Math. Struct. in Comp. Science (2002), *vol.* 12, *pp.* 149–175.
DOI: 10.1017/S0960129501003486 Printed in the United Kingdom

Hypergraph construction and its application to the static analysis of concurrent systems†

BARBARA KÖNIG

Fakultät für Informatik, Technische Universität München

Received 7 July 2000; revised 6 December 2000

We define a construction operation on hypergraphs based on a colimit and show that its expressiveness is equal to the graph expressions of Bauderon and Courcelle. We also demonstrate that by closing a set of rewrite rules under graph construction we obtain a notion of rewriting equivalent to the double-pushout approach of Ehrig. The usefulness of our approach for the compositional modelling of concurrent systems is then demonstrated by giving a semantics of process graphs (corresponding to a process calculus with mobility) and of Petri nets. We introduce on the basis if a hypergraph construction, a method for the static analysis of process graphs, related to type systems.

1. Introduction

Graph rewriting is one adequate approach for modelling the dynamics of concurrent systems: multi-dimensional structures describing interconnected computers or other components can be described naturally by graphs. While in 'string-based' notations connections between components are normally described implicitly (by having common channel names), they can be described explicitly (by connecting them with an edge) in a graphical representation.

When trying to model semantic frameworks for concurrency (such as process algebras) with graph rewriting, we encounter problems since it is hard to represent the compositionality and modularity inherent in these formalisms in the world of graphs. The double-pushout approach (Ehrig 1979), which is a standard method of defining graph rewriting, does not define the notion of a rewriting step in an inductive way. Process algebras, however, rely on compositionality in the definition of their syntax and their semantics, and in almost all proofs. Reduction semantics (or labelled transition semantics) is always defined inductively on the structure of processes, and the same is true for types of processes (Pierce and Sangiorgi 1993; Turner 1995). Furthermore, the notion of compositionality is important for the definition of behavioural equivalences (such as bisimulation), since these behavioural equivalences are expected to be preserved by composition, that is, they should be congruences.

Compositionality is easy to achieve in a 'string-based' syntax: systems are constructed

† Research supported by SFB 342 (subproject A3) of the DFG.

out of smaller systems by concatenating their descriptions, and connections are established by having common channel names. We propose an analogue to concatenation in the world of hypergraphs: hypergraphs have so-called 'external nodes', with their interface to the outside, and in order to attach two or more hypergraphs, information is needed on how these external nodes should be merged. In our case, this information or 'construction plan' is given in the form of graph morphisms, which are part of a colimit, and the result of the construction is obtained by completing the colimit.

The aim of this paper is to introduce this construction operator, to investigate its properties and then to apply the theory to concurrent systems such as process calculi and Petri nets. Although we model concurrent systems, no truly concurrent rewriting steps are considered: we concentrate on interleaving semantics and it is one of our central aims to provide methods of static analysis, allowing us to check invariant properties of the structure of concurrent systems. This idea will be demonstrated by introducing type systems for process graphs.

Existing approaches to the inductive definition of graphs are given in Bauderon and Courcelle (1987), Gadducci and Heckel (1997), and Engelfriet and Vereijken (1997), and are mainly based on a fixed set of operators on graphs, which can be used to construct graph terms, and an equational theory defining isomorphism on graph terms. In Bauderon and Courcelle (1987) the operators are disjoint sum, fusion and redefinition of external nodes and it will later (see Section 3.1) be shown how these operators can be obtained with the graph construction operation presented in this paper. In the latter two papers graphs have variable nodes and root nodes, and there is, among others, a concatenation operation merging the root nodes of the first graph with the variable nodes of the second graph. This specific operation is less general than ours since we can, for example, also express reordering of external nodes.

Our construction operation is closer to the double-pushout approach since it involves colimits, and we can therefore exploit their universal properties. Under certain conditions we can automatically infer the existence of graph morphisms (such as the embeddings of the subgraphs into the constructed graph) without the need to construct them explicitly. Furthermore, we can rely on the fact that the composition of colimits again yields a colimit, which can be a great help in many proofs. Since there is only one operation of hypergraph construction, we only need to consider one case in structural induction on graphs.

The notion of inductive definition of graphs naturally extends to the notion of a rewriting step: if a graph can be disassembled into several parts, one of which is a left-hand side, we simply replace the left-hand side by the corresponding right-hand side and reassemble the graph as before. The expressive power of graph rewriting in our approach is the same as for graph expressions (Bauderon and Courcelle 1987) and for the double-pushout approach.

Additionally, we integrate into our setting a method for the annotation of hypergraphs with an algebraic structure (in our case, a commutative monoid). This is motivated by the observation that pure graph structure is sometimes not sufficient to adequately model concurrent systems. For example, in the case of Petri nets (where hyperedges represent transitions and nodes represent places), it is convenient to annotate nodes by

natural numbers, indicating the number of tokens present at each node. It is, furthermore, necessary to define the notion of graph construction for annotated hypergraphs (if, in the Petri net example, two nodes are merged, their annotations should be added) and, also, the notion of graph morphism must be modified to take annotations into account. Another application for annotations is the new notion of generic type systems introduced in Section 6.2, where types can be regarded as abstractions of the future behaviour of a process. In this case, annotations can be considered as constraints on the behaviour. Formally, potential annotations for a graph are given by a functor from the category of hypergraphs (with graph morphisms) into the category of commutative monoids (with monoid morphisms).

In the rest of this paper, we will first define some basic notions such as hypergraphs and hypergraph morphisms, then a graph construction operator will be defined. Afterwards, this construction operator will be compared to existing approaches, more specifically it will be shown that the expressiveness of our construction operator is equivalent to the operators introduced in Bauderon and Courcelle (1987) and that the closure of rewriting rules under graph construction gives a notion of graph transformation equivalent to the double-pushout approach (Ehrig 1979). We will investigate the properties of the construction operator and then present a first application by modelling process graphs, which are strongly related to the polyadic π-calculus (Milner 1993). In π-calculus there is a concept of type systems, which can be employed to avoid runtime errors produced by mismatching arities. An equivalent type check can be conducted for process graphs, the correctness proof of which exploits many properties of hypergraph construction established in the previous sections. We then introduce our method of annotating hypergraphs and use it to model Petri nets and to extend the type check in order to obtain so-called generic type systems, which can be instantiated to check specific properties such as input/output-behaviour or secrecy violations.

2. Hypergraphs and hypergraph construction

We first define some basic notions, namely hypergraph, hypergraph morphism, and isomorphism (see also Habel (1992)). In the following, the term 'graph' is also intended to denote a hypergraph. Hypergraphs are a generalisation of directed graphs where an arbitrarily long sequence of nodes is assigned to every edge. Intuitively, we construct hypergraphs by drawing edges (with nodes) and then merging the nodes, rather than by drawing nodes and then connecting them by edges. This intuition will also guide our choice of a hypergraph construction operator.

Definition 1. (Hypergraph, hypergraph morphism, isomorphism) Let L be a fixed set of labels. A *hypergraph* H is a tuple $H = (V_H, E_H, s_H, l_H, \chi_H)$ where V_H is a set of *nodes*, E_H is a set of *edges*, $s_H : E_H \to V_H^*$ maps each edge to a string of *source nodes*, $l_H : E_H \to L$ assigns a label to each edge, and $\chi_H \in V_H^*$ is a string of *external nodes*.

Let H, H' be two hypergraphs. A *hypergraph morphism* $\phi : H \to H'$ consists of two

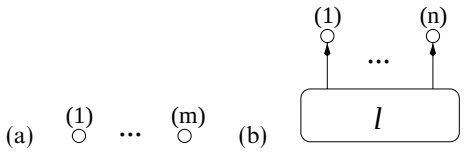

Fig. 1. A discrete hypergraph and a hypergraph with a single edge.

mappings $\phi_V : V_H \to V_{H'}$, $\phi_E : E_H \to E_{H'}$ satisfying† $\phi_V(s_H(e)) = s_{H'}(\phi_E(e))$ and $l_H(e) = l_{H'}(\phi_E(e))$ for all $e \in E_H$. If, furthermore, $\phi_V(\chi_H) = \chi_{H'}$, we call ϕ a *strong* hypergraph morphism and denote it by $\phi : H \twoheadrightarrow H'$. The hypergraphs H and H' are called *isomorphic* ($H \cong H'$) if there exists a bijective strong morphism from one hypergraph into the other.

A hypergraph morphisms $\eta : H \to H'$ is called an *embedding* if it is injective on E_H and on all non-external nodes of H and if, furthermore, if there is an $e' \in E_{H'}$ such that $s_{H'}(e')$ contains a node that is the image under η of a non-external node of H, then e' is in the range of η. It must also be the case that if $\eta(v)$ is external for a node $v \in V_H$, then v must also be external.

As usual, we omit the index in ϕ_V and ϕ_E if it can be derived from the context. The arity of a hypergraph H is defined as $ar(H) = |\chi_H|$ (where $|\tilde{s}|$ denotes the length of a string $\tilde{s}$) while the arity of an edge e of H is $ar(e) = |s_H(e)|$. We can regard hypergraphs and hypergraph morphisms as the objects and arrows, respectively, of a category **HGraph**, which has a subcategory **SHGraph** containing only the strong hypergraph morphisms. If we say that two hypergraphs are isomorphic, we usually refer to isomorphism in **SHGraph** (see Definition 1).

Notation: We call a hypergraph *discrete*, if its edge set is empty. The symbol **m** denotes a discrete graph of arity $m \in \mathbb{N}$ with m nodes where every node is external (see Figure 1(a) where external nodes are labelled (1), (2), ... in their respective order). We will also call a graph morphism discrete if both its source and target graph are discrete.

Furthermore, $H = [l]_n$ is the hypergraph with exactly one edge e labelled l where $s_H(e) = \chi_H$, $|\chi_H| = n$, χ_H contains no duplicates and $V_H = Set(\chi_H)$ where $Set(\tilde{s})$ is the set of all elements of a string $\tilde{s}$ (see Figure 1(b) where the nodes of an edge are ordered from left to right).

We now present the 'concatenation operation' discussed in the introduction. The construction plan, telling us how this concatenation is supposed to happen, is represented by hypergraph morphisms ζ_i mapping discrete graphs of the form depicted in Figure 1(a) to arbitrary discrete graphs. This construction plan is independent of the specific graphs to be concatenated (apart from their arity) and is applied to hypergraphs $H_1, \ldots, H_n$ by taking their disjoint sum and fusing their external nodes as specified by the morphisms ζ_i.

The following definitions use concepts from category theory, namely categories and colimits. For an introduction to these concepts see Crole (1993) and Mac Lane (1971).

† The application of morphisms to sequences of nodes is conducted pointwise.

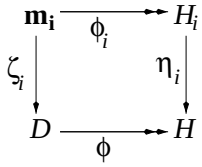

Fig. 2. Hypergraph construction as a colimit.

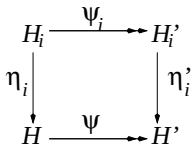

Fig. 3.

