TECHNISCHE UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen

A General Framework for Types in Graph Rewriting

Barbara König

TUM-I0014
SFB-Bericht Nr. 342/10/00 A
September 00

TUM–INFO–09-I0014-0/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

A General Framework for Types in Graph Rewriting*

Barbara König (koenigb@in.tum.de)

Fakultät für Informatik, Technische Universität München

Abstract. A general framework for typing graph rewriting systems is presented: the idea is to statically derive a type graph from a given graph. In contrast to the original graph, the type graph is invariant under reduction, but still contains meaningful behaviour information. We present conditions, a type system for graph rewriting should satisfy, and a methodology for proving these conditions. In three case studies it is shown how to incorporate existing type systems (for the polyadic π-calculus and for a concurrent object-oriented calculus) and a new type system into the general framework.

1 Introduction

In the past, many formalisms for the specification of concurrent and distributed systems have emerged. Some of them are aimed at providing an encompassing theory: a very general framework in which to describe and reason about interconnected processes. Examples are action calculi [18], rewriting logic [16] and graph rewriting [3] (for a comparison see [4]). They all contain a method of building terms (or graphs) from basic elements and a method of deriving reduction rules describing the dynamic behaviour of these terms in an operational way.

A general theory is useful, if concepts appearing in instances of a theory can be generalised, yielding guidelines and relieving us of the burden to prove universal concepts for every single special case. An example for such a generalisation is the work presented for action calculi in [15] where a method for deriving a labelled transition semantics from a set of reaction rules is presented. We concentrate on graph rewriting (more specifically hypergraph rewriting) and attempt to generalise the concept of type systems, where, in this context, a type may be a rather complex structure.

Compared to action calculi[1] and rewriting logic, graph rewriting differs in a significant way in that connections between components are described explicitly (by connecting them by edges) rather than implicitly (by referring to the same channel name). We claim that this feature—together with the fact that it is easy to add an additional layer containing annotations and constraints to a graph—can simplify the design of a type system and therefore the static analysis of a graph rewriting system.

After introducing our model of graph rewriting and a method for annotating graphs, we will present a general framework for type systems where both—the expression to be typed and the type itself—are hypergraphs and will show how to reduce the proof obligations for instantiations of the framework. We are interested in the following properties: correctness of a type system (if an expression has a certain type, then we can conclude that this expression has certain properties), the subject reduction property (types are invariant under reduction) and compositionality (the type of an expression can always be derived from the types of its subexpressions). Parts of the proofs of these properties can already be conducted for the general case.

We will then show that our framework is realistic by instantiating it to two well-known type systems: a type system avoiding run-time errors in the polyadic π-calculus [17] and a type system avoiding "message not understood"-errors in a concurrent object-oriented setting. As a final example we model reception and execution of an untrustworthy applet and check that no trustworthy data is ever modified by the applet.

Note that we do not present a method for automatically deriving a type system from a given set of rewrite rules. In this paper we are rather interested in fixing the minimal properties a type

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

[1] Here we mean action calculi in their standard string notation. There is also a graph notation for action calculi, see e.g. [7].

system should satisfy and in the development of a proof methodology which simplifies the task of showing that these properties are indeed met.

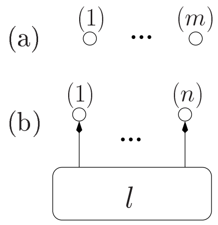
2 Hypergraph Rewriting and Hypergraph Annotation

We first define some basic notions concerning hypergraphs (see also [6]) and a method for inductively constructing hypergraphs.

Definition 1. (Hypergraph)
Let L be a fixed set of labels. A hypergraph $H = (V_H, E_H, s_H, l_H, \chi_H)$ consists of a set of nodes V_H, a set of edges E_H, a connection mapping $s_H : E_H \to V_H^*$, an edge labelling $l_H : E_H \to L$ and a string $\chi_H \in V_H^*$ of external nodes. A hypergraph morphism $\phi : H \to H'$ (consisting of $\phi_V : V_H \to V_{H'}$ and $\phi_E : E_H \to E_{H'}$) maps nodes to nodes and edges to edges, preserving connections and labelling, i.e.[2] *$\phi_V(s_H(e)) = s_{H'}(\phi_E(e))$ and $l_H(e) = l_{H'}(\phi_E(e))$. A strong morphism (denoted by the arrow $\twoheadrightarrow$) additionally preserves the external nodes, i.e. $\phi_V(\chi_H) = \chi_{H'}$. We write $H \cong H'$ (H is isomorphic to H') if there is a bijective strong morphism from H to H'.*

The arity of a hypergraph H is defined as $ar(H) = |\chi_H|$ while the arity of an edge e of H is $ar(e) = |s_H(e)|$. External nodes are the interface of a hypergraph towards its environment and are used to attach hypergraphs.

Notation: We call a hypergraph *discrete*, if its edge set is empty. By $\mathbf{m}$ we denote a discrete graph of arity $m \in \mathbb{N}$ with m nodes where every node is external (see Figure (a) to the right, external nodes are labelled (1), (2), ... in their respective order).
The hypergraph $H = [l]_n$ contains exactly one edge e with label l where $s_H(e) = \chi_H$, $ar(e) = n$ and[3] $V_H = Set(\chi_H)$ (see (b), nodes are ordered from left to right).



The next step is to define a method (first introduced in [10]) for the annotation of hypergraphs with lattice elements and to describe how these annotations change under morphisms. We use annotated hypergraphs as types where the annotations can be considered as extra typing information, therefore we use the terms *annotated hypergraph* and *type graph* as synonyms.

Definition 2. (Annotated Hypergraphs) *Let $\mathcal{A}$ be a mapping assigning a lattice $\mathcal{A}(H) = (I, \leq)$ to every hypergraph and a function $\mathcal{A}_\phi : \mathcal{A}(H) \to \mathcal{A}(H')$ to every morphism $\phi : H \to H'$. We assume that $\mathcal{A}$ satisfies:*

$$\mathcal{A}_\phi \circ \mathcal{A}_\psi = \mathcal{A}_{\phi\circ\psi} \qquad \mathcal{A}_{id_H} = id_{\mathcal{A}(H)} \qquad \mathcal{A}_\phi(a \vee b) = \mathcal{A}_\phi(a) \vee \mathcal{A}_\phi(b) \qquad \mathcal{A}_\phi(\bot) = \bot$$

where $\vee$ is the join-operation, a and b are two elements of the lattice $\mathcal{A}(H)$ and $\bot$ is its bottom element.

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

Example: We consider the following annotation mapping $\mathcal{A}$: let $(\{\mathit{false}, \mathit{true}\}, \leq)$ be the boolean lattice where $\mathit{false} < \mathit{true}$. We define $\mathcal{A}(H)$ to be the set of all mappings from V_H into $\{\mathit{false}, \mathit{true}\}$ (which yields a lattice with pointwise order). By choosing an element of $\mathcal{A}(H)$ we fix a subset of the nodes. So let $a : V_H \to \{\mathit{false}, \mathit{true}\}$ be an element of $\mathcal{A}(H)$ and let $\phi : H \to H'$, $v' \in V_H$. We define: $\mathcal{A}_\phi(a) = a'$ where $a'(v') = \bigvee_{\phi(v)=v'} a(v)$. That is, if a node v with annotation *true* is mapped to a node v' by ϕ, the annotation of v' will also be *true*.

[2] The application of ϕ_V to a string of nodes is defined pointwise.
[3] $Set(\tilde{s})$ is the set of all elements of a string $\tilde{s}$

From the point of view of category theory, $\mathcal{A}$ is a functor from the category of hypergraphs and hypergraph morphisms into the category of lattices and join-morphisms (i.e. functions preserving the join operation of the lattice).

We now introduce a method for attaching (annotated) hypergraphs with a construction plan consisting of discrete graph morphisms.

Definition 3. (Hypergraph Construction) *Let $H_1[a_1], \ldots, H_n[a_n]$ be annotated hypergraphs and let $\zeta_i : \mathbf{m_i} \to D$, $1 \le i \le n$ be hypergraph morphisms where $ar(H_i) = m_i$ and D is discrete. Furthermore let $\phi_i : \mathbf{m_i} \twoheadrightarrow H_i$ be the unique strong morphisms.*

For this construction we assume 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 their nodes satisfying $\zeta_i(v) \approx \phi_i(v)$ if $1 \le i \le n$, $v \in V_{\mathbf{m_i}}$. The nodes of the constructed graph are the equivalence classes of $\approx$. We define

$$\textcircled{\scriptsize D}_{i=1}^{n}(H_i, \zeta_i) = ((V_D \cup \bigcup_{i=1}^{n} V_{H_i})/\approx, \bigcup_{i=1}^{n} E_{H_i}, s_H, l_H, \chi_H)$$

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 $e \in E_{H_i}$. And we define $\chi_H = [v_1]_\approx \ldots [v_k]_\approx$ if $\chi_D = v_1 \ldots v_k$.

If $n = 0$, the result of the construction is D itself.

