Hypergraph Construction and Its Application to the Compositional Modelling of Concurrency

Barbara König (koenigb@in.tum.de)

Fakultät für Informatik, Technische Universität München

Abstract. We define a construction operation on hypergraphs using a colimit and show that its expressiveness concerning graph rewriting is equal to the graph expressions of Courcelle and the double-pushout approach of Ehrig. With an inductive way of representing graphs, graph rewriting arises naturally as a derived concept. The usefulness of our approach for the compositional modelling of concurrent systems is then shown by defining the semantics of a process calculus with mobility and of Petri nets.

1 Introduction

Graph rewriting is one adequate approach for modelling the semantics of concurrent systems: multi-dimensional structures describing interconnected computers or other components can be naturally described by graphs. When trying to model semantic frameworks for concurrency (such as process algebras) with graph rewriting, we run into problems since it is hard to represent compositionality and modularity inherent in these formalisms in the world of graphs. Process algebras rely on compositionality in the defi nition of their syntax and their semantics, and in almost all proofs. Compositionality is easy to achieve in a "string-based" syntax: systems are constructed out of smaller systems by concatenating their descriptions, 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", 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. Such a form of graph concatenation was proposed in [1] in the form of "graph expressions", where three operators (disjoint sum, node fusion, redefi nition of external nodes) were introduced. These operators can be used in order to gradually construct hypergraphs out of smaller hypergraphs.

We propose a similar approach of graph construction where the information on how to concatenate graphs is not provided by operators, but rather by a part of a colimit. The construction itself consists of completing the colimit and is related to the double-pushout approach [3]. The expressive power of graph rewriting in our approach is the same as for graph expressions [1] (and thus the same as in the double-pushout approach [7]). Compared to [1] we have additional information in the uniqueness property of the colimit and embeddings of the subgraphs into the constructed graph which are provided by the colimit. This information can be helpful when we want to reason about concurrent systems or if we want to extend the formalism. We still have a description of the graph in terms of edge and node sets, which makes it easy to add additional labels or annotations.

We demonstrate the usefulness of our approach by giving a compositional semantics of a process calculus with mobility and of Petri nets. Note that, although we model concurrent systems, the semantics itself will not be concurrent in the sense of true concurrency (as in [5]).

2 Hypergraphs and Hypergraph Construction

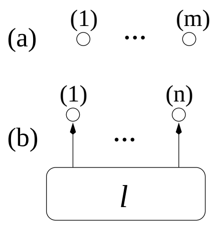
We first define some basic notions concerning hypergraphs (see also [7]).

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 *disjoint from V_H, $s_H\colon E_H \to V_H^*$ maps each edge to a string of* source nodes*, $l_H\colon E_H \to L$ 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 *$\phi\colon H \to H'$ consists of two mappings $\phi_E\colon E_H \to E_{H'}$, $\phi_V\colon V_H \to V_{H'}$ satisfying*[1] *$\phi_V(s_H(e)) = s_{H'}(\phi_E(e))$ 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 (= isomorphism) from one hypergraph into the other.*

The arity of a hypergraph H is defined as $ar(H) = |\chi_H|$ while the arity of an edge e of H is $ar(e) = |s_H(e)|$. We can regard hypergraphs and (strong) hypergraph morphisms as objects respectively morphisms of a category.

Notation: We call a hypergraph *discrete*, if its edge set is empty. $\mathbf{m}$ denotes a discrete graph of arity $m \in \mathbb{N}$ with m nodes where every node is external (see (a), external nodes are labelled (1), (2), … in their respective order).
$H = [l]_n$ is the hypergraph with exactly one edge e with label l where $s_H(e) = \chi_H$, $|\chi_H| = n$, χ_H contains no duplicates and $V_H = Set(\chi_H)$, where $Set(s)$ is the set of all elements of a string s (see (b), nodes 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 mapping discrete graphs to discrete graphs. The following definitions use concepts from category theory, namely categories and colimits. For an introduction to these concepts see [3, 2].