Definition 2. (Hypergraph construction) Let $H_1, \ldots, H_n$ be hypergraphs and let $\zeta_i : \mathbf{m_i} \to D$, $i \in [n]$ (where $[n]$ stands for the set $\{1, \ldots, n\}$) be morphisms where $ar(H_i) = m_i \in \mathbb{N}$ and D is a discrete graph. There is always a unique strong morphism $\phi_i : \mathbf{m_i} \twoheadrightarrow H_i$ for every $i \in [n]$.

Now we let H (with morphisms $\phi : D \twoheadrightarrow H$, $\eta_i : H_i \to H$) be the colimit of $\zeta_1, \ldots, \zeta_n, \phi_1, \ldots, \phi_n$ in **HGraph** such that ϕ is a strong morphism (see Figure 2). We define: $\bigotimes_{i=1}^{n}(H_i, \zeta_i) = H$. (Alternatively, we write $(H_1, \zeta_1) \otimes \ldots \otimes (H_n, \zeta_n)$, or $\otimes(H_1, \zeta_1)$ if $n = 1$, instead of $\bigotimes_{i=1}^{n}(H_i, \zeta_i)$.)

The colimit defined above always exists (see Proposition 1), but is unique only up to isomorphism in **HGraph**. Therefore we demand above that the morphism ϕ from D into the colimit is a strong morphism, thereby determining the string of external nodes of the result and getting a graph that is unique up to isomorphism in **SHGraph**. We can also think of the diagram as a colimit where the horizontal arrows are from **SHGraph** and the vertical arrows are from **HGraph**.

Furthermore, the morphisms η_i generated by the colimit are always embeddings.

If we now partition the category of strong hypergraph morphisms into categories $\mathbf{SHGraph}_m$, each of which contains exactly the hypergraphs of a fixed arity $m \in \mathbb{N}$, we can regard $\bigotimes_{i=1}^{n}(\__i, \zeta_i)$ with $\zeta_i : \mathbf{m_i} \to D$ as a functor from the cartesian product $\mathbf{SHGraph}_{m_1} \times \ldots \times \mathbf{SHGraph}_{m_n}$ into the category $\mathbf{SHGraph}_{ar(D)}$. Each tuple $(\psi_1, \ldots, \psi_n)$ of strong morphisms with $\psi_i : H_i \twoheadrightarrow H_i'$ and $ar(H_i) = m_i$ yields a strong morphism

$$\psi = \bigotimes_{i=1}^{n}(\psi_i, \zeta_i) : \underbrace{\bigotimes_{i=1}^{n}(H_i, \zeta_i)}_{H} \twoheadrightarrow \underbrace{\bigotimes_{i=1}^{n}(H_i', \zeta_i)}_{H'}$$

where ψ is generated by the colimit in Figure 3 and the $\eta_i : H_i \to H$ and $\eta_i' : H_i' \to H$ are the embeddings generated by the graph construction.

Although the characterisation of hypergraph construction is more elegant in the categorical setting, we can also describe it without category theory in a constructive way. In this way we also show that the colimit actually exists.

Proposition 1. Let $H_1, \ldots, H_n$ be hypergraphs with $m_i = ar(H_i)$ and let $\zeta_i : \mathbf{m_i} \to D$, $i \in [n]$ be discrete morphisms such that the node and edge sets of $H_1, \ldots, H_n$ and D are pairwise disjoint. Furthermore, let $\approx$ be the smallest equivalence on the union of their nodes satisfying $\zeta_i(v) \approx \phi_i(v)$ if $i \in [n]$, $v \in V_{\mathbf{m_i}}$. The nodes of the constructed graph are the equivalence classes of $\approx$. Thus $\bigotimes_{i=1}^{n}(H_i, \zeta_i)$ is isomorphic to

$$H = ((V_D \cup \bigcup_{i=1}^{n} V_{H_i})/\approx, \bigcup_{i=1}^{n} E_{H_i}, s_H, l_H, \chi_H)$$

where $s_H(e) = [v_1]_\approx \ldots [v_k]_\approx$ if $e \in E_{H_i}$ and $s_{H_i}(e) = v_1 \ldots v_k$. Furthermore, $l_H(e) = l_{H_i}(e)$ if $e \in E_{H_i}$, so we define $\chi_H = [v_1]_\approx \ldots [v_k]_\approx$ if $\chi_D = v_1 \ldots v_k$.

Additionally, the morphisms $\phi : D \twoheadrightarrow H$ and $\eta_i : H_i \to H$ generated by the colimit can be constructed as follows: $\phi(v) = [v]_\approx$ if $v \in V_D$. Furthermore, $\eta_i(v) = [v]_\approx$ if $v \in V_{H_i}$ and $\eta_i(e) = e$ if $e \in E_{H_i}$.

In other words, we unite disjointly all graphs $D, H_1, \ldots, H_n$ and fuse two nodes whenever they are the image of one and the same node in one of the $\mathbf{m_i}$. The equivalence classes of the nodes of χ_D form the new sequence of external nodes.

Proof. In order to show that our constructive definition is indeed correct, we have to prove that whenever there are morphisms $\eta_i' : H_i \to H'$ and a strong morphism $\phi' : D \twoheadrightarrow H'$, there exists a unique morphism $\psi : H \to H'$ such that $\psi \circ \eta_i = \eta_i'$ and $\psi \circ \phi = \phi'$. The only way to get such a ψ is to define $\psi([e]_\approx) = \eta_i'(e)$ whenever $e \in E_{H_i}$, and $\psi([v]_\approx) = \eta_i'(v)$ whenever $v \in V_{H_i}$, and $\psi([v]_\approx) = \phi'(v)$ whenever $v \in V_D$. It can be shown straightforwardly that ψ is a well-defined graph morphism satisfying the necessary conditions. □

Example (1): We want to construct a graph H consisting of a hyperedge representing a message (labelled M) and two hyperedges representing processes (labelled P, Q). (In Section 5 we will show how the reception of a message by a process can be modelled with graph rewriting.)

The hypergraph H consists of two subgraphs H_1 (containing the process labelled P and a message) and H_2 (containing a process labelled Q) that are concatenated according to the 'construction plan' given by the discrete morphisms ζ_1, ζ_2 (see Figure 4). For example, the third external node of H_1 is fused with the third external node of D, which, in turn, is fused with the first external node of H_2. Isolated nodes in H are generated by nodes in D that are not in the range of either of the ζ_i.

Example (2): Another example is an operator that takes two hypergraphs of the same arity and attaches them at their external nodes, that is, the nodes are merged in their respective order. Let m be a fixed natural number and let $\zeta : \mathbf{m} \twoheadrightarrow \mathbf{m}$ be a strong morphism. For H_1, H_2 satisfying $ar(H_1) = ar(H_2) = m$, we define $H_1 \square H_2 = (H_1, \zeta) \otimes (H_2, \zeta)$. This operator

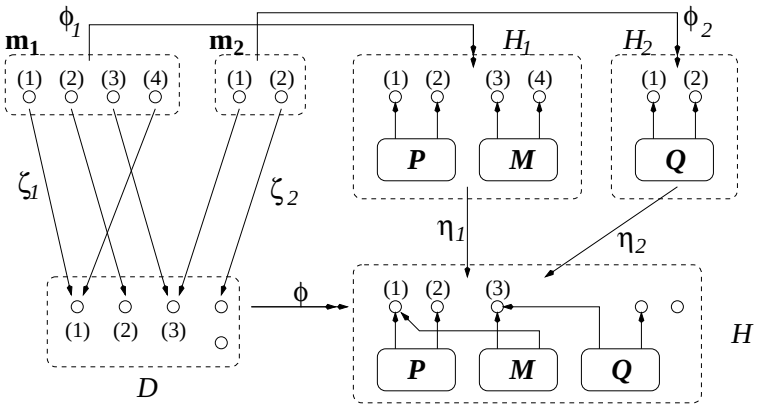

Fig. 4. Example of a graph construction operation.

is similar to the duplicator defined in Gadducci and Heckel (1997), and will be used later in this paper.

3. Comparison with graph expressions and the double-pushout approach

3.1. *Graph expressions*

Graph expressions were introduced in Bauderon and Courcelle (1987) as an algebraic structure for graph construction. They introduced three operators (explained below) and a complete set of equations relating hypergraphs if and only if they are isomorphic. We will now introduce the three operators and give their corresponding version in our framework in terms of the discrete morphisms ζ_i.

Disjoint sum: Given two hypergraphs H_1, H_2, we have $H_1 \oplus H_2$ is the hypergraph resulting from the disjoint union of the node and edge sets, and of the source and labelling functions of H_1 and H_2. Furthermore, χ_{H_1} and χ_{H_2} are concatenated to form the sequence of external nodes of $H_1 \oplus H_2$. We then have $H_1 \oplus H_2 \cong \bigotimes_{i=1}^{2}(H_i, \zeta_i)$ where ζ_1, ζ_2 are defined as in Figure 5(a) ($m = ar(H_1)$, $n = ar(H_2)$).

Redefinition of external nodes: Let $\alpha : [p] \to [m]$ and $m = ar(H)$. Then $\sigma_\alpha(H)$ is the hypergraph resulting from the redefinition of the external nodes of H according to α, that is, χ_H is replaced by[†] $\lfloor \chi_H \rfloor_{\alpha(1)} \dots \lfloor \chi_H \rfloor_{\alpha(p)}$. The rest of the hypergraph stays unchanged.

We exploit the fact that α can always be decomposed into $\alpha = \alpha_1 \circ \dots \circ \alpha_k$ where each α_i is either a permutation, or it hides the last external node, or it duplicates the last external node (see below). According to Bauderon and Courcelle (1987), $\sigma_\alpha(H) \cong \sigma_{\alpha_k}(\dots \sigma_{\alpha_1}(H) \dots)$. Thus we only have to consider the following three cases, in each of which we have $\sigma_\alpha(H) \cong \otimes(H, \zeta)$ with the appropriate discrete morphisms shown in Figure 5.

[†] If $\mathfrak{s} = a_1 \dots a_n$ is a string of elements $a_1, \dots, a_n$, then $\lfloor \mathfrak{s} \rfloor_{i_1 \dots i_n}$ denotes the string $a_{i_1} \dots a_{i_n}$.

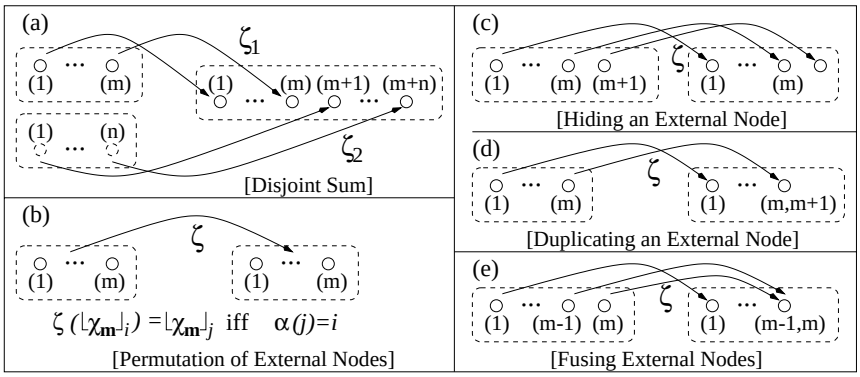

Fig. 5. Converting a graph expression into the corresponding colimit construction.