We construct embeddings $\phi : D \twoheadrightarrow H$ and $\eta_i : H_i \to H$ by mapping every node to its equivalence class and every edge to itself. Then the construction of annotated graphs can be defined as follows:

$$\textcircled{\scriptsize D}_{i=1}^{n}(H_i[a_i], \zeta_i) = \left(\textcircled{\scriptsize D}_{i=1}^{n}(H_i, \zeta_i)\right)\left[\bigvee_{i=1}^{n} \mathcal{A}_{\eta_i}(a_i)\right]$$

In other words: we join all graphs $D, H_1, \ldots, H_n$ and fuse exactly the nodes which are the image of one and the same node in the $\mathbf{m_i}$, χ_D becomes the new sequence of external nodes. Lattice annotations are joined if the annotated nodes are merged. In terms of category theory, $\textcircled{\scriptsize D}_{i=1}^{n}(H_i[a_i], \zeta_i)$ is the colimit of the ζ_i and the ϕ_i regarded as $\mathcal{A}$-morphisms (D and the $\mathbf{m_i}$ are annotated with the bottom element $\perp$). The properties of the annotation mapping, given in Definition 2, are needed to show that $\textcircled{\scriptsize D}_{i=1}^{n}(H_i[a_i], \zeta_i)$ is in fact a colimit.

Proposition 1. *Let $H[a_1], \ldots, H_n[a_n]$ be annotated hypergraphs with with $m_i = ar(H_i)$, let $\zeta_i : \mathbf{m_i}[\perp] \to_{\mathcal{A}} D[\perp]$ be discrete morphisms and let $\phi_i : \mathbf{m_i}[\perp] \twoheadrightarrow_{\mathcal{A}} H_i[a_i]$ be the unique strong morphisms.*

Then $H[a] = \textcircled{\scriptsize D}_{i=1}^{n}(H_i[a_i], \zeta_i)$ (with morphisms η_i, ϕ of Definition 3) is the colimit of the ζ_i and the ϕ_i in the category of annotated hypergraphs and $\mathcal{A}$-morphisms.

Proof. We first have to show that $\eta_i \circ \phi_i = \phi \circ \zeta_i$ holds: all $v \in V_{\mathbf{m_i}}$ satisfy $\phi_i(v) \approx \zeta_i(v)$ by definition by $\approx$ is the equivalence defined in Definition 3. Therefore $\eta_i(\phi_i(v)) = [\phi_i(v)]_\approx = [\zeta_i(v)]_\approx = \phi(\zeta_i(v))$.

Now we assume that there is another annotated hypergraph $H'[a']$ and $\mathcal{A}$-morphisms $\eta_i' : H_i[a_i] \to_{\mathcal{A}} H'[a']$ and $\phi' : D[\perp] \twoheadrightarrow_{\mathcal{A}} H'[a']$ such that $\eta_i' \circ \phi_i = \phi' \circ \zeta_i$. In order to show that $H[a]$ is a colimit we need to construct a unique $\mathcal{A}$-morphism $\psi : H[a] \twoheadrightarrow_{\mathcal{A}} H'[a']$ such that $\psi \circ \eta_i = \eta_i'$ and $\psi \circ \phi = \phi'$.

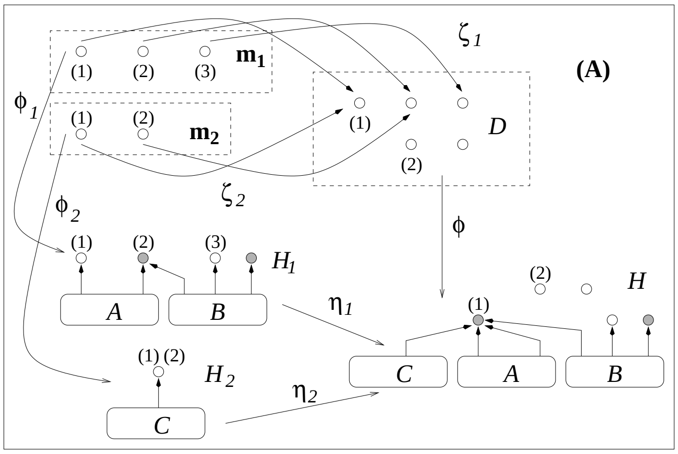
The nodes and edges of $H[a]$ are defined in Definition 3. The only possible definition of ψ which satisfies the conditions above is to set

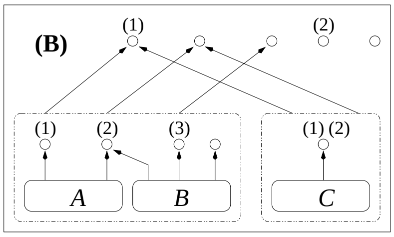
$$\psi(\eta_i(e)) = \eta_i'(e) \quad \text{and} \quad \psi([v]_\approx) = \begin{cases} \phi'(v) \text{ if } v \in V_D \\ \eta_i'(v) \text{ if } v \in V_{H_i} \end{cases}$$

It is straightforward to show that ψ is well-defined. It rests to prove that it is an $\mathcal{A}$-morphism:

$$\begin{aligned}\mathcal{A}_\psi(a) = \mathcal{A}_\psi(\bigvee_{i=1}^{n} \mathcal{A}_{\eta_i}(a_i)) = \bigvee_{i=1}^{n} \mathcal{A}_\psi(\mathcal{A}_{\eta_i}(a_i)) = \bigvee_{i=1}^{n} \mathcal{A}_{\psi\circ\eta_i}(a_i) \\ = \bigvee_{i=1}^{n} \mathcal{A}_{\eta_i'}(a_i) \le \bigvee_{i=1}^{n} a' = a'\end{aligned}$$

Example: we present a small example for graph construction, where we combine hypergraphs H_1, H_2 with the discrete morphisms $\zeta_1 : \mathbf{3} \to D$ and $\zeta_2 : \mathbf{2} \to D$ depicted in Figure (A) below (ignore the grey nodes for the moment). The resulting hypergraph is H.

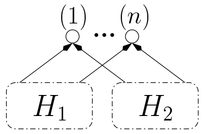




The following points are noteworthy:

- the first external node of $\mathbf{m_1}$ and the first external node of $\mathbf{m_2}$ are mapped to the same node in D, which means that the respective nodes of H_1 and H_2 are to be fused in H.
- the hypergraph H_2 has duplicates in the sequence of its external nodes. This causes all nodes that are to be fused with either the first or the second node of H_2 to be fused themselves, which happens to the two nodes attached to the A-edge.
- the discrete graph D contains an internal and an external node which are not in the range of the ζ_i. This indicates that they are still present in the resulting graph H, but not attached to any edge.

If we assume an annotation mapping as in the example above (mapping the node set to $\{\mathit{true}, \mathit{false}\}$) for H_1 and H_2 and shade all nodes that are labelled *true* with grey, then, in the annotation mapping for H, exactly the nodes that are the image of at least one grey node will be again grey.



We also use another, more intuitive notation for graph construction. Let $\zeta_i : \mathbf{m_i} \to D$, $1 \le i \le n$. Then we depict $\textcircled{D}_{i=1}^{n}(H_i, \zeta_i)$ by drawing the hypergraph $(V_D, \{e_1, \ldots, e_n\}, s_H, l_H, \chi_D)$ where $s_H(e_i) = \zeta_i(\chi_{\mathbf{m_i}})$ and $l_H(e_i) = H_i$.

Example: we can draw $\textcircled{n}_{i=1}^{2}(H_i, \zeta_i)$ where $\zeta_1, \zeta_2 : \mathbf{n} \twoheadrightarrow \mathbf{n}$ as in the picture above (note that the edges have dashed lines). Here we fuse the external nodes of H_1 and H_2 in their respective order and denote the resulting graph by $H_1 \Box H_2$. If there is an edge with a dashed line labelled with an edge $[l]_n$ we rather draw it with a solid line and label it with l (see e.g. the second figure in section 4.1).

In our example above, the new notation looks as drawn in Figure (B).

Definition 4. (Hypergraph Rewriting) *Let $\mathcal{R}$ be a set of pairs (L, R) (called rewriting rules), where the left-hand side L and the right-hand side R are both hypergraphs of the same arity. Then $\to_{\mathcal{R}}$ is the smallest relation generated by the pairs of $\mathcal{R}$ and closed under hypergraph construction.*

In our approach we generate the same transition system as in the double-pushout approach to graph rewriting described in [2] (for details see [13]).

We need one more concept: a linear mapping which is an inductively defined transformation, mapping hypergraphs to hypergraphs and adding annotation.

Definition 5. (Linear Mapping) *A function from hypergraphs to hypergraphs is called arity-preserving if it preserves arity and isomorphism classes of hypergraphs.*

Let t be an arity-preserving function that maps hypergraphs of the form $[l]_n$ to annotated hypergraphs. Then t can be extended to arbitrary hypergraphs by defining $t(\textcircled{D}_{i=1}^{n}([l_i]_{n_i}, \zeta_i)) = \textcircled{D}_{i=1}^{n}(t([l_i]_{n_i}), \zeta_i)$ and is then called a linear mapping.