Definition 2. (Hypergraph Construction) *Let $H_1, \dots, H_n$ be hypergraphs and let*[2] *$\zeta_i : \mathbf{m_i} \to D$, $i \in [n]$ 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 $i \in [n]$.*

[1] Application of morphisms to sequences of nodes is conducted pointwise.
[2] $[n]$ stands for the set $\{1, \dots, n\}$.

$$\begin{tikzcd} \mathbf{m_i} \arrow[r, "\phi_i"] \arrow[d, "\zeta_i"'] & H_i \arrow[d, "\eta_i"] \\ D \arrow[r, "\phi"'] & H \end{tikzcd}$$

Let H (with morphisms $\phi : D \to H$, $\eta_i : H_i \to H$) be the colimit of $\zeta_1, \ldots, \zeta_n, \phi_1, \ldots, \phi_n$ such that ϕ is a strong morphism. 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 $\bigotimes_{i=1}^{n}(H_i, \zeta_i)$.)

Generally, colimits do not necessarily exist, but they always exist in our case. The colimit is unique up to bijective morphisms, but *not* unique up to isomorphism. Therefore we demand above that the morphism ϕ from D into the colimit be a strong morphism and thereby determine the string of external nodes of the result.

3 Graph Expressions and the Double-Pushout Approach

3.1 Graph Expressions

Graph expressions were introduced by Michel Bauderon and Bruno Courcelle in [1] 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: $H_1 \oplus H_2$ 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 $H_1 \oplus H_2$. $H_1 \oplus H_2 \cong \bigotimes_{i=1}^{2}(H_i, \zeta_i)$ where ζ_1, ζ_2 are defined as in figure 1 (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 redefining the external nodes of H according to α, i.e. χ_H is replaced by[3] $\lfloor \chi_H \rfloor_{\alpha(1)} \ldots \lfloor \chi_H \rfloor_{\alpha(p)}$. The rest of the hypergraph stays unchanged.

We exploit the fact that α can always be decomposed into $\alpha = \alpha_1 \circ \ldots \circ \alpha_k$ where each α_i has one of the following three forms listed below. According to [1] $\sigma_\alpha(H) \cong \sigma_{\alpha_k}(\ldots \sigma_{\alpha_1}(H) \ldots)$. Thus we only have to consider these three cases. In each of the following cases $\sigma_\alpha(H) \cong \otimes(H, \zeta)$.

Permutation of External Nodes: $\alpha : [m] \to [m]$ is a bijection. ζ is defined in (b).

Hiding an External Node: $\alpha : [m] \to [m+1]$ where $\alpha(i) = i$. ζ is defined in (c).

Duplicating an External Node: $\alpha : [m+1] \to [m]$ where $\alpha(i) = i$ if $i \in [m]$ and $\alpha(m+1) = m$. Then ζ is defined as in figure 1 (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 which are related by δ. The arity of the hypergraph is not changed.

According to [1] $\theta_\delta(H) \cong \theta_{\delta_k}(\ldots \theta_{\delta_1}(H) \ldots)$ where each δ_i is an equivalence generated by a single pair (i, j) with $i, j \in [m]$. With the permutation operation

[3] $\lfloor s \rfloor_i$ denotes the i-th element of the string s.

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 1 (e).

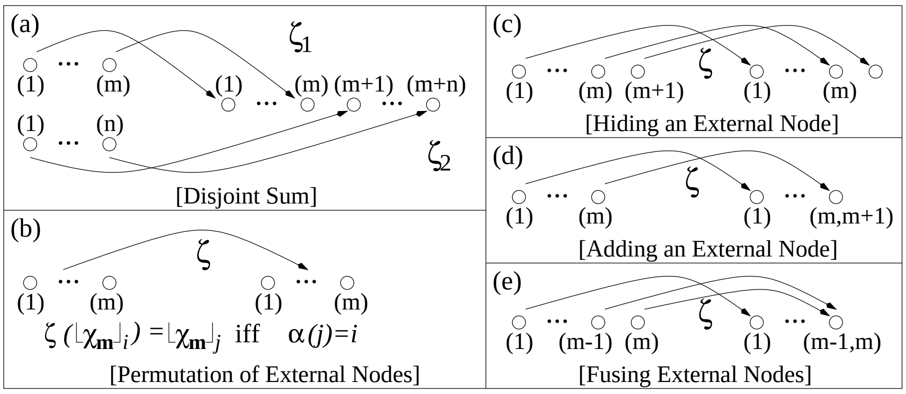


Fig. 1. Converting a graph expression into the corresponding colimit construction.