Permutation of external nodes: Let $\alpha : [m] \to [m]$ be a bijection. In this case ζ is defined as in Figure 5(b).

Hiding an external node: Let $\alpha : [m] \to [m+1]$ where $\alpha(i) = i$. Then ζ is defined as in Figure 5(c).

Duplicating an external node: Let $\alpha : [m+1] \to [m]$ where $\alpha(i) = i$ if $i \in [m]$ and $\alpha(m+1) = m$. Then ζ is defined as in Figure 5(d).

Fusing external nodes: Let δ be an equivalence relation on $[m]$ where $m = ar(H)$, $\theta_\delta(H)$ is obtained by fusing all external nodes that are related by δ. The arity of the hypergraph is not changed.

According to Bauderon and Courcelle (1987), $\theta_\delta(H) \cong \theta_{\delta_k}(\dots\theta_{\delta_1}(H)\dots)$ where each δ_l is an equivalence generated by a single pair (i, j) with $i, j \in [m]$. With the permutation operation defined above, it suffices to define a colimit construction fusing the last two nodes of a hypergraph. Let δ' be the equivalence on $[m]$ generated by the pair $(m-1, m)$. Then $\theta_{\delta'}(H) \cong \otimes(H, \zeta)$ where ζ is defined as in Figure 5(e).

We have shown how to emulate all three operators by colimits. It is still left to show how the subsequent application of colimits can be converted into one single colimit construction. We will delay this until Section 4.

If, on the other hand, we are given discrete morphisms $\zeta_i : \mathbf{m_i} \to D$, $i \in [n]$, we can construct a corresponding graph expression $g(X_1, \dots, X_n)$ as follows:

$$g(X_1, \dots, X_n) = \sigma_\alpha(\theta_\delta(X_1 \oplus \dots \oplus X_n \oplus \mathbf{k}))$$

where $V_D = \{v_1, \dots, v_k\}$, $N_i = \sum_{j=1}^{i} ar(H_i)$, δ is the equivalence generated by the set of pairs $\{(N_{i-1} + j, N_n + l) \mid \zeta_i(\lfloor\chi_{\mathbf{m_i}}\rfloor_j) = v_l\}$ and $\alpha : [ar(D)] \to [N_n + k]$ with $\alpha(i) = N_n + j$ if $\lfloor\chi_D\rfloor_i = v_j$.

By comparing the expression above to the alternative constructive definition of hypergraph construction given in Proposition 1, it can be shown that for all hypergraphs $H_1, \dots, H_n$ with $ar(H_i) = m_i$ we have $g(H_1, \dots, H_n) \cong \bigotimes_{i=1}^{n}(H_i, \zeta_i)$.

3.2. *The double-pushout approach*

The double-pushout approach to graph rewriting was first introduced by Ehrig (Ehrig 1979), while the double-pushout approach to *hypergraph* rewriting was presented in Habel (1992). A rewriting rule consists of two hypergraph morphisms $\phi_L : M \to L$ and $\phi_R : M \to R$ (where L, R are called the left- and right-hand sides, respectively, and M is called the interface). A morphism $\eta : L \to G$ is called an occurrence of L in G. The hypergraph G can be rewritten to G' if and only if we can find a hypergraph K and morphisms such that the diagram in Figure 6(d) consists of two pushouts.

Our colimit construction is closely related to this approach. We will now make this connection precise. First, we define the notion of a rewriting step in our setting: let $r = (L, R)$ be a rewriting rule, where L, R are hypergraphs with $ar(L) = ar(R)$. Then $\overset{r}{\Longrightarrow}$ is the smallest relation that is generated by the following two rules and is closed under isomorphism.

$$L \overset{r}{\Longrightarrow} R \qquad \frac{H_1 \overset{r}{\Longrightarrow} H_1'}{(H_1, \zeta_1) \otimes (H_2, \zeta_2) \overset{r}{\Longrightarrow} (H_1', \zeta_1) \otimes (H_2, \zeta_2)} \quad \text{if } m_i = ar(H_i), \zeta_i : \mathbf{m_i} \to D, i \in [2]$$

In Ehrig *et al.* (1973) it was shown that every double-pushout can be converted into a double-pushout where the interface M in the production span is discrete, without changing the transition relation. In terms of the set of productions this ordinarily means replacing every rule by a (finite) set of rules, in order to handle morphisms η that are non-injective on the edges. We can also assume that all the nodes of M are external, that it is, therefore, isomorphic to $\mathbf{m}$, and that there are strong morphisms $\phi_L : \mathbf{m} \twoheadrightarrow L$ and $\phi_R : \mathbf{m} \twoheadrightarrow R$, which may involve changing the the sequences of external nodes of L and R.

Note that the double-pushout approaches with a discrete interface and with an arbitrary interface are only equivalent in an interleaving semantics. Whenever a truly concurrent semantics is considered, the corresponding rules with the discrete interface in general allow strictly fewer concurrent rewrites.

Proposition 2. Let G, G' be hypergraphs of arity 0 and let $r = (L, R)$ be a rewriting rule. Then $G \overset{r}{\Longrightarrow} G'$ if and only if G can be transformed into G' by r in the double-pushout approach (with a production span $L \twoheadleftarrow \mathbf{m} \twoheadrightarrow R$ where $m = ar(L) = ar(R)$).

Proof. It has already been shown in Bauderon and Courcelle (1987) that the expressive power of rewriting in terms of graph expressions is equal to the expressive power of the double-pushout approach. The proposition follows from the fact that graph expressions are equivalent to our form of graph construction (see Section 3.1).

We now show the proposition in a more direct way and explain how the double-pushout approach is related to our approach:

— First, it can be shown by using the universal property of colimits that if H is a colimit of $\zeta_1, \zeta_2, \phi_1, \phi_2$ in Figure 6(a), and 6(b) consists of two pushouts (1) and (2), then $H \cong H'$. This is also known as the butterfly lemma.
Diagrams 6(a) and 6(b) can always be completed to form colimits, since we can

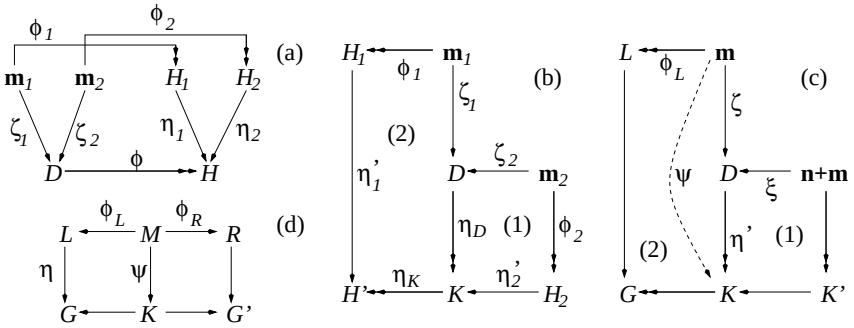

Fig. 6. Converting hypergraph construction into a pushout.

always explicitly construct the colimit as in Proposition 1. In particular, the pushouts in Figure 6(b) exist.

— We now assume that $G \overset{(L,R)}{\Longrightarrow} G'$. Because of Proposition 3, which will be shown in Section 4 (without referring back to this proposition), we can assume that each of the two rules generating $\overset{(L,R)}{\Longrightarrow}$ is used exactly once. Thus we can conclude that $G \cong (L, \zeta_1) \otimes (J, \zeta_2)$ and $G' \cong (R, \zeta_1) \otimes (J, \zeta_2)$ for some hypergraph J and suitable discrete morphisms ζ_1, ζ_2.

We set $K = \otimes(J, \zeta_2)$ (as in pushout (1) in Diagram 6(b)). Then, according to the facts we have shown previously, G is the pushout of $\eta_D \circ \zeta_1$ and ϕ_1 (the canonical strong morphism) in Diagram 6(b). (We assume that $H_1 = L$, $H_2 = J$ and $H' = G$.) In the same way, we can describe G' as a pushout of $\eta_D \circ \zeta_1$ and the canonical strong morphism from $\mathbf{m_1}$ into R.

— Now assume we can transform G into G' by applying the production span $L \twoheadleftarrow \mathbf{m} \twoheadrightarrow R$ in the double-pushout approach. A double-pushout has the form shown in Figure 6(d), where the embedding of L into G may also be non-injective on the external nodes of L. We can assume that $M = \mathbf{m}$ for some natural number m and that both ϕ_L and ϕ_R are strong.

We assume that K' is the same hypergraph as K with a different sequence of external nodes. We obtain K' by replacing the sequence of external nodes of K by the concatenation of χ_K and $\psi(\chi_\mathbf{m})$. Let $\xi : \mathbf{n} + \mathbf{m} \to D$ (where $n = ar(K)$) be a discrete morphism such that $K \cong \otimes(K', \xi)$ (see pushout (1) in Figure 6(c)), that is, ξ hides the last m nodes of K' (compare with Section 3.1). Now let $\zeta : \mathbf{m} \to D$ be a discrete morphism such that $\zeta(\chi_\mathbf{m}) = \xi(\lfloor \chi_{\mathbf{n}+\mathbf{m}} \rfloor_{n+1 \ldots n+m})$, and, therefore, $\eta'(\zeta(\chi_\mathbf{m})) = \eta'(\xi(\lfloor \chi_{\mathbf{n}+\mathbf{m}} \rfloor_{n+1 \ldots n+m})) = \psi(\chi_\mathbf{m})$, and thus $\eta' \circ \zeta = \psi$ (η' is generated by pushout (1) in Figure (c)).

This implies that $G \cong (L, \zeta) \otimes (K', \xi)$. In the same way, we can show that $G' \cong (R, \zeta) \otimes (K', \xi)$. It thus follows that $G \overset{(L,R)}{\Longrightarrow} H$. □

In this section we have only considered the case where rewriting is closed under binary graph construction. However, this is equivalent to the general case (see also Proposition 3).

We have not yet treated the problem of discovering a left-hand side L in a graph H, that is, under what conditions do there exist discrete morphisms ζ_1, ζ_2 and a graph K

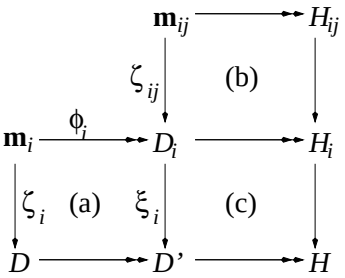

Fig. 7. Collapsing levels of hypergraph construction.

such that $H \cong (L, \zeta_1) \otimes (K, \zeta_2)$? Since our approach coincides with the double-pushout approach, ζ_1, ζ_2 and K exist whenever there is a morphism $\eta : L \to H$ satisfying the dangling edge condition and the identification condition of the double-pushout approach, in other words: η must be an embedding (see Definition 1).

4. Some properties of hypergraph construction

As promised in the previous section, we now introduce a mechanism for combining several construction operations into one by collapsing hierarchies of graph constructions. In the world of strings this has a rough analogue in the associativity of concatenation, which does not hold for graph construction.