A linear mapping satisfies $t(\textcircled{D}_{i=1}^{n}(H_i, \zeta_i)) \cong \textcircled{D}_{i=1}^{n}(t(H_i), \zeta_i)$ for arbitrary hypergraphs H_i. Note that the construction operator on the left-hand side of the equation works on ordinary hypergraphs, while the one on the right-hand side operates on annotated hypergraphs.

3 Static Analysis and Type Systems for Graph Rewriting

Having introduced all underlying notions we now specify the requirements for type systems. We assume that there is a fixed set $\mathcal{R}$ of rewrite rules, an annotation mapping $\mathcal{A}$, a predicate X on hypergraphs (representing the property we want to check), a property Y on type graphs and a relation $\triangleright$ with the following meaning: if $H \triangleright T$ where H is a hypergraph and T a type graph (annotated wrt. to $\mathcal{A}$), then H has type T. It is required that H and T have the same arity.

We demand that $\triangleright$ satisfies the following conditions: first, a type should contain information concerning the properties of a hypergraph, i.e. if a hypergraph has a type and Y holds for this type, then we can be sure that the property X holds.

$$H \triangleright T \,\wedge\, Y(T) \;\Rightarrow\; X(H) \quad \textbf{(correctness)} \tag{1}$$

Note that in the short version of this paper [11], we have omitted the predicate Y, since it is always true for the two examples presented there. But for some examples (see section 4.3) it is convenient to have possibility to perform an additional check on the type graph.

During reduction, the type stays invariant.

$$H \triangleright T \,\wedge\, H \to_{\mathcal{R}} H' \;\Rightarrow\; H' \triangleright T \quad \textbf{(subject reduction property)} \tag{2}$$

From (1) and (2) we can conclude that $H \triangleright T$, $Y(T)$ and $H \to_{\mathcal{R}}^{*} H'$ imply $X(H')$, that is X holds during the entire reduction.

The strong $\mathcal{A}$-morphisms introduced in Definition 2 impose a preorder on type graphs. It should always be possible to weaken the type with respect to that preorder.

$$H \triangleright T \,\wedge\, T \twoheadrightarrow_{\mathcal{A}} T' \;\Rightarrow\; H \triangleright T' \quad \textbf{(weakening)} \tag{3}$$

We also demand that the type system is compositional, i.e a graph has a type if and only if this type can be obtained by typing its subgraphs and combining these types. We can not sensibly demand that the type of an expression is obtained by combining the types of the subgraphs in exactly the same way the expression is constructed, so we introduce a partial arity-preserving mapping f doing some post-processing.

$$\begin{aligned} &\forall i\colon H_i \triangleright T_i \;\Rightarrow\; \textcircled{D}_{i=1}^{n}(H_i, \zeta_i) \triangleright f(\textcircled{D}_{i=1}^{n}(T_i, \zeta_i)) \qquad \textbf{(compositionality)} \\ &\textcircled{D}_{i=1}^{n}(H_i, \zeta_i) \triangleright T \;\Rightarrow\; \exists T_i\colon (H_i \triangleright T_i \text{ and } f(\textcircled{D}_{i=1}^{n}(T_i, \zeta_i)) \twoheadrightarrow_{\mathcal{A}} T) \end{aligned} \tag{4}$$

A last condition—the existence of minimal types—may not be strictly needed for type systems, but type systems satisfying this condition are much easier to handle.

$$H \text{ typable} \;\Rightarrow\; \exists T\colon (H \triangleright T \wedge (H \triangleright T' \iff T \twoheadrightarrow_{\mathcal{A}} T')) \quad \textbf{(minimal types)} \tag{5}$$

Let us now assume that types are computed from graphs in the following way: there is a linear mapping t, such that $H \triangleright f(t(H))$, if $f(t(H))$ is defined, and all other types of H are derived by the weakening rule, i.e. $f(t(H))$ is the minimal type of H.

The meaning of the mappings t and f can be explained as follows: t is a transformation *local* to edges, abstracting from irrelevant details and adding annotation information to a graph. The mapping f on the other hand, is a *global* operation, merging or removing parts of a graph in order to anticipate future reductions and thus ensure the subject reduction property. In the example in section 4.1 f "folds" a graph into itself, hence the letter f. In order to obtain compositionality, it is required that f can be applied arbitrarily often at any stage of type inference, without losing information (see condition (7) of Theorem 1).

This early restriction to a somewhat more specialised type system is partly motivated by the fact that it allows a classical rule-based formulation. By definition $H \triangleright T$ holds if and only if $f(t(H)) \twoheadrightarrow_{\mathcal{A}} T$. And this, in turn, holds if and only if $H \triangleright T$ can be derived with the following typing rules:

$$[l]_m \triangleright f(t([l]_m)) \qquad \frac{\forall i\colon H_i \triangleright T_i}{\textcircled{\scriptsize D}_{i=1}^{n}(H_i,\zeta_i) \triangleright f(\textcircled{\scriptsize D}_{i=1}^{n}(T_i,\zeta_i))} \qquad \frac{H \triangleright T,\ T \twoheadrightarrow_{\mathcal{A}} T'}{H \triangleright T'}$$

In this setting it is sufficient to prove some simpler conditions, especially the proof of (2) can be conducted locally.

Theorem 1. *Let $\mathcal{A}$ be a fixed annotation mapping, let f be an arity-preserving mapping as above, let t be a linear mapping, let X and Y be predicates on hypergraphs respectively type graphs and let $H \triangleright T$ if and only if $f(t(H)) \twoheadrightarrow_{\mathcal{A}} T$. Let us further assume that f and Y satisfy*[4]

$$T \twoheadrightarrow_{\mathcal{A}} T' \wedge Y(T') \Rightarrow Y(T) \tag{6}$$
$$f(\textcircled{\scriptsize D}_{i=1}^{n}(T_i,\zeta_i)) \cong f(\textcircled{\scriptsize D}_{i=1}^{n}(f(T_i),\zeta_i)) \tag{7}$$
$$T \twoheadrightarrow_{\mathcal{A}} T' \Rightarrow f(T) \twoheadrightarrow_{\mathcal{A}} f(T') \tag{8}$$

Then the relation $\triangleright$ satisfies conditions (1)–(5) if and only if it satisfies

$$Y(f(t(H))) \Rightarrow X(H) \tag{9}$$
$$(L,R) \in \mathcal{R} \Rightarrow f(t(R)) \twoheadrightarrow_{\mathcal{A}} f(t(L)) \tag{10}$$

Proof. Note that in the short version of this paper [11], we have omitted the predicate Y. This is equivalent to setting $Y(T) = \mathit{true}$ for every type graph T.

We first show that (9) and (10) imply (1)–(5)

(1) Let $H \triangleright T$ and $Y(T)$. From the definition of $\triangleright$ it follows that $f(t(H)) \twoheadrightarrow_{\mathcal{A}} T$ and (6) implies that $Y(f(t(H)))$ is satisfied. With (9) we conclude that $X(H)$ holds.

(2) Let $H \triangleright T$ and $H \rightarrow_{\mathcal{R}} H'$. From the definition of $\triangleright$ it follows that $f(t(H)) \twoheadrightarrow_{\mathcal{A}} T$.
The relation $\rightarrow_{\mathcal{R}}$ is defined via the closure of $\mathcal{R}$ under hypergraph construction, i.e. $H \cong \textcircled{\scriptsize D}_{i=1}^{n}(H_i,\zeta_i)$, $H' \cong \textcircled{\scriptsize D}_{i=1}^{n}(H'_i,\zeta_i)$ and there is a rule $(L,R) \in \mathcal{R}$ such that $H_j \cong L$ and $H'_j \cong R$. For all other i with $i \neq j$ it holds that $H_i \cong H'_i$.
Since f and t preserve isomorphism classes, it holds that $f(t(H_i)) \cong f(t(H'_i))$ and with (10) it follows that $f(t(R)) \twoheadrightarrow_{\mathcal{A}} f(t(L))$.
Since $\mathcal{A}$-morphisms are preserved by graph construction (this can easily be shown via the characterisation of graph construction as a colimit) it follows that

$$\textcircled{\scriptsize D}_{i=1}^{n}(f(t(H'_i)),\zeta_i) \twoheadrightarrow_{\mathcal{A}} \textcircled{\scriptsize D}_{i=1}^{n}(f(t(H_i)),\zeta_i)$$

Conditions (8) and (7) imply that

$$f(t(H')) \cong f(\textcircled{\scriptsize D}_{i=1}^{n}(f(t(H'_i)),\zeta_i)) \twoheadrightarrow_{\mathcal{A}} f(\textcircled{\scriptsize D}_{i=1}^{n}(f(t(H_i)),\zeta_i)) \cong f(t(H)) \twoheadrightarrow_{\mathcal{A}} T$$

Thus $f(t(H')) \twoheadrightarrow_{\mathcal{A}} T$ and this implies $H' \triangleright T$.

