency in high level replacement systems, *Math. Structures Comput. Sci.*, to appear.

[5] H. Ehrig and H.J. Kreowski, Pushout properties – an analysis of gluing constructions, *Math. Nachr.* **91** (1979) 135–149.

[6] H. Ehrig, H.J. Kreowski, A. Maggiolo-Schettini, B.K. Rosen and J. Winkowski, Transformations of structures – an algebraic approach, *Math. Systems Theory* **14** (1981) 305–334.

[7] H. Ehrig, M. Pfender and H.J. Schneider, Graph grammars – an algebraic approach, in: *Proc. 14th Ann. Conf. on Switching and Automata Theory* (1973) 167–180.

[8] H.J. Kreowski, A comparison between Petri-nets and graph grammars, Lecture Notes in Computer Science, Vol. 100 (Springer, Berlin, 1981) 306–317.

[9] H.J. Kreowski, Is parallelism already concurrency? – part 1: derivations in graph grammars, in: *Proc. 3rd Internat. Workshop on Graph Grammars and Their Applications to Computer Science*, Lecture Notes in Computer Science, Vol. 291 (Springer, Berlin, 1987) 343–360.

[10] H.J. Kreowski and A. Wilharm, Is parallelisms already concurrency? – part 2: non-sequential processes in graph grammars, in: *Proc. 3rd Internat. Workshop on Graph Grammars and Their Applications to Computer Science*, Lecture Notes in Computer Science, Vol. 291 (Springer, Berlin, 1987) 361–377.

[11] F. Parisi-Presicce, H. Ehrig and U. Montanari, Graph rewriting with unification and composition, Lecture Notes in Computer Science, Vol. 291 (Springer, Berlin, 1987) 496–514.

[12] T. Pratt, Pair grammars, graph languages, and string-to-graph translations, *J. Comput. System Sci.* **5** (1971) 560–595.

[13] J.C. Raoult, On graph rewritings, *Theoret. Comput. Sci.* **32** (1984) 1–24.

[14] W. Reisig, A graph grammar representation of nonsequential processes, Lecture Notes in Computer Science, Vol. 100 (Springer, Berlin, 1981) 318–325.

[15] H.J. Schneider, Describing distributed systems by categorical graph grammars. in: *Proc. 15th Internat. Workshop on Graph-theoretic Concepts in Computer Science*, Lecture Notes in Computer Science, Vol. 411 (Springer, Berlin, 1990) 121–135.

[16] A. van Wijngaarden et al. Report on the algorithmic language ALGOL 68, *Numer. Math.* **14** (1969) 79–218.

[17] J.C. Wileden, Relationships between graph grammars and the design and analysis of concurrent software, Lecture Notes in Computer Science, Vol. 73 (Springer, Berlin, 1979) 456–463.