Proposition 3. In the following, let i range over $[n]$ and j range over $[n_i]$. Let $\zeta_{ij} : \mathbf{m_{ij}} \to D_i$ and $\zeta_i : \mathbf{m_i} \to D$ be morphisms with $m_i = ar(D_i)$. Let $\phi_i : \mathbf{m_i} \twoheadrightarrow D_i$ be the unique strong morphisms and let the ξ_i be the morphisms generated by colimit (a) in Figure 7. Then we have for arbitrary hypergraphs H_{ij} with $m_{ij} = ar(H_{ij})$ that

$$\bigotimes_{i=1}^{n}\left(\bigotimes_{j=1}^{n_i}(H_{ij}, \zeta_{ij}), \zeta_i\right) \cong \bigotimes_{i,j}(H_{ij}, \xi_i \circ \zeta_{ij}). \tag{1}$$

Proof. The proof of the proposition is shown in Figure 7. The left-hand side of Equation (1) is formed by applying first colimits (b) to the H_{ij}, and then colimit (a)+(c). If (a)+(c) is a colimit and (a) is a colimit, then it follows from the standard properties of colimits that (c) is also a colimit, which implies that (b)+(c), which corresponds to the right-hand side of Equation (1), is also a colimit. □

Example: We will translate a sequence of redefinitions of external nodes (see Section 3.1) into graph constructions and collapse them into one single application of a colimit. Assume we want to compute $\sigma_\alpha(H)$ where $ar(H) = 3$, $\alpha : [3] \to [3]$ and $\alpha(1) = 2$, $\alpha(2) = \alpha(3) = 1$. As mentioned in Section 3.1, we can decompose α into $\alpha = \alpha_1 \circ \alpha_2 \circ \alpha_3$ with $\alpha_1 : [2] \to [3]$ (hiding the last external node), $\alpha_2 : [2] \to [2]$ (permutation, exchanging the first and the second node) and $\alpha_3 : [3] \to [2]$ (duplicating the last external node).

We can now construct the respective discrete morphisms ζ_1, ζ_2 and ζ_3 according to Figure 5, and we have $\sigma_{\alpha_1 \circ \alpha_2 \circ \alpha_3}(H) \cong \otimes(\otimes(\otimes(H, \zeta_1), \zeta_2), \zeta_3)$. If we combine them according

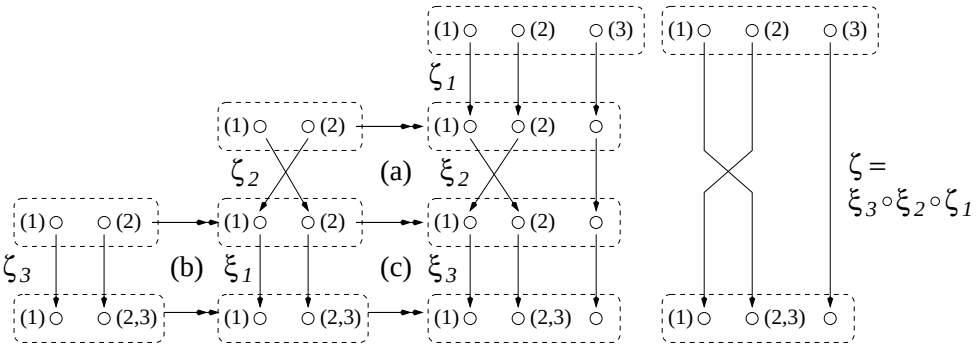

Fig. 8. Collapsing levels of hypergraph construction (example).

to Proposition 3, we obtain the result ζ, shown in Figure 8 where the squares marked (a), (b), (c) are colimits. That is, $\sigma_\alpha(H) \cong \otimes(H, \zeta)$ for every hypergraph H of arity 3.

Hyperedges are the basic units of graph construction. Just as every element of a vector space can be decomposed into base vectors or every natural number can be factored into primes in a unique way, there is a unique decomposition of every hypergraph into hyperedges, where isolated nodes are created by nodes of the discrete hypergraph D that are not in the range of any of the ζ_i.

Proposition 4. (Unique factorisation) Let H be a hypergraph. Then there exists a natural number n, labels l_i and morphisms $\zeta_i : \mathbf{m_i} \to D$ (where $i \in [n]$ and D is a discrete hypergraph) such that $H \cong \bigotimes_{i=1}^{n}([l_i]_{m_i}, \zeta_i)$. This factorisation is unique up to isomorphism and index permutation.

More specifically, if there are other discrete morphisms $\zeta'_i : \mathbf{m'_i} \to D'$, $i \in [n']$ and labels $l'_1, \ldots, l'_{n'}$ such that $H \cong \bigotimes_{i=1}^{n'}([l'_i]_{m'_i}, \zeta'_i)$, then $n = n'$, and there are a permutation $\alpha : [n] \to [n']$ and an isomorphism $\psi : D' \twoheadrightarrow D$ such that $l_i = l'_{\alpha(i)}$, $m_i = m'_{\alpha(i)}$ and $\zeta_i = \psi \circ \zeta'_{\alpha(i)}$.

Proof. By the results of Section 3.1, (Bauderon and Courcelle 1987) and Proposition 3, it is straightforward to see that such a factorisation of a hypergraph H is always possible (isolated nodes in H are generated by nodes of the discrete graph D that are not in the range of any of the ζ_i).

We will now show that the factorisation is unique. Let $H \cong \bigotimes_{i=1}^{n}([l_i]_{m_i}, \zeta_i)$ and $H \cong \bigotimes_{i=1}^{n'}([l'_i]_{m'_i}, \zeta'_i)$ where $\zeta_i : \mathbf{m_i} \to D$, $i \in [n]$ and $\zeta'_i : \mathbf{m'_i} \to D'$, $i \in [n']$, respectively. Furthermore, let $\phi_i : \mathbf{m_i} \twoheadrightarrow [l_i]_{m_i}$ and $\phi'_i : \mathbf{m'_i} \twoheadrightarrow [l'_i]_{m'_i}$ be the canonical strong morphisms. Let the morphisms generated by the colimits be $\eta_i : [l_i]_{m_i} \to H$, $i \in [n]$ and $\phi : D \twoheadrightarrow H$, and $\eta'_i : [l'_i]_{m'_i} \to H$, $i \in [n']$ and $\phi' : D' \twoheadrightarrow H$, respectively.

Since n and n', respectively, denote the number of edges of H, it follows that $n = n'$. The η_i and η'_i, respectively, cover all the edges of H and a morphism with a source graph of the form $[l]_m$ is already fully determined if the image of the only edge in $[l]_m$ is fixed. Therefore, the two sets $\{\eta_i \mid i \in [n]\}$ and $\{\eta'_i \mid i \in [n']\}$ are equal, and it follows that there

is a permutation $\alpha : [n] \to [n]$ such that $\eta_i = \eta'_{\alpha(i)}$, which implies that $m_i = m'_{\alpha(i)}$, $l_i = l'_{\alpha(i)}$ and $\phi_i = \phi_{\alpha(i)}$.

It is left to show that ζ_i and $\zeta'_{\alpha(i)}$ are equal up to isomorphism, that is, that there is an isomorphism $\psi : D' \twoheadrightarrow D$ such that $\zeta_i = \psi \circ \zeta'_{\alpha(i)}$. We first prove that ϕ_V is a bijection (the same holds for ϕ'_V with the same arguments): from Proposition 1 it follows that a hypergraph isomorphic to H can be constructed by taking the union of D and the edges $[l_i]_{m_i}$ and fusing the nodes according to the equivalence $\approx$. Because of the special nature of the hypergraphs that are concatenated (the $[l_i]_{m_i}$ have no duplicates in their sequences of external nodes and no internal nodes), it follows that every node of an edge $[l_i]_{m_i}$ is related to some node in D and that no two distinct nodes of D are related. That is, the equivalence classes are exactly the nodes of D. Since ϕ_V is isomorphic to a function that maps every node of D to its equivalence class, it follows that $\phi_V : V_D \to V_H$ is a bijection and, furthermore,

$$(\zeta_i)_V = \phi_V^{-1} \circ (\eta_i)_V \circ (\phi_i)_V = \phi_V^{-1} \circ (\eta'_{\alpha(i)})_V \circ (\phi'_{\alpha(i)})_V = \phi_V^{-1} \circ \phi'_V \circ (\zeta'_{\alpha(i)})_V.$$

The function $\phi_V^{-1} \circ \phi'_V : V_{D'} \to V_D$ is a bijection and since ζ_i, $\zeta'_{\alpha(i)}$ are defined only on discrete graphs, they are the same up to isomorphism. □

As with vector spaces, we can define linear mappings on hypergraphs.

Definition 3. (Linear mapping) A *linear mapping* L maps every hypergraph to a hypergraph of the same arity and satisfies $L(\bigotimes_{i=1}^{n}(H_i, \zeta_i)) \cong \bigotimes_{i=1}^{n}(L(H_i), \zeta_i)$.

Proposition 5. (Unique linear mapping) For each mapping of hyperedges $[l]_m$ to hypergraphs of arity m, there is exactly one linear mapping (up to isomorphism) that is an extension of the original mapping.

Proof. Let L' map every hyperedge to a hypergraph of the same arity.

— We first show that there is at most one linear mapping L that extends L': let H be an arbitrary hypergraph and let $H \cong \bigotimes_{i=1}^{n}([l_i]_{m_i}, \zeta_i)$ be the unique factorisation of H according to Proposition 4. Any linear mapping L extending L' must satisfy $L(H) \cong \bigotimes_{i=1}^{n}(L'([l_i]_{m_i}), \zeta_i)$, and $L(H)$ is fixed up to isomorphism.

— We now show that there is at least one linear mapping L. We define L as follows:

$$L(\bigotimes_{i=1}^{n}([l_i]_{m_i}, \zeta_i)) = \bigotimes_{i=1}^{n}(L'([l_i]_{m_i}), \zeta_i).$$

This is well-defined since the factorisation of every hypergraph is unique. It is left to show that L is a linear mapping: let $H \cong \bigotimes_{i=1}^{n}(H_i, \zeta_i)$. According to Proposition 4, $H_i \cong \bigotimes_{j=1}^{n_i}([l_{ij}]_{m_{ij}}, \zeta_{ij})$ for suitable $l_{ij}, m_{ij}, \zeta_{ij}$. It follows that

$$L(H) \cong L(\bigotimes_{i=1}^{n}(\bigotimes_{j=1}^{n_i}([l_{ij}]_{m_{ij}}, \zeta_{ij}), \zeta_i)).$$

From Proposition 3, we have the existence of morphisms ξ_i such that

$$\begin{aligned} L(H) &\cong L(\bigotimes_{i,j}([l_{ij}]_{m_{ij}}, \xi_i \circ \zeta_{ij})) = \bigotimes_{i,j}(L'([l_{ij}]_{m_{ij}}), \xi_i \circ \zeta_{ij}) \\ &\cong \bigotimes_{i=1}^{n}(\bigotimes_{j=1}^{n_i}(L'([l_{ij}]_{m_{ij}}), \zeta_{ij}), \zeta_i) \cong \bigotimes_{i=1}^{n}(L(H_i), \zeta_i). \end{aligned}$$

□

One can view the application of a linear mapping as a synchronous rewriting step, replacing every hyperedge at the same time.