(3) Let $H \triangleright T$ and $T \twoheadrightarrow_{\mathcal{A}} T'$. From the definition of $\triangleright$ it follows that $f(t(H)) \twoheadrightarrow_{\mathcal{A}} T \twoheadrightarrow_{\mathcal{A}} T'$ and therefore $H \triangleright T'$.

[4] In an equation of the form $T \cong T'$ we assume that T is defined if and only if T' is defined. And in a condition of the form $T \twoheadrightarrow_{\mathcal{A}} T'$ we assume that T is defined if T' is defined.

(4) We show that both directions are satisfied:

- we assume that there are type graphs T_i such that $H_i \rhd T_i$. It follows that $f(t(H_i)) \twoheadrightarrow_{\mathcal{A}} T_i$. Since $\mathcal{A}$-morphisms are preserved by graph construction (see above) and by the operation f (see condition (8)) we conclude that

$$f(t(\textcircled{\scriptsize D}_{i=1}^{n}(H_i,\zeta_i))) \overset{(7)}{\cong} f(\textcircled{\scriptsize D}_{i=1}^{n}(f(t(H_i)),\zeta_i)) \twoheadrightarrow_{\mathcal{A}} f(\textcircled{\scriptsize D}_{i=1}^{n}(T_i,\zeta_i))$$

  And therefore $\textcircled{\scriptsize D}_{i=1}^{n}(H_i,\zeta_i) \rhd f(\textcircled{\scriptsize D}_{i=1}^{n}(T_i,\zeta_i))$.
- let $\textcircled{\scriptsize D}_{i=1}^{n}(H_i,\zeta_i) \rhd T$. From the definition of $\rhd$ and with (7) it follows that

$$f(\textcircled{\scriptsize D}_{i=1}^{n}(f(t(H_i)),\zeta_i)) \twoheadrightarrow_{\mathcal{A}} T$$

  We set $T_i = f(t(H_i))$ and $H_i \rhd T_i$ is also satisfied.

(5) Let H be typable, i.e. $T = f(t(H))$ is defined. We show that T is the minimal type. If $H \rhd T'$ for any type graph T' it follows from the definition of $\rhd$ that $T \cong f(t(H)) \twoheadrightarrow_{\mathcal{A}} T'$. If, on the other hand, $T \cong f(t(H)) \twoheadrightarrow_{\mathcal{A}} T'$, it follows immediately with (3) that $H \rhd T'$.

We will now show that (1)–(5) imply (9) and (10).

(9) We assume that $Y(f(t(H)))$ holds. Since $H \rhd f(t(H))$, condition (1) implies that $X(H)$ holds.
(10) let $(L,R) \in \mathcal{R}$, that is $L \to_{\mathcal{R}} R$ and $L \rhd f(t(L))$. Condition (2) implies that $R \rhd f(t(L))$. And from the definition of $\rhd$ it follows that $f(t(R)) \twoheadrightarrow_{\mathcal{A}} f(t(L))$.

Note: it is a direct consequence of condition (7) above that the operator f is idempotent, i.e. $f(f(T)) \cong f(T)$. Just take the identity graph construction with $n = 1$ and $\zeta_1 : \mathbf{n} \twoheadrightarrow \mathbf{n}$ where $n = ar(T)$.

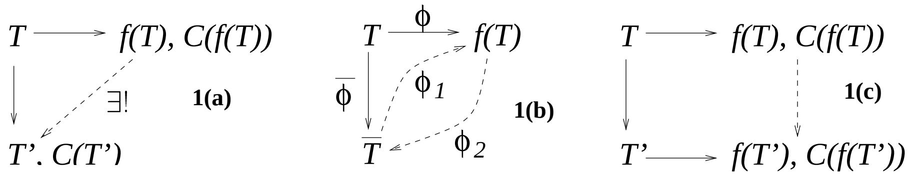
The operation f can often be characterised by a universal property with the intuitive notion that $f(T)$ is the "smallest" type graph (wrt. the preorder $\twoheadrightarrow_{\mathcal{A}}$) for which $T \twoheadrightarrow_{\mathcal{A}} f(T)$ and a property C hold. The morphism $T \twoheadrightarrow_{\mathcal{A}} f(T)$ can also be seen as the initial element in a comma category $(T \downarrow F)$ where F is the obvious functor from the category of hypergraphs satisfying C (with all strong morphisms between them) into the category of all hypergraphs with strong morphisms.

Proposition 2. *Let C be a property on type graphs such that $f(T)$ can be characterised in the following way: $f(T)$ satisfies C, there is a morphism $\phi : T \twoheadrightarrow_{\mathcal{A}} f(T)$ and for every other morphism $\phi' : T \to_{\mathcal{A}} T'$ where $C(T')$ holds, there is a unique morphism $\psi : f(T) \to_{\mathcal{A}} T'$ such that $\psi \circ \phi = \phi'$. Furthermore we demand that if there exists a morphism $\phi : T \to_{\mathcal{A}} T'$ such that $C(T')$ holds, then $f(T)$ is defined.*

Then if $f(T)$ is defined, it is unique up to isomorphism. Furthermore f satisfies conditions (7) and (8).

Proof. We first show that $f(T)$ is unique up to isomorphism, if it exists. The property (also called universal property) which $f(T)$ satisfies is depicted in Figure 1(a). Let us assume that there is another graph $\overline{T}$ which also satisfies this property.

Thus $\phi : T \twoheadrightarrow_{\mathcal{A}} f(T)$ and $\overline{\phi} : T \twoheadrightarrow_{\mathcal{A}} \overline{T}$. Because $f(T)$ as well as $\overline{T}$ can take the role of T' in Figure 1(a), it follows that there exist morphisms $\phi_1 : \overline{T} \twoheadrightarrow_{\mathcal{A}} f(T)$ and $\phi_2 : f(T) \twoheadrightarrow_{\mathcal{A}} \overline{T}$ such that $\phi_1 \circ \overline{\phi} = \phi$ and $\phi_2 \circ \phi = \overline{\phi}$ (see 1(b)).



The universal property must also hold if we set $T' = f(T)$ in Figure 1(a) and use the morphism ϕ twice. We have two candidates for the unique morphism ψ satisfying $\psi \circ \phi = \phi$. One is the identity on $f(T)$ and the other is $\phi_1 \circ \phi_2$. Since ψ is unique, it follows that $\phi_1 \circ \phi_2 = id_{f(T)}$, which implies that ϕ_1 is surjective and ϕ_2 is injective.

We can apply the same argument to $\overline{T}$ and obtain $\phi_2 \circ \phi_1 = id_{\overline{T}}$, which implies that ϕ_2 is surjective and ϕ_1 is injective.

So both morphisms, ϕ_1 and ϕ_2, are isomorphisms and we conclude that $f(T) \cong \overline{T}$.

We next show that condition (8) is satisfied. If $T \rightarrow_{\mathcal{A}} T'$ and $T' \twoheadrightarrow_{\mathcal{A}} f(T')$ (because of the definition of f), it follows that $T \rightarrow_{\mathcal{A}} f(T')$, and because of the universal property for $f(T)$, this implies the existence of a morphism $f(T) \rightarrow_{\mathcal{A}} f(T')$ (see Figure 1(c)).

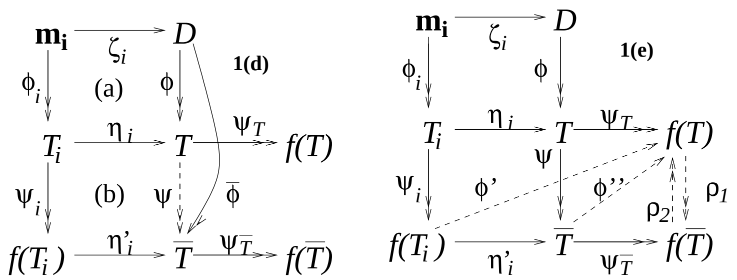
If $T \twoheadrightarrow_{\mathcal{A}} T'$ is a strong morphism, then $f(T) \twoheadrightarrow_{\mathcal{A}} f(T')$ is also strong.

At last we prove that condition (7) holds. Let $T = \textcircled{D}_{i=1}^{n}(T_i, \zeta_i)$ and $\overline{T} = \textcircled{D}_{i=1}^{n}(f(T_i), \zeta_i)$. We show that there exists an isomorphism from $f(T)$ to $f(\overline{T})$.

Since T (together with morphisms η_i and ϕ) is the colimit of the ζ_i and the ϕ_i, the square (a) in Figure 1(d) below commutes and is a colimit. Let $\psi_i : T_i \twoheadrightarrow_{\mathcal{A}} f(T_i)$ be the $\mathcal{A}$-morphisms satisfying the universal property. The $\psi_i \circ \phi_i$ are the only strong morphisms from $\mathbf{m_i}$ into $f(T_i)$. Let $\overline{T}$ (together with morphisms η'_i and $\overline{\phi}$) be the colimit generated by the ζ_i and the $\psi_i \circ \phi_i$.