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.

3.2 The Double-Pushout Approach

The double-pushout approach to graph rewriting was introduced by Ehrig [3], while the double-pushout approach to *hypergraph* rewriting is presented in [7]. Our colimit construction is closely related to this approach. We will now make this connection precise. If, in the diagram in definition 2, we set $n = 1$ and allow D to be an arbitrary non-discrete graph, we obtain exactly the right-hand side of a double-pushout.

This does not yet reveal anything about the expressive power of our approach. We will now define the notion of a rewriting step: 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 which is generated by the following two rules and is closed under isomorphism.

$$L \overset{r}{\Longrightarrow} R \qquad \frac{H_1 \overset{r}{\Longrightarrow} H_1'}{(H_1, \zeta_1) \otimes (H_2, \zeta_2) \overset{r}{\Longrightarrow} (H_1', \zeta_1) \otimes (H_2, \zeta_2)}$$

Proposition 1. *Let G, H be hypergraphs and let $r = (L, R)$ be a rewriting rule. Then $G \overset{r}{\Longrightarrow} H$ if and only if G can be transformed into H 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 was already shown in [1] 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). This proposition can of course also be shown in a more direct way. $\square$

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 construction.

Proposition 2. *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 the figure below. Then it holds for arbitrary hypergraphs H_{ij} with $m_{ij} = ar(H_{ij})$ that*

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

Proof.

The proof of the proposition is shown in the figure. While the left-hand side of equation (1) is formed by applying first colimits (b) to the H_{ij} followed by an application of colimit (a)+(c), the same goal can be achieved by forming colimit (a) first and applying colimits (b)+(c) afterwards. This argumentation is valid since the combination of two colimits always yields another colimit.

$$\begin{tikzcd}
 & \mathbf{m}_{ij} \arrow[r] \arrow[d, "\zeta_{ij}"'] \arrow[dr, phantom, "(b)"] & H_{ij} \arrow[d] \\
\mathbf{m}_i \arrow[r, "\phi_i"] \arrow[d, "\zeta_i"] \arrow[dr, phantom, "(a)"] & D_i \arrow[r] \arrow[d, "\xi_i"'] \arrow[dr, phantom, "(c)"] & H_i \arrow[d] \\
D \arrow[r] \arrow[rr, bend right] & D' \arrow[r, dashed] & H
\end{tikzcd}$$

□

Hyperedges are the basic units of graph construction. Just as every element of a vector space can be decomposed into base vectors in a unique way, there is a unique decomposition of every hypergraph into hyperedges.

Proposition 3. (Unique Factorization) *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 factorization is unique up to isomorphism and index permutation.*

As in vector spaces we can define linear mappings on hypergraphs.

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

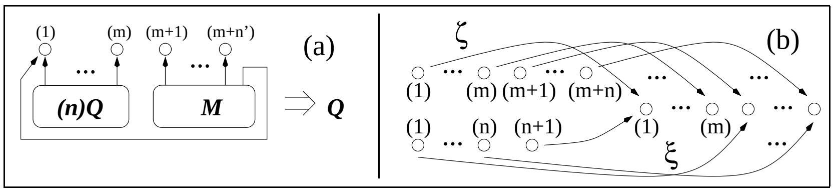
Proposition 4. (Unique Linear Mapping) *For each mapping of hyperedges $[l]_m$ to hypergraphs of arity m, there is exactly one linear mapping (up to isomorphism) which is an extension of the original mapping.*

5 Typed Process Graphs

We show how to model a process calculus, closely related to the asynchronous polyadic π-calculus [13], by so-called process graphs. There is an encoding from the π-calculus into process graphs [9, 8]. On the other hand there is a straightforward encoding of process graphs into closed action calculi [6] and a close relation of our process graphs to the ones in [15]. A process graph is defined inductively in the following way.