Examples: We define a mapping that duplicates every edge in a hypergraph. If $H = (V, E, s, l, \chi)$ is a hypergraph, $Dupl(H)$ is defined by $(V, E \cup \bar{E}, s \cup \bar{s}, l \cup \bar{l}, \chi)$ where $\bar{E} = \{\bar{e} \mid e \in E\}$, $\bar{s} : \bar{E} \to V^*$ with $\bar{s}(\bar{e}) = s(e)$ and $\bar{l} : \bar{E} \to L$ with $\bar{l}(\bar{e}) = l(e)$. The function $Dupl$ is a linear mapping and can be generated by fixing the images of single hyperedges: $Dupl([l]_m) = [l]_m \square [l]_m$ (where $\square$ is the operator defined in Example (2) at the end of Section 2). Note that a mapping that duplicates all nodes is not linear.

Another simple example is a mapping that deletes all edges, that is, produces a discrete graph. We define $Discrete(H) = (V, \varnothing, \varnothing, \varnothing, \chi)$. It is linear and can be generated by defining $Discrete([l]_m) = \mathbf{m}$ for all hyperedges.

The usefulness of linear mappings will become clear in the following section where we will use a linear mapping in order to analyse mobile processes. Another important use of linear mappings is to annotate hypergraphs. In order to do this we will slightly extend the notion of a linear mapping in Section 6.

5. Typing process graphs

We show how to model a process calculus, which is closely related to the asynchronous polyadic π-calculus (Milner 1993), using so-called process graphs. There is an encoding from the π-calculus into process graphs and *vice versa* (König 2000). On the other hand, there is a straightforward encoding of process graphs into closed action calculi (Gardner 1998) and a close relation of our process graphs to the ones in Yoshida (1994).

As a second step we will then show how to associate process graphs with abstract behaviour descriptions, which are closely related to type systems in process calculi.

Definition 4. (**Process graph**) A *process graph* P is inductively defined as follows: P is a hypergraph where each edge e is either labelled with $(n)Q$ where Q is again a process graph and $1 \leqslant n \leqslant ar(Q)$ (e is a process waiting for a message with n ports arriving at its first node), with $!Q$ (e is a process that can replicate, creating arbitrarily many instances of Q) or with the constant M (e is a message sent to its last node). The reduction relation involving the reception of a message and its nodes by a process, that is, communication, and replication is generated by the rewrite rules in Figure 9 and is closed under graph construction. (The operator $\square$ is defined in Example (2) at the end of Section 2.)

The rewrite rules in Figure 9 are not always defined, since there might be arity mismatches between the left-hand and right-hand side. They may fail if $ar(Q) \neq m + n'$

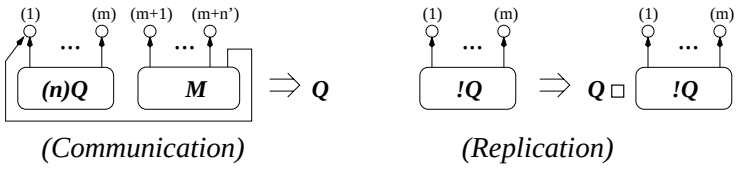

Fig. 9. Communication and replication of process graphs.

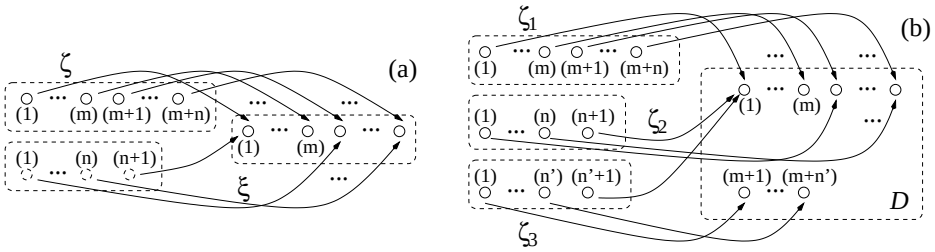

Fig. 10. Some morphisms needed in Definition 5 and for the proof of Proposition 6.

($*_1$) in the communication rule or $ar(Q) \neq m$ ($*_2$) in the replication rule. Furthermore, we want to avoid the occurrence of $n \neq n'$ ($*_3$) in the communication rule, that is we want to ensure that the expected number of nodes is received. We say that a process graph P contains a runtime error if it contains the left-hand side of the communication rule and either ($*_1$) or ($*_3$) hold, or if it contains the left-hand side of the replication rule and ($*_2$) holds.

We use morphisms, graph construction and a linear mapping in order to define a condition that is sufficient for avoiding these runtime errors and can be checked statically.

Definition 5. (Type graph) Let L be a linear mapping that is defined on the hyperedges as follows:

$$\begin{aligned} L([M]_n) &= [t]_n \quad (t \text{ is a new edge label}) \\ L([!Q]_m) &= L(Q) \quad \text{if } m = ar(Q) \text{ (undefined otherwise)} \\ L([(n)Q]_m) &= (L(Q), \zeta) \otimes ([t]_{n+1}, \xi) \quad \text{if } n + m = ar(Q) \text{ (undefined otherwise).} \end{aligned}$$

The morphisms ζ, ξ are given in Figure 10(a).

Now let P be a process graph. If there exists a strong morphism $\psi : L(P) \twoheadrightarrow T$ into a hypergraph T that satisfies

$$\forall\, e_1, e_2 \in E_T : (\lfloor s_T(e_1) \rfloor_{ar(e_1)} = \lfloor s_T(e_2) \rfloor_{ar(e_2)} \;\Rightarrow\; e_1 = e_2) \tag{2}$$

(that is, all edges that share the last node are already the same), then T is called the *type graph* of P.

The aim is to show that every process graph that can be typed does not produce runtime errors of the form described above. The linear mapping L extracts pure communication structure from a process graph, that is, an edge of the form $[t]_n$ indicates that its nodes

(except the last) might be sent or received via its last node. Condition (2) makes sure that the arity of the arriving message matches the expected arity, and that nodes that might get fused during reduction are already fused in T. It thus guarantees the absence of undefined rewrites for the entire reduction.

We can easily unfold T into well-known type trees of π-calculus processes (Pierce and Sangiorgi 1993; Turner 1995). Therefore we decided to call T a *type graph* of P, although this term clashes with a different form of type graph used in typed graph grammars (Corradini *et al.* 1996).

Proposition 6. If a process graph P has a type graph, it will never encounter a runtime error, that is, if $P \Longrightarrow^* P'$, then P' does not contain a runtime error.

Proof. Let P be a process graph with $\psi : L(P) \twoheadrightarrow T$ where T satisfies Condition (2).

We now show that P does not encounter a runtime error in its next reduction step, along with the fact that the subject reduction property is satisfied. The subject reduction property says that if $P \Longrightarrow P'$, there is also a strong morphism $\psi' : L(P') \twoheadrightarrow T$. These two properties together ensure the absence of runtime errors for the entire reduction.

We proceed by induction on the reduction rules:

— Let P be the left-hand side of the communication rule in Figure 9 and $P \Longrightarrow Q$. From Proposition 3 it follows that $L(P) \cong (L(Q), \zeta_1) \otimes ([t]_{n+1}, \zeta_2) \otimes ([t]_{n'+1}, \zeta_3)$ where $\zeta_1, \zeta_2, \zeta_3$ are defined in Figure 10(b).
The condition in the definition of L tells us that $m + n = ar(Q)$, and since we will later show that $n = n'$, we have thus eliminated runtime error $(*_1)$.
Now let $\eta_1 : L(Q) \to L(P)$, $\eta_2 : [t]_{n+1} \to L(P)$, $\eta_3 : [t]_{n'+1} \to L(P)$ be the embeddings into $L(P)$ generated by the colimit. Furthermore, we know that there exists a strong morphism $\psi : L(P) \twoheadrightarrow T$. Our aim is to show that $\psi \circ \eta_1 : L(Q) \to T$ is the strong morphism we are looking for. We proceed in two steps:

– We first show that $\psi(\eta_1(\lfloor \chi_{L(Q)} \rfloor_{1\ldots m})) = \lfloor \chi_T \rfloor_{1\ldots m}$:

$$\begin{aligned} \psi(\eta_1(\lfloor \chi_{L(Q)} \rfloor_{1\ldots m})) &= \psi(\eta_1(\phi_1(\lfloor \chi_{\mathbf{m}+\mathbf{n}} \rfloor_{1\ldots m}))) \\ &= \psi(\phi(\zeta_1(\lfloor \chi_{\mathbf{m}+\mathbf{n}} \rfloor_{1\ldots m}))) \\ &= \psi(\phi(\lfloor \chi_D \rfloor_{1\ldots m})) \\ &= \psi(\lfloor \chi_{L(P)} \rfloor_{1\ldots m}) \\ &= \lfloor \chi_T \rfloor_{1\ldots m} \end{aligned}$$

where ϕ_1 is the canonical strong morphism from $\mathbf{m} + \mathbf{n}$ into $L(Q)$ and $\phi : D \twoheadrightarrow L(P)$ is the strong morphism generated by the colimit. We then have by construction that $\zeta_1(\lfloor \chi_{\mathbf{m}+\mathbf{n}} \rfloor_{1\ldots m}) = \lfloor \chi_D \rfloor_{1\ldots m}$.

– In the next step we show that $\psi(\eta_1(\lfloor \chi_{L(Q)} \rfloor_{m+1\ldots m+n})) = \lfloor \chi_T \rfloor_{m+1\ldots m+n'}$, and thus $n = n'$, which avoids runtime error $(*_3)$.
Let e be the only edge in $[t]_{n+1} = U$ and let e' be the only edge in $[t]_{n'+1} = U'$. It

follows that

$$\begin{aligned}
\lfloor s_T(\psi(\eta_2(e)))\rfloor_{ar(e)} &= \lfloor \psi(\eta_2(s_U(e)))\rfloor_{n+1} \\
&= \lfloor \psi(\eta_2(\chi_U))\rfloor_{n+1} \\
&= \lfloor \psi(\eta_2(\phi_2(\chi_{\mathbf{n+1}})))\rfloor_{n+1} \\
&= \psi(\phi(\zeta_2(\lfloor \chi_{\mathbf{n+1}}\rfloor_{n+1}))) \\
&= \psi(\phi(\zeta_3(\lfloor \chi_{\mathbf{n'+1}}\rfloor_{n'+1}))) \\
&= \lfloor \psi(\eta_3(\phi_3(\chi_{\mathbf{n'+1}})))\rfloor_{n'+1} \\
&= \lfloor \psi(\eta_3(\chi_{U'}))\rfloor_{n'+1} \\
&= \lfloor \psi(\eta_3(s_{U'}(e')))\rfloor_{n'+1} \\
&= \lfloor s_T(\psi(\eta_3(e')))\rfloor_{ar(e')}
\end{aligned}$$

where ϕ_2 and ϕ_3, respectively, are the canonical strong morphisms from $\mathbf{n}+\mathbf{1}$ into $[t]_{n+1}$ and from $\mathbf{n}'+\mathbf{1}$ into $[t]_{n'+1}$, respectively. We again have by construction that $\zeta_2(\lfloor \chi_{\mathbf{n+1}}\rfloor_{n+1}) = \zeta_3(\lfloor \chi_{\mathbf{n'+1}}\rfloor_{n'+1})$.