It is a standard property of colimits (see e.g. [2]) that this implies the existence of a morphism $\psi : T \twoheadrightarrow_{\mathcal{A}} \overline{T}$ such that $\psi \circ \phi = \overline{\phi}$ and the square (b) in Figure 1(d) is again a colimit.

Furthermore let $\psi_T : T \twoheadrightarrow_{\mathcal{A}} f(T)$ and $\psi_{\overline{T}} : \overline{T} \twoheadrightarrow_{\mathcal{A}} f(\overline{T})$ be the morphisms satisfying the universal property.



Since $\psi_{\overline{T}} \circ \psi : T \twoheadrightarrow_{\mathcal{A}} f(\overline{T})$, it follows from the universal property of $f(T)$ that there is a unique morphism $\rho_1 : f(T) \twoheadrightarrow_{\mathcal{A}} f(\overline{T})$ such that $\rho_1 \circ \psi_T = \psi_{\overline{T}} \circ \psi$.

It also holds that $\psi_T \circ \eta_i : T_i \rightarrow_{\mathcal{A}} f(T)$ and the universal property of $f(T_i)$ implies the existence of a unique morphism $\phi' : f(T_i) \rightarrow_{\mathcal{A}} f(T)$ such that $\phi' \circ \psi_i = \psi_T \circ \eta_i$.

Furthermore $\phi' : f(T_i) \rightarrow_{\mathcal{A}} f(T)$ and $\psi_T : T \twoheadrightarrow_{\mathcal{A}} f(T)$. Thus the universal property of the colimit (b) implies that there exists a morphism $\phi'' : \overline{T} \twoheadrightarrow_{\mathcal{A}} f(T)$ such that $\phi'' \circ \eta'_i = \phi'$ and $\phi'' \circ \psi = \psi_T$ (the last condition implies that ϕ'' is a strong morphism).

At last, the morphism $\phi'' : \overline{T} \twoheadrightarrow_{\mathcal{A}} f(T)$ and the universal property of $f(\overline{T})$ imply that there exists a unique morphism $\rho_2 : f(\overline{T}) \twoheadrightarrow_{\mathcal{A}} f(T)$ such that $\rho_2 \circ \psi_{\overline{T}} = \phi''$.

In the last step we will show that ρ_1 and ρ_2 are both bijective and inverse to each other by proving $\rho_1 \circ \rho_2 = id_{f(\overline{T})}$ and $\rho_2 \circ \rho_1 = id_{f(T)}$.

- we show $\rho_2 \circ \rho_1 = id_{f(T)}$. It holds that $(\rho_2 \circ \rho_1) \circ \psi_T = \rho_2 \circ \psi_{\overline{T}} \circ \psi = \phi'' \circ \psi = \psi_T$.
  Since the morphism satisfying this property is unique (universal property of $f(T)$), and $id_{f(T)}$ is already satisfying it, we conclude that $\rho_2 \circ \rho_1 = id_{f(T)}$.
- we show that $\rho_1 \circ \phi' = \psi_{\overline{T}} \circ \eta'_i$. It holds that $(\rho_1 \circ \phi') \circ \psi_i = \rho_1 \circ \psi_T \circ \eta_i = \psi_{\overline{T}} \circ \psi \circ \eta_i = (\psi_{\overline{T}} \circ \eta'_i) \circ \psi_i$. A morphism satisfying this property is unique (universal property of $f(T_i)$) and thus $\rho_1 \circ \phi' = \psi_{\overline{T}} \circ \eta'_i$.
- we show that $\rho_1 \circ \phi'' = \psi_{\overline{T}}$. It holds that $(\rho_1 \circ \phi'') \circ \psi = \rho_1 \circ \psi_T = \psi_{\overline{T}} \circ \psi$ and $(\rho_1 \circ \phi'') \circ \eta'_i = \rho_1 \circ \phi' = \psi_{\overline{T}} \circ \eta'_i$. A morphism satisfying this property is unique (universal property of the colimit (b)) and thus $\rho_1 \circ \phi'' = \psi_{\overline{T}}$.

– finally we show that $\rho_1 \circ \rho_2 = id_{f(\overline{T})}$. It holds that $(\rho_1 \circ \rho_2) \circ \psi_{\overline{T}} = \rho_1 \circ \phi'' = \psi_{\overline{T}}$. From the universal property of $f(\overline{T})$ we know that a morphism satisfying this property is unique and $id_{f(\overline{T})}$ is already satisfying it. So it follows that $\rho_1 \circ \rho_2 = id_{f(\overline{T})}$.

It is left to show that $f(T)$ is defined if and only if $f(\overline{T})$ is defined: let $f(\overline{T})$ be defined, then there is, according to Figure 1(d), a morphism $\psi_{\overline{T}} \circ \psi : T \twoheadrightarrow_{\mathcal{A}} f(\overline{T})$, i.e. there is a morphism from T into a hypergraph satisfying C. The preconditions then imply that $f(T)$ is defined.

If, on the other hand, $f(T)$ is defined, then there are morphisms $\psi_T \circ \eta_i : T_i \to_{\mathcal{A}} f(T)$, which implies that the $f(T_i)$ and therefore also $\overline{T}$ are defined. It follows that there exists a morphism $\phi'' : \overline{T} \twoheadrightarrow_{\mathcal{A}} f(T)$ (see Figure 1(e)) where $f(T)$ satisfies C and therefore $f(\overline{T})$ is defined.

4 Case Studies

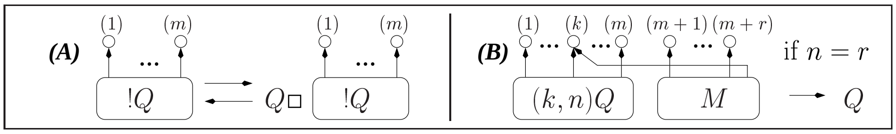
4.1 A Type System for the Polyadic π-Calculus

We present a graph rewriting semantics for the asynchronous polyadic π-calculus [17] without choice and matching, already introduced in [12]. Different ways of encoding the π-calculus into graph rewriting can be found in [21, 5, 4].

We apply the theory presented in section 3, introduce a type system avoiding runtime errors produced by mismatching arities and show that it satisfies the conditions of Theorem 1. Afterwards we show that a graph has a type if and only if the corresponding π-calculus process has a type in a standard type system with infinite regular trees.

Definition 6. (Process Graphs) *A* process graph P *is inductively defined as follows: P is a hypergraph with a duplicate-free string of external nodes. Furthermore each edge e is either labelled with $(k,n)Q$ where Q is again a process graph, $1 \leq n \leq ar(Q)$ and $1 \leq k \leq ar(e) = ar(Q) - n$ (e is a process waiting for a message with n ports arriving at its k-th node), with $!Q$ where $ar(Q) = ar(e)$ (e is a process which can replicate itself) or with the constant M (e is a message sent to its last node).*

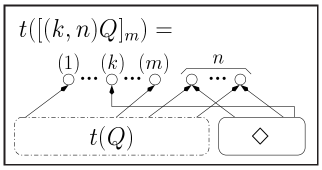
The reduction relation is generated by the rules in (A) (replication) and by rule (B) (reception of a message by a process) and is closed under isomorphism and graph construction.



A process graph may contain a bad redex, if it contains a subgraph corresponding to the left-hand side of rule (B) with $n \neq r$, so we define the predicate X as follows: $X(P)$ if and only if P does not contain a bad redex.

We now propose a type system for process graphs by defining the mappings t and f. (Note that in this case, the type graphs are trivially annotated by $\perp$, and so we omit the annotation mapping.)

The linear t mapping is defined on the hyperedges as follows: $t([M]_n) = [\diamond]_n$ ($\diamond$ is a new edge label), $t([!Q]_m) = t(Q)$ and $t([(k,n)Q]_m)$ is defined as in the image to the right (in the notation explained after Definition 3). It is only defined if $n + m = ar(Q)$.



The mapping f is defined as in Proposition 2 where C is defined as follows[5]

$$C(T) \iff \forall e_1, e_2 \in E_T : (\lfloor s_T(e_1) \rfloor_{ar(e_1)} = \lfloor s_T(e_2) \rfloor_{ar(e_2)} \Rightarrow e_1 = e_2)$$

[5] $\lfloor s \rfloor_i$ extracts the i-th element of a string s.

The linear mapping t extracts the communication structure from a process graph, i.e. an edge of the form $[\diamond]_n$ indicates that its nodes (except the last) might be sent or received via its last node. Then f 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 $f(t(H))$.

The condition that we want to check is simply that $f(t(H))$ is defined. Thus we set $Y(T) = \mathit{true}$ for every type graph T.

Proposition 3. *The trivial annotation mapping $\mathcal{A}$ (where every lattice consists of a single element $\bot$), the mappings f and t and the predicates X and Y defined above satisfy conditions (6)–(10) of Theorem 1. Thus if $P \triangleright T$ and $Y(T)$, then P will never produce a bad redex during reduction.*

Proof. We show that $\mathcal{A}$, f, t, X, Y satisfy the conditions of Theorem 1.