Definition 4. *(***Process Graphs***) 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 $1 \leq n \leq ar(Q)$ (e is a process waiting for a message with n ports arriving at its first node) or is labelled with the constant M (e is a message sent to its last node). The reduction relation (reception of a message and its nodes by a process) is generated by the rewrite rule in figure (a) and is closed under graph construction.*

For simplicity we have omitted a construct and a corresponding rewrite rule describing replication. However, it introduces no additional complication, even for the type check introduced below.



The rewrite rule in (a) is not always defined, it may fail if $ar(Q) \neq m + n'$. Furthermore we want to avoid that $n \neq n'$, that is we want to ensure that the expected number of nodes is received. We use morphisms, graph construction and a linear mapping in order to define a condition which is sufficient for avoiding this kind of runtime errors and which can be checked statically.

Proposition 5. *Let L be a linear mapping which is defined on the hyperedges as follows: $L([M]_n) = [t]_n$ (t is a new edge label) and $L([(n)Q]_m) = (L(Q), \zeta) \otimes ([t]_{n+1}, \xi)$ if $n + m = ar(Q)$ (undefined otherwise). ζ, ξ are defined in figure (b).*

Let P be a process graph. If there exists a strong morphism $\phi : L(P) \twoheadrightarrow H$ into a hypergraph H which satisfies

$$e_1, e_2 \in E_H,\ \lfloor s_H(e_1) \rfloor_{ar(e_1)} = \lfloor s_H(e_2) \rfloor_{ar(e_2)} \ \Rightarrow\ e_1 = e_2 \tag{2}$$

(i.e. all messages that share the last node are already the same) then P will never encounter a runtime error during reduction.

L extracts pure communication structure from a process graph, i.e. 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 H. It thus guarantees absence of undefined rewrites for the entire reduction.

H can be regarded as a type of P and we can easily unfold H into well-known type trees of π-calculus processes [14]. The method presented above corresponds to a type system for the π-calculus with recursive types and simple polymorphism. The type of a process can also be specified by inductive typing rules rather than by a linear mapping.

More expressive (generic) type systems for the analysis of processes, which can be nicely integrated into the graph-based setting, can be found in [9, 8].

6 A Compositional Semantics for Petri Nets

We will now show another application of graph construction by giving a compositional semantics for Petri nets. A Petri net can easily be represented by a hypergraph H (compare with [11]): nodes are places and edges are transitions. 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 $l : E \to \mathbb{N} \times \mathbb{N}$. If $l(e) = (s,t)$ then $s + t = \mathit{ar}(e)$, s is the number of sources (the first s nodes) and t (the last t nodes) is the number of targets. We also need an additional labelling $z : V_H \to \mathit{Mon}$ mapping each node to an element of a cancellative commutative monoid, in order to represent the tokens present at each node. In our example we will set $\mathit{Mon} = \mathbb{N}$, but we could also represent high-level Petri nets by assuming that *Mon* is the set of all multi-sets over certain elements.

A Petri net is now a pair $[H,z]$ where H is a hypergraph with labels taken from $\mathbb{N} \times \mathbb{N}$ and $z : V_H \to \mathit{Mon}$ is a mapping. Before we can define the semantics we first have to extend our notion of hypergraph construction to hypergraphs with tokens. But this is easy since our construction operation yields morphisms of the subgraphs into the constructed graph. Let $[H_i, z_i]$ be Petri nets. We define:

$$\bigotimes_{i=1}^{n}([H_i,z_i],\zeta_i) \cong [\bigotimes_{i=1}^{n}(H_i,\zeta_i), z] \text{ where } z(v) = \sum_{i=1}^{n} \sum_{\eta_i(w)=v} z_i(w)$$

The η_i are the morphisms of H_i into $\bigotimes_{i=1}^{n}(H_i,\zeta_i)$ yielded by the colimit. $\sum$ is the commutative operation of *Mon*.