Condition (2) implies that $\psi(\eta_2(e)) = \psi(\eta_3(e'))$. It follows that

$$n+1 = ar(e) = ar(\psi(\eta_2(e))) = ar(\psi(\eta_3(e'))) = ar(e') = n'+1$$

and thus $n = n'$. Now

$$\begin{aligned}
\psi(\eta_1(\lfloor \chi_{L(Q)}\rfloor_{m+1\ldots m+n})) &= \psi(\eta_1(\phi_1(\lfloor \chi_{\mathbf{m+n}}\rfloor_{m+1\ldots m+n}))) \\
&= \psi(\phi(\zeta_1(\lfloor \chi_{\mathbf{m+n}}\rfloor_{m+1\ldots m+n}))) \\
&= \psi(\phi(\zeta_2(\lfloor \chi_{\mathbf{n+1}}\rfloor_{1\ldots n}))) \\
&= \psi(\eta_2(\phi_2(\lfloor \chi_{\mathbf{n+1}}\rfloor_{1\ldots n}))) \\
&= \psi(\eta_2(\lfloor s_U(e)\rfloor_{1\ldots n})) \\
&= \lfloor s_T(\psi(\eta_2(e)))\rfloor_{1\ldots n} \\
&= \lfloor s_T(\psi(\eta_3(e')))\rfloor_{1\ldots n} \\
&= \psi(\eta_3(\lfloor s_{U'}(e')\rfloor_{1\ldots n})) \\
&= \psi(\eta_3(\phi_3(\lfloor \chi_{\mathbf{n'+1}}\rfloor_{1\ldots n}))) \\
&= \psi(\phi(\zeta_3(\lfloor \chi_{\mathbf{n'+1}}\rfloor_{1\ldots n}))) \\
&= \psi(\phi(\lfloor \chi_D\rfloor_{m+1\ldots m+n'})) \\
&= \lfloor \chi_T\rfloor_{m+1\ldots m+n'}.
\end{aligned}$$

Taking these together, we conclude that $\psi(\eta_1(\chi_{L(Q)})) = \chi_T$, and $\psi \circ \eta_1$ is thus a strong morphism.

— Let $P = [!Q]_m$ be the left-hand side of the replication rule in Figure 9, $P \Longrightarrow P'$ and $P' = Q\Box[!Q]_m$. Furthermore, we have $L(P) = L(Q)$ and $L(P') = L(Q)\Box L(Q)$. Since $L(P) = L(Q)$ is defined, it follows that $ar(Q) = m$, and therefore runtime error $(*_2)$ is eliminated and $L(P')$ is defined. We have to show that there exists a strong morphism $\psi : L(Q)\Box L(Q) \twoheadrightarrow L(Q)$. (Recall that $L(Q)\Box L(Q) = (L(Q), \zeta) \otimes (L(Q), \zeta)$ where $\zeta : \mathbf{m} \twoheadrightarrow \mathbf{m}$.)

First there is the identity morphism $id : L(Q) \twoheadrightarrow L(Q)$ (see Figure 11(a)). Furthermore,

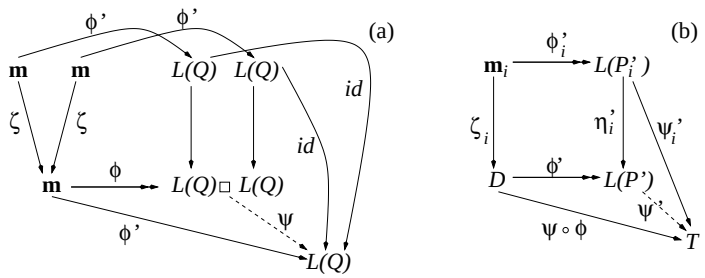

Fig. 11. Diagrams for the proof of Proposition 6.

there is a unique strong morphism $\phi' : \mathbf{m} \twoheadrightarrow L(Q)$, and we have $\phi' \circ \zeta = id \circ \phi'$, since ζ is strong. Now let $\phi : \mathbf{m} \twoheadrightarrow L(Q)\square L(Q)$ be the morphism generated by the colimit. The properties of a colimit imply the existence of a morphism $\psi : L(Q)\square L(Q) \to L(Q)$ such that $\psi \circ \phi = \phi'$. Since ϕ and ϕ' are both strong, it follows that ψ is also a strong morphism.

— Let $P_1 \Longrightarrow P_1'$, $P_2 \cong P_2'$ and

$$P = (P_1, \zeta_1) \otimes (P_2, \zeta_2) \Longrightarrow (P_1', \zeta_1) \otimes (P_2', \zeta_2) = P' \quad \text{where } \zeta_i : \mathbf{m_i} \to D,\ i \in [2].$$

We know that $L(P) \cong (L(P_1), \zeta_1) \otimes (L(P_2), \zeta_2)$. Let $\eta_i : L(P_i) \to L(P)$ be the embeddings and let $\phi : D \twoheadrightarrow L(P)$ be the strong morphism generated by the colimit. It follows that[†]

$$\psi \circ \eta_i : L(P_i) \twoheadrightarrow T\langle \psi(\eta_i(\chi_{L(P_i)}))\rangle.$$

The induction hypothesis implies that there is a strong morphism $\psi_1' : L(P_1') \twoheadrightarrow T\langle \psi(\eta_1(\chi_{L(P_1)}))\rangle$. Furthermore, we set $\psi_2' = \psi \circ \eta_2$.

Now let $\phi_i' : \mathbf{m_i} \twoheadrightarrow L(P_i')$ be the canonical strong morphisms, and let $\eta_i' : L(P_i') \to L(P')$ and $\phi' : D \twoheadrightarrow L(P')$ be the morphisms generated by the colimit $L(P') \cong (L(P_1'), \zeta_1) \otimes (L(P_2'), \zeta_2)$ (see Figure 11(b)).

We know that $\psi_i' : L(P_i') \to T$ and $\psi \circ \phi : D \to T$. If we can show that $\psi_i' \circ \phi_i' = (\psi \circ \phi) \circ \zeta_i$, the properties of the colimit guarantee the existence of a morphism $\psi' : L(P') \to T$. And since $\psi' \circ \phi' = \psi \circ \phi$ and ψ, ϕ, ϕ' are strong, it follows that ψ' is also strong.

So it is left to show that $\psi_i' \circ \phi_i' = (\psi \circ \phi) \circ \zeta_i$ (see Figure 11(b)):

$$\psi_i'(\phi_i'(\chi_{\mathbf{m_i}})) = \psi_i'(\chi_{L(P_i')}) = \psi(\eta_i(\chi_{L(P_i)})) = \psi(\phi(\zeta_i(\chi_{\mathbf{m_i}}))).$$ □

A process graph may have several type graphs, so the question arises whether one of them is 'minimal' in a certain sense. Let F be the inclusion functor from the category of hypergraphs (with strong hypergraph morphisms) satisfying Condition (2) in Definition 5 into **SHGraph**, mapping every hypergraph and every morphism to itself. It can be shown that if P has a type graph, then the comma category $(L(P) \downarrow F)$ (containing all strong morphisms of the form $L(P) \twoheadrightarrow F(T')$) contains an initial element $\psi : L(P) \twoheadrightarrow F(T)$. This

[†] By $H\langle\chi'\rangle$, we denote the hypergraph we obtain from a hypergraph H by replacing χ_H with χ'.

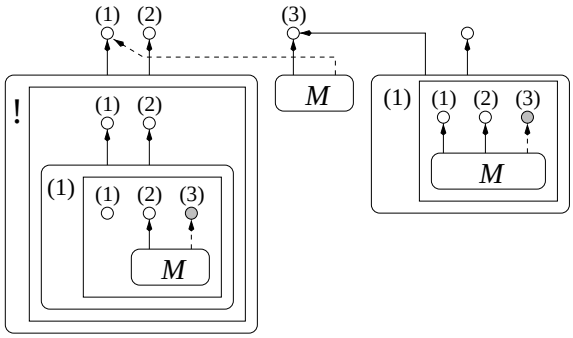

Fig. 12. Example of a process graph.

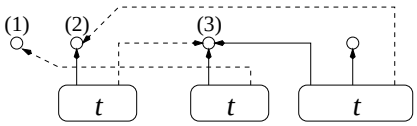

Fig. 13. Type graph.

hypergraph T can be considered as the minimal or principal type of P, and there is an algorithm computing T for a given type P.

As an example, consider the process graph in Figure 12. There is a server (the process on the left) that receives messages on its first port (before the server can receive a message it must replicate itself) and sends back its second port to the address that was attached to the message. At this address another process (the process on the right) is waiting, it receives the message with the second port of the server and sends its own message there.

We use the following syntactic sugar: the last node of a message (that is, the node or port to which the message is sent) is connected to the message by a dashed line. Nodes of a hypergraph in an inner level that will be merged with nodes attached to a message are shaded grey.

If we represent the three external ports by a, b, c, respectively, and denote the internal node by d, the process graph above corresponds to the following process in the polyadic π-calculus (Milner 1993):

$$!a(x).\bar{x}\langle b\rangle \mid \bar{a}\langle c\rangle \mid (\nu d)(c(y).\bar{y}\langle c, d\rangle).$$

Typing the process graph above intuitively means flattening the hypergraph until it consists of only one hierarchy level. Hyperedges representing replicating processes are discarded, messages and processes waiting for a message are appropriately replaced by edges labelled t. After folding the hypergraph (that is, merging the hyperedges according to Condition (2)), we obtain the minimal type graph T depicted in Figure 13.

6. Annotated hypergraphs and their applications

6.1. *Annotating hypergraphs*

Often a pure hypergraph structure (even with the complex hyperedge labels used for process graphs) is not enough to model certain properties. Node labels or extra annotations forming an algebraic structure can be useful.

We will define a method for annotating hypergraphs and specify how these annotations behave under graph morphisms and graph construction. In the following, we present two applications for the concept of annotated hypergraphs: an extended type system for process graphs and the modelling of Petri nets.

We first define a functor $\mathscr{A}$ that assigns to every hypergraph a commutative monoid from which an annotation can be picked. The concept of annotations is deliberately kept very general and we do not specify whether we want to annotate nodes, edges, tuples of nodes or even assign only a single annotation to the entire hypergraph. Furthermore, $\mathscr{A}$ defines how annotations behave under morphisms by mapping every graph morphism to a monoid morphism. This form of graph annotation was introduced in König (1999), where it is used for the definition of generic type systems for process graphs (see Section 6.2).

Annotated hypergraphs are loosely related to attributed graphs (Löwe *et al.* 1993), but are more general in some respects, since we are able to annotate structures different from single nodes or edges, and more restricted in others, since we restrict ourselves to monoids and do not consider other algebraic structures.