(6) This holds obviously since $Y(T) = \mathit{true}$ for every graph T.

(7)&(8) We have to show that the conditions of Proposition 2 are satisfied. The universal property holds by definition and it is left to show that $f(T)$ is defined whenever there is a morphism $\phi : T \to T'$ with $C(T')$.

We define a condition on equivalences $\sim$ on the nodes and edges of T:

$$(\lfloor s_T(e)\rfloor_{ar(e)} \sim \lfloor s_T(e')\rfloor_{ar(e')} \;\Rightarrow\; e \sim e') \wedge (e \sim e' \;\Rightarrow\; \forall j\colon \lfloor s_T(e)\rfloor_j \sim \lfloor s_T(e')\rfloor_j)$$
$$\wedge\, (e \sim e' \;\Rightarrow\; l_T(e) = l_T(e')) \qquad (11)$$

That is, a hypergraph factored by $\sim$ is well-defined (since $\sim$ is a congruence) and it satisfies C.

It is easy to check that the intersection of two equivalences satisfying this condition, again satisfies the condition. So the smallest equivalence $\approx$ satisfying (11) does either not exist (if there are no equivalences satisfying (11)), or it is the intersection of all equivalences satisfying the condition.

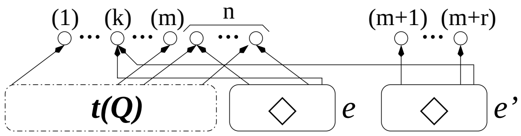
Now let the equivalence $\sim'$ on T be defined in the following way: $e \sim' e' \iff \phi(e) = \phi(e')$ and $v \sim' v' \iff \phi(v) = \phi(v')$. It is straightforward to check that $\sim'$ also satisfies condition (11), therefore $\approx$ exists and $\approx \subseteq \sim'$.

So we can define $f(T) = T/\!\approx$, i.e. T factored by the equivalence $\approx$. The morphism $\psi : T \twoheadrightarrow f(T)$ maps each node or edge to its equivalence class and obviously $f(T)$ satisfies condition C.

We still have to show that the universal property holds: let $\phi : T \to T'$ again be a morphism such that $C(T')$ holds. We define an equivalence $\sim'$ as above. A morphism $\phi' : f(T) \to T'$ can be defined as follows $\phi'([v]_\approx) = \phi(v)$ and $\phi'([e]_\approx) = \phi(e)$. It is well defined because of $\approx \subseteq \sim'$ and it satisfies $\phi' \circ \psi = \phi$.

It is left to show that ϕ' is unique: we assume that there is another morphism $\phi'' : f(T) \to T'$ such that $\phi'' \circ \psi = \phi$. Then $\phi''([v]_\approx) = \phi''(\psi(v)) = \phi(v) = \phi'([v]_\approx)$. The same is true for the edges and so ϕ' and ϕ'' coincide.

(9) Let $Y(f(t(H)))$ hold, which means that $f(t(H))$ is defined. Let us assume that H contains a bad redex Red, which implies that $t(H)$ contains $t(\mathit{Red})$ which is depicted in the figure below ($n \neq r$).

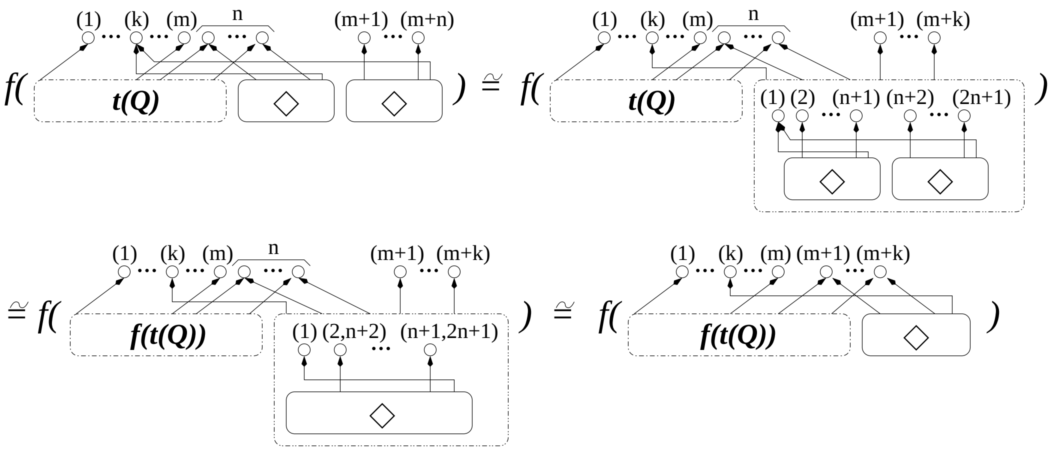


Furthermore $f(t(H))$ is defined only if $f(t(\mathit{Red}))$ is defined. We show that $f(t(\mathit{Red}))$ is undefined. The two edges to the right of $t(\mathit{Red})$ are denoted by e respectively e'. We assume that there is a morphism $\psi : t(\mathit{Red}) \twoheadrightarrow_{\mathcal{A}} f(t(\mathit{Red}))$. It holds that $\lfloor s_{f(t(\mathit{Red}))}(\psi(e))\rfloor_{ar(e)} = \psi(\lfloor s_{t(\mathit{Red})}(e)\rfloor_{ar(e)}) = \psi(\lfloor s_{t(\mathit{Red})}(e')\rfloor_{ar(e)}) = \lfloor s_{f(t(\mathit{Red}))}(\psi(e'))\rfloor_{ar(e)}$ and condition C implies that $\psi(e) = \psi(e')$, but since they have different arities, this can not be the case.

(10) We show the local subject reduction property for both rules (or in the case of (A) for both directions)

(A) We first consider the direction from right to left, i.e. $L = Q\Box[!Q]_m$ and $R = [!Q]_m$. We assume that $f(t(Q\Box[!Q]_m))$ is defined and since $f(t(Q\Box[!Q]_m)) \cong f(f(t(Q))\Box t([!Q]_m))$ we know that $f(t(Q))$ is also defined and there exists a strong $\mathcal{A}$-morphism from $f(t(Q))$ into $f(t(Q\Box[!Q]_m))$.
Concerning the direction from left to right, i.e. $L = [!Q]_m$ and $R = Q\Box[!Q]_m$: we assume that $f(t(L)) \cong f(t(Q))$ is defined and therefore $f(t(Q))\Box f(t(Q)) \cong f(t(Q))\Box f(t([!Q]_m))$ is defined. Since there is an $\mathcal{A}$-morphism from this graph into $f(t(Q)) \cong f(t(L))$, it follows with Proposition 2, that $f(t(R)) \cong f(f(t(Q))\Box f(t([!Q]_m)))$ is defined and that $f(t(R)) \twoheadrightarrow_{\mathcal{A}} f(f(t(L))) \cong f(t(L))$.

(B) In this case $f(t(R)) \cong f(t(Q))$ and $t(L)$ is the type graph depicted above in the proof of condition (9), but in this case $r = n$. With condition (7) and the graph construction operation we can transform $f(t(L))$ as shown in the figure below.



And from the fact that $\mathcal{A}$-morphisms are preserved by graph construction, from condition (8) and from $f(t(Q)) \cong f(f(t(Q)))$, it follows that $f(t(R)) \cong f(t(Q)) \twoheadrightarrow_{\mathcal{A}} f(t(L))$.

We now compare our type system to a standard type system of the π-calculus. An encoding of process graphs into the asynchronous π-calculus (for the operational semantics of the π-calculus see appendix A) can be defined as follows.

Definition 7. (Encoding) *Let P be a process graph, let $\mathcal{N}$ be the name set of the π-calculus and let $\tilde{t} \in \mathcal{N}^*$ such that $|\tilde{t}| = ar(P)$. We define $\Theta_{\tilde{t}}(P)$ inductively as follows:*

$$\Theta_{a_1\ldots a_{n+1}}([M]_{n+1}) = \overline{a_{n+1}}\langle a_1,\ldots,a_n\rangle \qquad \Theta_{\tilde{t}}([!Q]_m) = !\,\Theta_{\tilde{t}}(Q)$$
$$\Theta_{a_1\ldots a_m}([(k,n)Q]_m) = a_k(x_1,\ldots,x_n).\Theta_{a_1\ldots a_m x_1\ldots x_n}(Q)$$
$$\Theta_{\tilde{t}}(\textcircled{D}_{i=1}^{n}(P_i,\zeta_i)) = (\nu\,\mu(V_D\backslash Set(\chi_D)))(\Theta_{\mu(\zeta_1(\chi_{\mathbf{m_1}}))}(P_1) \mid \ldots \mid \Theta_{\mu(\zeta_n(\chi_{\mathbf{m_n}}))}(P_n))$$

where $\zeta_i : \mathbf{m_i} \to D$, $1 \le i \le n$ and $\mu : V_D \to \mathcal{N}$ is a mapping such that μ restricted to $V_D\backslash Set(\chi_D)$ is injective, $\mu(V_D\backslash Set(\chi_D)) \cap \mu(Set(\chi_D)) = \emptyset$ and $\mu(\chi_D) = \tilde{t}$. Furthermore the $x_1,\ldots,x_n \in \mathcal{N}$ are fresh names.