Inductive Definition of Petri Nets: a Petri net N is either of the form $[[s,t]_{s+t}, z]$ where $z : V_{[s,t]_{s+t}} \to \mathit{Mon}$ 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: we now assume that $\mathit{Mon} = \mathbb{N}$. A single transition fires if all its source nodes are labelled with tokens. And if one transition fires, the entire net is changed accordingly.

$$[T,z] \Longrightarrow [T,z'] \qquad \frac{N_1 \Longrightarrow N_1'}{(N_1,\zeta_1) \otimes (N_2,\zeta_2) \Longrightarrow (N_1',\zeta_1) \otimes (N_2,\zeta_2)}$$

where $T = [s,t]_{s+t}$ is a transition, $z, z' : V_T \to \{0,1\}$ and $z(\lfloor \chi_T \rfloor_i) = 1 \iff i \in [s] \iff z'(\lfloor \chi_T \rfloor_i) = 0$.

This approach works also when a place is allowed to hold more than one token. In this case the extra tokens which are not needed to fire a transition are not attached to the transition (or edge) itself but to the external nodes of the surrounding hypergraphs.

7 Conclusion

We have presented a method of hypergraph construction which allows us to build hypergraphs out of smaller ones. The basic units are single edges. This method can be used to give an operational semantics to concurrent systems whose states can often be represented by hypergraphs in a natural way. Another approach concerning inductive graph representation was given in [4] with a different categorical representation of graphs (in this paper graphs are the *morphisms* of a category) and different operators on graphs.

Future work will consist in designing further techniques for the analysis of concurrent systems with a hypergraph-based semantics. One promising direction is to continue the work on generic type systems for process graphs [9] and to extend it to more general graph rewrite systems. Furthermore we plan to investigate the connection between our approach to model Petri nets and existing approaches as in [12, 11].

Acknowledgements: I would like to thank my colleagues and my former advisor Jürgen Eickel. Furthermore I want to express my gratitude to the anonymous referees for their valuable comments.

References

1. Michel Bauderon and Bruno Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20:83–127, 1987.
2. Roy L. Crole. *Categories for Types*. Cambridge University Press, 1993.
3. H. Ehrig. Introduction to the algebraic theory of graphs. In *Proc. 1st International Workshop on Graph Grammars*, pages 1–69. Springer-Verlag, 1979. LNCS 73.
4. F. Gadducci and R. Heckel. An inductive view of graph transformation. In *Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT '97*, volume 1376, pages 223–237, 1997. LNCS 1376.
5. F. Gadducci, R. Heckel, and M. Llabres. A bicategorical axiomatization of concurrent graph rewriting. In *Proceedings of CTCS'99*, 1999. ENTCS 29.
6. Philippa Gardner. Closed action calculi. *Theoretical Computer Science (in association with the conference on Mathematical Foundations in Programming Semantics)*, 1998.
7. Annegret Habel. *Hyperedge Replacement: Grammars and Languages*. Springer-Verlag, 1992. LNCS 643.
8. Barbara König. *Description and Verification of Mobile Processes with Graph Rewriting Techniques*. PhD thesis, Technische Universität München, 1999.
9. Barbara König. Generating type systems for process graphs. In *Proc. of CONCUR '99*, pages 352–367. Springer-Verlag, 1999. LNCS 1664.
10. Barbara König. Hypergraph construction and its application to the compositional modelling of concurrency (full version). Technical Report TUM-I0003, Technische Universität München, 2000.
11. H.-J. Kreowski. A comparison between petri-nets and graph grammars. In H. Noltemeier, editor, *Graphtheoretic Concepts in Computer Science*, pages 306–317. Springer-Verlag, 1980. LNCS 100.
12. José Meseguer and Ugo Montanari. Petri nets are monoids. *Information and Computation*, 88(2):105–155, 1990.
13. Robin Milner. The polyadic π-calculus: a tutorial. In F. L. Hamer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*. Springer-Verlag, Heidelberg, 1993.
14. David Turner. *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis, University of Edinburgh, 1995. ECS-LFCS-96-345.
15. Nobuko Yoshida. Graph notation for concurrent combinators. In *Proc. of TPPP '94*. Springer-Verlag, 1994. LNCS 907.