Definition 6. (Annotated hypergraphs) Let $\mathscr{A}$ be a functor from the category of hypergraphs with hypergraph morphisms into the category of commutative monoids with monoid morphisms (functions respecting the monoid operation and the unit 0).

That is, $\mathscr{A}$ assigns a commutative monoid $\mathscr{A}(H) = (Mon, +)$ to every hypergraph and a function $\mathscr{A}_\phi : \mathscr{A}(H) \to \mathscr{A}(H')$ to every graph morphism $\phi : H \to H'$ satisfying

$$\mathscr{A}_\phi \circ \mathscr{A}_\psi = \mathscr{A}_{\phi\circ\psi} \qquad \mathscr{A}_{id_H} = id_{\mathscr{A}(H)} \qquad \mathscr{A}_\phi(a+b) = \mathscr{A}_\phi(a) + \mathscr{A}_\phi(b) \qquad \mathscr{A}_\phi(0) = 0$$

where a, b are two elements of the monoid $\mathscr{A}(H)$.

If $a \in \mathscr{A}(H)$, then $H[a]$ is called an annotated hypergraph. Furthermore, $H[a]$ and $H'[a']$ are called isomorphic (with respect to $\mathscr{A}$) if there is a strong bijective morphism ϕ with $\mathscr{A}_\phi(a) = a'$ between them.

We introduce two annotation functors, which will both be used later in the examples.

Example (1): In our first example we want to annotate a hypergraph by fixing a subset of all pairs of nodes. For a given hypergraph H, the commutative monoid is $\mathscr{A}(H) = (\mathscr{P}(V_H^2), \cup)$, that is, the power set of V_H^2 where the monoid operation is simply set union with the empty set as the unit.

Furthermore, for a given monoid element $a \in \mathscr{A}(H)$ and a graph morphism $\phi : H \to H'$, we define $\mathscr{A}_\phi(a) = \{(\phi(v_1), \phi(v_2)) \mid (v_1, v_2) \in a\}$.

Example (2): Every node is annotated with an element of a fixed commutative monoid *Mon*, that is, $\mathscr{B}(H) = (\{a : V_H \to Mon\}, +)$ where $+$ is the point-wise application of the

monoid operation. And if $a : V_H \to \mathbb{N}$, $\phi : H \to H'$ and v' is a node of H', we set $\mathscr{B}_\phi(a)(v') = \sum_{\phi(v)=v'} a(v)$.

Now the question arises as to whether we can define a category of annotated hypergraphs in which graph construction is a colimit, analogously to Definition 2. For general commutative monoids it is not obvious how this should be done, so we first give an alternative characterisation. If we restrict ourselves to lattices with the join operation, we can again characterise graph construction by a colimit and the result corresponds to the one in the alternative characterisation.

Definition 7. (Hypergraph construction with annotations) Let $H_i[a_i]$, $i \in [n]$ be hypergraphs annotated with respect to an annotation functor $\mathscr{A}$. We define

$$\bigotimes_{i=1}^{n}(H_i[a_i], \zeta_i) = \bigotimes_{i=1}^{n}(H_i, \zeta_i)[\sum_{i=1}^{n} \mathscr{A}_{\eta_i}(a_i)]$$

where the η_i are the embeddings of the H_i into the constructed graph as defined in Definition 2.

As mentioned before, in the case of lattices, a hypergraph construction can again be characterised by a colimit construction.

Proposition 7. Let $\mathscr{A}$ be an annotation mapping where for every hypergraph H, we have $\mathscr{A}(H) = (I, \vee)$ is a lattice with a bottom element $\perp$ (the unit), the partial order $\leqslant$ and the join operation $\vee$.

We call $\phi : H[a] \to_{\mathscr{A}} H'[a']$ an *$\mathscr{A}$-morphism* if $\phi : H \to H'$ is a morphism and $F_\phi(a) \leqslant a'$. Annotated hypergraphs together with $\mathscr{A}$-morphisms form a category.

Furthermore, if $\zeta_i : \mathbf{m_i} \to D$ and $\phi_i : \mathbf{m_i} \twoheadrightarrow H_i$, $i \in [n]$, then $\bigotimes_{i=1}^{n}(H_i[a_i], \zeta_i)$ is the colimit of $\zeta_i : \mathbf{m_i}[\perp] \to_{\mathscr{A}} D[\perp]$ and $\phi_i : \mathbf{m_i}[\perp] \twoheadrightarrow_{\mathscr{A}} H_i[a_i]$ in this category.

Proof. The proof is straightforward. □

Of the two annotation functors given above, $\mathscr{A}$ in Example (1) maps hypergraphs to lattices and so graph construction can be characterised by a colimit. The same is true for the functor $\mathscr{B}$ if $(Mon, +)$ in Example (2) is a lattice.

We can extend the notion of a linear mapping to annotated hypergraphs: a linear mapping L is now a function mapping hypergraphs without annotations to annotated hypergraphs satisfying, again, the linearity condition of Definition 3. A linear mapping is again uniquely defined, provided it is defined on the hyperedges.

6.2. *Type graphs with annotations*

We now show how to exploit the concept of annotated hypergraphs in order to obtain a refined type system. This time we aim not only to avoid runtime errors but also to check other properties that are related to the graph structure and are invariant under reduction, such as the input/output-behaviour of external nodes or secrecy properties.

In order to achieve this result, we define a linear mapping L as in Definition 5, but every occurrence of a message or a process contributes annotations that are added to the type graph.

If the annotation somehow reflects the property X to be checked, we can infer X for a process graph P if the annotations of $L(P)$ satisfy certain constraints. If these constraints are, furthermore, invariant under inverse graph morphisms and they hold for the type graph $T[a]$, then we know that they hold for all successors of P. A property is invariant under inverse graph morphisms if from the fact that it holds for the target graph of a graph morphism, we can always infer that it also holds for the source graph.

Definition 8. (Annotated type graph) Let $\mathscr{A}$ be an annotation functor that assigns a lattice to every hypergraph. Furthermore, let a_n^M and $a_n^P \in \mathscr{A}(\mathbf{n})$ be annotations associated with messages and processes, respectively, of arity $n \in \mathbb{N}$. We define a linear mapping L assigning annotated hypergraphs to hypergraphs as follows:

$$\begin{aligned} L([M]_n) &= [t]_n[\bot] \,\square\, \mathbf{n}[a_n^M] \\ L([!Q]_m) &= L(Q) \quad \text{if } m = ar(Q) \text{ (undefined otherwise)} \\ L([(n)Q]_m) &= ((L(Q), \zeta) \otimes ([t]_{n+1}[\bot], \xi)) \,\square\, \mathbf{m}[a_m^P] \\ &\qquad \text{if } n + m = ar(Q) \text{ (undefined otherwise).} \end{aligned}$$

(The morphisms ζ, ξ are defined in Figure 10(a).)

Now let P be a process graph. If there exists a strong $\mathscr{A}$-morphism $\psi : L(P) \twoheadrightarrow_{\mathscr{A}} T[a]$ into an annotated hypergraph $T[a]$ that satisfies Condition (2) of Definition 5, then $T[a]$ is called an annotated type graph of P (with respect to $\mathscr{A}, a_n^M, a_n^P$).

Proposition 8. Let $\mathscr{A}$ be an annotation functor and let a_n^M, a_n^P for $n \in \mathbb{N}$ be given as in Definition 8. Let X be a predicate on process graphs and let Y be a predicate on annotated hypergraphs satisfying

$$\begin{aligned} &(\phi : H[a] \twoheadrightarrow_{\mathscr{A}} H'[a'] \;\wedge\; Y(H'[a'])) \;\Rightarrow\; Y(H[a]) \quad \text{for all } \mathscr{A}\text{-morphisms } \phi \\ &Y(L(P)) \Rightarrow X(P) \quad \text{for all process graphs } P. \end{aligned}$$

If a process graph P has an annotated type graph $T[a]$ (with respect to $\mathscr{A}, a_n^M, a_n^P$) and $Y(T[a])$ holds, then $X(P')$ holds for every P' such that $P \Longrightarrow^* P'$.

Proof. Since graph construction is a colimit for hypergraphs annotated with lattice elements, we can show the subject reduction property by adapting the proof of Proposition 6, that is, by showing that if $L(P) \twoheadrightarrow_{\mathscr{A}} T[a]$, $T[a]$ satisfies (2) and $P \Longrightarrow P'$, then $L(P') \twoheadrightarrow_{\mathscr{A}} T[a]$.

Furthermore, if $Y(T[a])$ holds, it follows that $Y(L(P'))$ is satisfied, which implies that X holds for P'. □

This form of type system can be considered as a generic type system, since it can be instantiated by choosing an appropriate annotation functor $\mathscr{A}$, annotations a_n^M and a_n^P and predicates X, Y. The notion of generic type system is new in the area of process calculi and it seems to be easier to define for graphs, since in the case of graphs it is straightforward to add an extra layer of annotation.

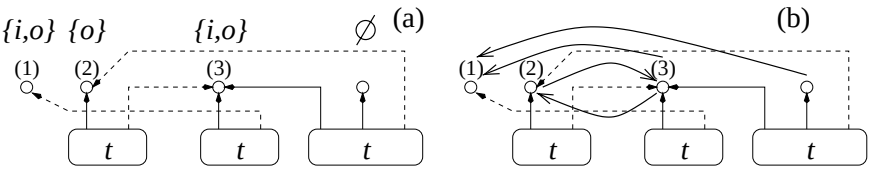

Fig. 14. Type graphs with annotations.

It is an object of further study to determine which properties X are amenable to a static analysis of this kind, and if and how we can automatically derive $Y, \mathscr{A}, a_n^P, a_n^M$, given a property X. Related to this work are properties of hypergraphs compatible with context-free derivations studied in Habel (1992).

We give two examples by deriving useful type systems with different annotation functors.

Input/output capabilities: The aim here is to check which external nodes are used for input, for output or for both. We use the annotation functor $\mathscr{B}$ from Example (2) after Definition 6, where we take $(\mathscr{P}(\{i,o\}),\cup)$ (the power set of $\{i,o\}$ with set union) as the commutative monoid *Mon*. We define

$$a_n^M(\lfloor\chi_n\rfloor_j) = \begin{cases} \varnothing & \text{if } j \neq n \\ \{o\} & \text{if } j = n \end{cases} \qquad a_n^P(\lfloor\chi_n\rfloor_j) = \begin{cases} \{i\} & \text{if } j = 1 \\ \varnothing & \text{if } j \neq 1. \end{cases}$$

Furthermore, we choose the following predicate X: $X(P)$ holds if P does not contain a process that expects a message on the second external node of P. The corresponding predicate Y on annotated hypergraphs is $Y(T[a]) = (a(\lfloor\chi_T\rfloor_2) \subseteq \{o\})$.

It is not hard to check that the components of the type system satisfy the conditions imposed in Proposition 8. Furthermore, it is straightforward to compute the minimal type graph of the example process graph P from Figure 12. The annotated type graph is depicted in Figure 14(a) – it has the same graph structure as the type graph without annotations (see Figure 13).