The encoding of a discrete graph is included in the last case, if we set $n = 0$ and assume that the empty parallel composition yields the nil process **0**.

An operational correspondence can be stated as follows:

Proposition 4. *Let p be an arbitrary expression in the asynchronous polyadic π-calculus without summation. Then there exists a process graph P and a duplicate-free string $\tilde{t} \in \mathcal{N}^*$ such that $\Theta_{\tilde{t}}(P) \equiv p$. Furthermore for process graphs P, P' and for every duplicate-free string $\tilde{t} \in \mathcal{N}^*$ with $|\tilde{t}| = ar(P) = ar(P')$ it is true that:*

- *$P \cong P'$ implies $\Theta_{\tilde{t}}(P) \equiv \Theta_{\tilde{t}}(P')$*
- *$P \to^* P'$ implies $\Theta_{\tilde{t}}(P) \to^* \Theta_{\tilde{t}}(P)$*
- *$\Theta_{\tilde{t}}(P) \to^* p \neq wrong$ implies that $P \to^* Q$ and $\Theta_{\tilde{t}}(Q) \equiv p$ for some process graph Q.*

– $\Theta_{\tilde{t}}(P) \to^* \mathit{wrong}$ *if and only if* $P \to^* P'$ *for some process graph* P' *containing a bad redex*

Proof. We do not show this proposition here, but refer the reader to [12].

We now compare our type system with a standard type system of the π-calculus: a *type tree* is a potentially infinite ordered tree with only finitely many non-isomorphic subtrees. A type tree is represented by the tuple $[t_1, \ldots, t_n]$ where $t_1, \ldots, t_n$ are again type trees, the children of the root. A type assignment $\Gamma = x_1 : t_1, \ldots, x_n : t_n$ assigns names to type trees where $\Gamma(x_i) = t_i$. The rules of the type system are simplified versions of the ones from [19], obtained by removing the subtyping annotations.

$$\Gamma \vdash \mathbf{0} \qquad \frac{\Gamma \vdash p \quad \Gamma \vdash q}{\Gamma \vdash p \mid q} \qquad \frac{\Gamma \vdash p}{\Gamma \vdash\, !p} \qquad \frac{\Gamma, a : t \vdash p}{\Gamma \vdash (\nu a)p}$$

$$\frac{\Gamma(a) = [t_1, \ldots, t_m] \quad \Gamma, x_1 : t_1, \ldots, x_m : t_m \vdash p}{\Gamma \vdash a(x_1, \ldots, x_m).p} \qquad \frac{\Gamma(a) = [\Gamma(a_1) \ldots, \Gamma(a_m)]}{\Gamma \vdash \overline{a}\langle a_1, \ldots, a_m\rangle}$$

We will now show that if a process graph has a type, then its encoding has a type in the π-calculus type system and vice versa. In order to express this we first describe the unfolding of a type graph into type trees.

Proposition 5. *Let T be a type graph and let σ be a mapping from V_T into the set of type trees. The mapping σ is called consistent, if it satisfies for every edge $e \in E_T$: $s_T(e) = v_1 \ldots v_n v \;\Rightarrow\; \sigma(v) = [\sigma(v_1), \ldots, \sigma(v_n)]$. Every type graph of the form $f(t(P))$ has such a consistent mapping.*

Let $P \triangleright T$ with $n = ar(T)$ and let σ be a consistent mapping for T. Then it holds for every duplicate-free string $\tilde{t}$ of length n that $\lfloor\tilde{t}\rfloor_1 : \sigma(\lfloor\chi_T\rfloor_1), \ldots, \lfloor\tilde{t}\rfloor_n : \sigma(\lfloor\chi_T\rfloor_n) \vdash \Theta_{\tilde{t}}(P)$.

Now let $\Gamma \vdash \Theta_{\tilde{t}}(P)$. Then there exists a type graph T such that $P \triangleright T$ and a consistent mapping σ such that for every $1 \leq i \leq |\tilde{t}|$ it holds that $\sigma(\lfloor\chi_T\rfloor_i) = \Gamma(\lfloor\tilde{t}\rfloor_i)$.

Proof. We first show two lemmata:

Lemma A: let $\phi : T \to T'$ and let σ' be a mapping which is consistent for T'. We define $\sigma(v) = \sigma'(\phi(v))$ for every $v \in V_T$ and claim that σ is consistent for T.
Proof: let $e \in E_T$ with $s_T(e) = v_1 \ldots v_n v$. Then

$$\begin{aligned}
\sigma(v) &= \sigma'(\phi(v)) = \sigma'(\phi(\lfloor s_T(e)\rfloor_{n+1})) = \sigma'(\lfloor s_{T'}(\phi(e))\rfloor_{n+1}) \\
&= [\sigma'(\lfloor s_{T'}(\phi(e))\rfloor_1), \ldots, \sigma'(\lfloor s_{T'}(\phi(e))\rfloor_n)] \\
&= [\sigma'(\phi(\lfloor s_T(e)\rfloor_1)), \ldots, \sigma'(\phi(\lfloor s_T(e)\rfloor_n))] = [\sigma(\lfloor s_T(e)\rfloor_1), \ldots, \sigma(\lfloor s_T(e)\rfloor_n)] \\
&= [\sigma(v_1), \ldots, \sigma(v_n)]
\end{aligned}$$

Lemma B: let T be a type graph which has a consistent mapping σ. Then it holds that $f(T)$ is defined and there is a consistent mapping σ' for $f(T)$ such that $\sigma'(\lfloor\chi_{f(T)}\rfloor_i) = \sigma(\lfloor\chi_T\rfloor_i)$.
Proof: we define an equivalence $\approxeq$ on the nodes and edges of T in the following way: $e_1 \approxeq e_2 \iff \sigma(\lfloor s_T(e_1)\rfloor_{ar(e_1)}) = \sigma(\lfloor s_T(e_2)\rfloor_{ar(e_2)})$ and $v_1 \approxeq v_2$ if and only if $v_1 = v_2$ or there are edges e_1, e_2 and an index i such that $e_1 \approxeq e_2$ and $\lfloor s_T(e_j)\rfloor_i = v_j$, $j \in \{1, 2\}$.
Factoring T by $\approxeq$ yields a well-defined hypergraph $T/\approxeq$.
We will prove that $v_1 \approxeq v_2$ implies $\sigma(v_1) = \sigma(v_2)$: if $v_1 \approxeq v_2$ then either $v_1 = v_2$ and the claim is obviously true, or there are edges e_1, e_2 such that $e_1 \approxeq e_2$ and $\lfloor s_T(e_j)\rfloor_i = v_j$, $j \in \{1, 2\}$. $e_1 \approxeq e_2$ implies that $\sigma(\lfloor s_T(e_1)\rfloor_{ar(e_1)}) = \sigma(\lfloor s_T(e_2)\rfloor_{ar(e_2)})$. Since σ is consistent it holds that $\sigma(v_1) = \sigma(\lfloor s_T(e_1)\rfloor_i) = \sigma(\lfloor s_T(e_2)\rfloor_i) = \sigma(v_2)$.
We define a consistent mapping $\sigma_{\approxeq}$ for $T/\approxeq$: let $\sigma_{\approxeq}([v]_{\approxeq}) = \sigma(v)$. From what we have just shown, it follows that $\sigma_{\approxeq}$ is well-defined and it is left to show that it is consistent: let $s_{T/\approxeq}([e]_{\approxeq}) = [v_1]_{\approxeq} \ldots [v_n]_{\approxeq}[v_{n+1}]_{\approxeq}$ such that $v_j = \lfloor s_T(e)\rfloor_j$. It follows that $\sigma_{\approxeq}([v_{n+1}]_{\approxeq}) = \sigma(v_{n+1}) = [\sigma(v_1), \ldots, \sigma(v_n)] = [\sigma_{\approxeq}([v_1]_{\approxeq}), \ldots, \sigma_{\approxeq}([v_n]_{\approxeq})]$ and therefore $\sigma_{\approxeq}$ is consistent.
We next show that $T/\approxeq$ satisfies condition C: let $[e_1]_{\approxeq}$ and $[e_2]_{\approxeq}$ be two edges such that $\lfloor s_{T/\approxeq}([e_1]_{\approxeq})\rfloor_{ar(e_1)} = \lfloor s_{T/\approxeq}([e_2]_{\approxeq})\rfloor_{ar(e_2)}$. Then $[\lfloor s_T(e_1)\rfloor_{ar(e_1)}]_{\approxeq} = [\lfloor s_T(e_2)\rfloor_{ar(e_2)}]_{\approxeq}$ and also