Since the type graph satisfies predicate Y, it follows that no process will ever expect a message on the second external node during the reduction of P.

Secrecy: We partition the set of external nodes of a process into two sets, secret and public nodes. We demand that no message with a secret node attached to it is ever sent to a public node, that is, if we assume that *Sec* is the set containing the numbers of secret ports, X can be formalised as follows:

$$\begin{aligned} X(P) \quad = \quad & \forall\, e \in E_P : \forall\, 1 \leqslant i < ar(P) : \forall\, 1 \leqslant j,k \leqslant ar(P) : (l_P(e) = M \wedge \\ & \lfloor s_P(e)\rfloor_i = \lfloor\chi_P\rfloor_j \wedge \lfloor s_P(e)\rfloor_{ar(e)} = \lfloor\chi_P\rfloor_k \wedge j \in Sec \Rightarrow\ k \in Sec). \end{aligned}$$

We use type functor $\mathscr{A}$ from Example (1) after Definition 6. Furthermore, we set $a_n^P = \varnothing$ for every n and $a_n^M = \{(\lfloor\chi_{\mathbf{n}}\rfloor_i, \lfloor\chi_{\mathbf{n}}\rfloor_n) \mid i \in [n-1]\}$. We define

$$Y(T[a]) = \forall\, j,k : ((\lfloor\chi_T\rfloor_k, \lfloor\chi_T\rfloor_j) \in a \wedge k \in Sec \Rightarrow\ j \in Sec).$$

Again we can show that $Y(L(P))$ implies $X(P)$ and that all other conditions imposed by Proposition 8 are satisfied.

Typing our running example P from Figure 12, we obtain the annotated type graph $T[a]$ depicted in Figure 14(b). We draw an arrow from node v_1 to node v_2 if and only if (v_1, v_2) is an element of a. Furthermore, we assume that $Sec = \{1\}$, that is, the first external node is secret and all others are public. Since there is no arrow originating in the first external node, $T[a]$ satisfies Y, and therefore no secrecy violations will ever take place.

6.3. *Modelling Petri nets*

We will now show another application for annotated hypergraphs by modelling Petri nets. A Petri net can easily be represented by a hypergraph H (compare with Kreowski (1980) and Löwe *et al.* (1993)): nodes are places and edges are transitions. We first give a semantics for high-level-like nets, where every place is labelled with a monoid element, where the monoid *Mon* could, for example, be the set of all multi-sets over certain elements. By using the natural numbers as a monoid, we obtain standard P/T-nets.

Since we do not distinguish source and target nodes *a priori*, we partition the nodes of an edge into sources and targets with the labelling function. If e is a transition (edge), $l(e) = (s, t, a_1 \dots a_n) \in \mathbb{N} \times \mathbb{N} \times Mon^{s+t}$ with $s + t = ar(e)$ where s is the number of sources (the first s nodes) and t (the last t nodes) is the number of targets. The tuple $(a_1, \dots, a_n)$ of monoid elements indicates which tokens are removed from the sources and which tokens are placed into the targets.

The placement of tokens is given by annotating a hypergraph H with the annotation functor $\mathscr{B}$ defined in Example (2) after Definition 6, that is, the set of nodes is mapped to a commutative monoid *Mon*. We demand that *Mon* is cancellative ($\forall\, a, b, c \in Mon : (a + b = a + c \Rightarrow b = c)$), since it seems natural that if tokens are added and taken away afterwards, this should result in the original number of tokens being present at a place.

Inductive Definition of Petri Nets: A Petri net N is either of the form $[(s, t, a_1 \dots a_n)]_{s+t}[z]$ where $(a_1, \dots, a_n) \in Mon^{s+t}$ and $z \in \mathscr{B}([(s, t, a_1 \dots a_n)]_{s+t})$ or $\bigotimes_{i=1}^{n}(N_i, \zeta_i)$ with adequate discrete morphisms ζ_i, where the N_i are again Petri nets.

Semantics of Petri Nets: A single transition fires if all its source nodes are labelled with appropriate tokens. And if one transition fires, the entire net is changed accordingly. So the reduction relation $\Longrightarrow$ is closed under graph construction and generated by the rule $T[z] \Longrightarrow T[z']$ where $T = [(s, t, a_1 \dots a_n)]_{s+t}$ is a transition, $z(\chi_T) + z'(\chi_T) = (a_1, \dots, a_n)$, $z(\lfloor \chi_T \rfloor_i) = 0$ if $s + 1 \leqslant i \leqslant s + t$ and $z'(\lfloor \chi_T \rfloor_i) = 0$ if $1 \leqslant i \leqslant s$.

If we use the natural numbers as monoid and set $(a_1, \dots, a_n) = (1, \dots, 1)$ for every transition, we obtain exactly the standard P/T-nets. Note that all cases where the places of a firing transition contain more than one token are obtained by the closure under graph construction. We can always add extra tokens by concatenating the transition with a discrete graph whose nodes are labelled by positive integers.

Although we also use monoids to describe Petri nets, they play an entirely different

role from that in Meseguer and Montanari (1990). There, entire Petri nets are given a monoid structure, whereas in our case, monoids are just used to represent sets of tokens.

7. Conclusion

We have presented a method for inductively constructing hypergraphs out of smaller components. The basic units are single edges. Together with a concept for the inductive annotation of hypergraphs, this can be used to define and reason about concurrent systems, whose operational semantics can often be specified by graph transformations in a natural way.

In Section 3.1 we compared our approach to graph expressions (Bauderon and Courcelle 1987), which are also a method for the inductive definition of hypergraphs. Since we need to define morphisms between graphs, and since in the proofs we depend on the universal property provided by a colimit, we have chosen to base our construction operator on colimits rather than on an equational theory.

So this approach is closely connected to the double-pushout approach, which we demonstrated in Section 3.2. We compared hypergraph construction to a double-pushout with a production span $L \leftarrow K \rightarrow R$ where K is discrete. While a single reduction step can always be simulated by production spans with a discrete K, this is not the case if a truly concurrent semantics is considered. In this paper, we have only treated interleaving semantics, which is common in many process calculi.

Another approach concerning the inductive representation of graphs was introduced in Gadducci and Heckel (1997), with a different categorical model, and where hypergraphs are the arrows of a category (a so-called dgs-monoidal category).

In this paper we have re-established the fact that the category of graphs can be generated by finite colimits, starting with a single node and a single edge (Heckel 1998). In our case this colimit takes on a specific, and, therefore, unique form, where first the discrete hypergraphs are generated by coproducts, and then the graph construction operation is applied as a final step.

In graph rewriting there already exists a concept of typed graph grammars (Corradini *et al.* 1996), related to the type systems introduced in Sections 5 and 6.2, but nevertheless different. In the case of typed graph grammars, a type graph is fixed *a priori* and there is only one type graph for every set of productions. Graphs are considered valid only if they can be mapped into the type graph by a graph morphism (this is similar to our proposal). In our case, different graphs may have different type graphs, which is a good thing since they also might have different properties. So we compute the type graphs *a posteriori*, and it is a crucial point in the design of every type system to distinguish as many graphs as possible by assigning different type graphs to them.

Future work will consist of designing further techniques for the analysis of concurrent systems with a semantics based on graph transformation. First, a closer comparison to existing type systems for process calculi is planned, especially type systems for the π-calculus contain interesting concepts for typing processes (such as co- and contravariance in subtyping) which could be transferred into the graph-based setting. Second, it would

be interesting to generalise the ideas concerning typing and static analysis presented in this paper to arbitrary graph rewriting systems.

Acknowledgements

I would like to thank my colleagues – especially Astrid Kiehn for proofreading – and my former advisor Jürgen Eickel. Many thanks are also due to Reiko Heckel and Wolfram Kahl for their helpful suggestions. Finally, I would like to express my gratitude to the anonymous referees for their valuable comments.

References

Bauderon, M. and Courcelle, B. (1987) Graph expressions and graph rewritings. *Mathematical Systems Theory* **20** 83–127.

Corradini, A., Montanari, U. and Rossi, F. (1996) Graph processes. *Fundamenta Informaticae* **26** (3/4) 241–265.

Crole, R. L. (1993) *Categories for Types*, Cambridge University Press.

Ehrig, H. (1979) Introduction to the algebraic theory of graphs. In: Proc. 1st International Workshop on Graph Grammars. *Springer-Verlag Lecture Notes in Computer Science* **73** 1–69.

Ehrig, H., Pfender, M. and Schneider, H. (1973) Graph grammars: An algebraic approach. In: *Proc. 14th IEEE Symp. on Switching and Automata Theory* 167–180.

Engelfriet, J. and Vereijken, J. J. (1997) Context-free graph grammars and concatenation of graphs. *Acta Informatica* **34** 773–803.

Gadducci, F. and Heckel, R. (1997) An inductive view of graph transformation. In: Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT '97. *Springer-Verlag Lecture Notes in Computer Science* **1376** 223–237.

Gardner, P. (1998) Closed action calculi. *Theoretical Computer Science* (in association with the conference on Mathematical Foundations in Programming Semantics).

Habel, A. (1992) Hyperedge Replacement: Grammars and Languages. *Springer-Verlag Lecture Notes in Computer Science* **643**.

Heckel, R. (1998) *Open Graph Transformation Systems*, Ph. D. thesis, Technische Universität Berlin.

König, B. (1999) Generating type systems for process graphs. In: Proc. of CONCUR '99. *Springer-Verlag Lecture Notes in Computer Science* **1664** 352–367.

König, B. (2000) A graph rewriting semantics for the polyadic pi-calculus. In: *Workshop on Graph Transformation and Visual Modeling Techniques (Geneva, Switzerland), ICALP Workshops '00*, Carleton Scientific 451–458.

Kreowski, H.-J. (1980) A comparison between Petri-nets and graph grammars. In: Noltemeier, H. (ed.) Graphtheoretic Concepts in Computer Science. *Springer-Verlag Lecture Notes in Computer Science* **100** 306–317.

Löwe, M., Korff, M. and Wagner, A. (1993) An algebraic framework for the transformation of attributed graphs. In: Sleep, M., Plasmeijer, M. and van Eekelen, M. (eds.) *Term Graph Rewriting*, Wiley, Chapter 14 185–199.

Mac Lane, S. (1971) *Categories for the Working Mathematician*, Springer-Verlag.

Meseguer, J. and Montanari, U. (1990) Petri nets are monoids. *Information and Computation* **88** (2) 105–155.

Milner, R. (1993) The polyadic π-calculus: a tutorial. In: Hamer, F. L., Brauer, W. and Schwichtenberg, H. (eds.) *Logic and Algebra of Specification*, Springer-Verlag.

Pierce, B. and Sangiorgi, D. (1993) Typing and subtyping for mobile processes. In: *Proc. of LICS '93* 376–385.

Turner, D. (1995) *The Polymorphic Pi-Calculus: Theory and Implementation*, Ph. D. thesis, University of Edinburgh. (ECS-LFCS-96-345.)

Yoshida, N. (1994) Graph notation for concurrent combinators. In: Proc. of TPPP '94. *Springer-Verlag Lecture Notes in Computer Science* **907**.