$\lfloor s_T(e_1)\rfloor_{ar(e_1)} \approxeq \lfloor s_T(e_2)\rfloor_{ar(e_2)}$. Therefore $\sigma(\lfloor s_T(e_1)\rfloor_{ar(e_1)}) = \sigma(\lfloor s_T(e_2)\rfloor_{ar(e_2)})$ and from the definition of $\approxeq$ it follows that $e_1 \approxeq e_2$ and thus $[e_1]_{\approxeq} = [e_2]_{\approxeq}$.
Now let $\phi : T \twoheadrightarrow T/\approxeq$ a morphism which maps each node or edge to its equivalence class. Since $C(T/\approxeq)$ holds, it follows from Proposition 2 and the definition of f, that $f(T)$ is defined, and there are morphisms $\psi : T \twoheadrightarrow f(T)$ and $\phi' : f(T) \twoheadrightarrow T/\approxeq$ such that $\phi' \circ \psi = \phi$.
With Lemma A, we can derive a consistent mapping σ' for $f(T)$ by defining $\sigma'(v') = \sigma_{\approxeq}(\phi'(v'))$. And it holds that

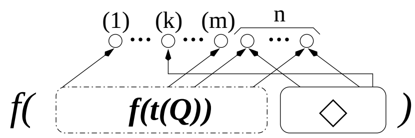
$$\sigma(\lfloor \chi_T \rfloor_i) = \sigma_{\approxeq}([\lfloor \chi_T \rfloor_i]_{\approxeq}) = \sigma_{\approxeq}(\lfloor \chi_{T/\approxeq} \rfloor_i) = \sigma_{\approxeq}(\phi'(\lfloor \chi_{f(T)} \rfloor_i)) = \sigma'(\lfloor \chi_{f(T)} \rfloor_i)$$

We first show that $f(t(P))$ has a consistent mapping σ: we know that $f(t(P))$ satisfies condition C. So we can define: for any node v of T for which there is no edge $e \in V_T$ with $\lfloor s_T(e)\rfloor_{ar(e)} = v$, let $\sigma(v)$ be an arbitrary type tree. For all other nodes v there is a unique edge e with $\lfloor s_T(e)\rfloor_{ar(e)} = v$ and we set $\sigma(v) = [\sigma(v_1), \ldots, \sigma(v_n)]$ if $s_T(e) = v_1 \ldots v_n v$. The definition has a smallest fixed-point, which is then our mapping σ.

We can now prove the two main parts of the proposition.

Type graphs $\rightarrow$ π-calculus: let $P \triangleright T$ which implies that there is a morphism $\phi : f(t(P)) \twoheadrightarrow T$. Furthermore let σ' be a mapping which is consistent for T.
Now we can define a mapping σ on the nodes of $f(t(P))$ with $\sigma(v) = \sigma'(\phi(v))$. From Lemma A it follows that σ is also consistent and furthermore σ and σ' coincide on the external nodes. So it is sufficient to show our claim for $T = f(t(P))$.
We will do so by induction on P but with a stronger induction hypothesis: let $\tilde{t}$ be a string of names (possibly with duplicates), σ be a consistent mapping for $T = f(t(P))$ such that $\lfloor \tilde{t} \rfloor_i = \lfloor \tilde{t} \rfloor_j$ implies $\sigma(\lfloor \chi \rfloor_i) = \sigma(\lfloor \chi \rfloor_j)$ (we will say that σ and $\tilde{t}$ are compatible). Then it holds that $\lfloor \tilde{t} \rfloor_1 : \sigma(\lfloor \chi_T \rfloor_1), \ldots, \lfloor \tilde{t} \rfloor_n : \sigma(\lfloor \chi_T \rfloor_n) \vdash \Theta_{\tilde{t}}(P)$.
From this we can derive the original claim, since there we demand that $\tilde{t}$ is duplicate-free.

- $P = [M]_{n+1}$, that is $T = f(t(P)) = [\Diamond]_{n+1}$ and let σ be a consistent mapping for T and let $\tilde{t}$ be a string of names, compatible with σ.
  The type assignment $\Gamma = \lfloor \tilde{t} \rfloor_i : \sigma(\lfloor \chi_T \rfloor_i)$ is well-defined because of the compatibility of σ and $\tilde{t}$. And since σ is consistent, it holds that $\Gamma(\lfloor \tilde{t} \rfloor_{n+1}) = \sigma(\lfloor \chi_T \rfloor_{n+1}) = [\sigma(\lfloor \chi_T \rfloor_1), \ldots, \sigma(\lfloor \chi_T \rfloor_n)] = [\Gamma(\lfloor \tilde{t} \rfloor_1), \ldots, \Gamma(\lfloor \tilde{t} \rfloor_n)]$.
  Therefore the typing rules for the π-calculus imply that $\Gamma \vdash \overline{\lfloor \tilde{t} \rfloor_{n+1}} \langle \lfloor \tilde{t} \rfloor_1, \ldots, \lfloor \tilde{t} \rfloor_n \rangle = \Theta_{\tilde{t}}(P)$.
- $P = [!Q]_m$, that is $T = f(t(P)) = f(t(Q))$. Let σ be a consistent mapping for T and let $\tilde{t}$ be compatible with σ. From the induction hypothesis it follows that $\Gamma = \lfloor \tilde{t} \rfloor_i : \sigma(\lfloor \chi_T \rfloor_i) \vdash \Theta_{\tilde{t}}(Q)$. Then the typing rules for the π-calculus imply that $\Gamma \vdash !\Theta_{\tilde{t}}(Q) = \Theta_{\tilde{t}}(P)$.
- $P = [(k, n)Q]_m$, that is $T = f(t(P))$ has the form depicted in the figure below.

  Now let σ be a consistent mapping for T and let $\tilde{t}$ be compatible with σ. Let $\phi : f(t(Q)) \rightarrow T$ be the embedding of $T' = f(t(Q))$ into T and we define a mapping σ' on the nodes of $f(t(Q))$ such that $\sigma'(v) = \sigma(\phi(v))$. According to Lemma A, σ' is consistent.
  Let $\tilde{s} = \tilde{t}x_1 \ldots x_n$, where $x_1, \ldots, x_n$ are fresh names. Since σ and $\tilde{t}$ are compatible, σ' and $\tilde{s}$ are also compatible.
  So it follows with the induction hypothesis that $\Delta = \lfloor \tilde{s} \rfloor_i : \sigma'(\lfloor \chi_{T'} \rfloor_i) \vdash \Theta_{\tilde{s}}(Q)$.
  Let $v_i = \lfloor s_T(e) \rfloor_i$, where $e \in E_T$ is the unique edge satisfying $\lfloor s_T(e)\rfloor_{ar(e)} = \lfloor \chi_T \rfloor_k$. It holds that $v_i = \phi(\lfloor \chi_{T'} \rfloor_{m+i})$. Furthermore

$$\begin{aligned}\Delta(\lfloor \tilde{t} \rfloor_k) = \Delta(\lfloor \tilde{s} \rfloor_k) = \sigma'(\lfloor \chi_{T'} \rfloor_k) = \sigma(\lfloor \chi_T \rfloor_k) &= [\sigma(v_1), \ldots, \sigma(v_n)] \\ &= [\sigma(\phi(\lfloor \chi_{T'} \rfloor_{m+1})), \ldots, \sigma(\phi(\lfloor \chi_{T'} \rfloor_{m+n}))] \\ &= [\Delta(\lfloor \tilde{s} \rfloor_{m+1}), \ldots, \Delta(\lfloor \tilde{s} \rfloor_{m+n})] = [\Delta(x_1), \ldots, \Delta(x_n)]\end{aligned}$$

This implies that $\lfloor\tilde{t}\rfloor_i : \sigma'(\lfloor\chi_{T'}\rfloor_i) \vdash \lfloor\tilde{t}\rfloor_k(x_1,\ldots,x_n).\Theta_{\tilde{s}}(Q)$. And since $\sigma'(\lfloor\chi_{T'}\rfloor_i) = \sigma(\lfloor\chi_T\rfloor_i)$ for $1 \leq i \leq m$, it holds that $\lfloor\tilde{t}\rfloor_i : \sigma(\lfloor\chi_T\rfloor_i) \vdash \Theta_{\tilde{t}}(P)$.

– $P = Ⓓ_{i=1}^{n}(P_i,\zeta_i)$, i.e. $T = f(t(P)) \cong f(Ⓓ_{i=1}^{n}(f(t(P_i)),\zeta_i))$.
Let $\overline{T} = Ⓓ_{i=1}^{n}(f(t(P_i)),\zeta_i)$, let $\eta_i : f(t(P_i)) \to \overline{T}$ and $\phi : D \to \overline{T}$ be the standard embeddings generated by graph construction and let $\psi : \overline{T} \twoheadrightarrow T$ be the morphism satisfying the universal property of $f(\overline{T}) = T$. We also set $T_i = f(t(P_i))$.
We define mappings σ_i on the T_i by setting $\sigma_i(v) = \sigma(\psi(\eta_i(v)))$. According to Lemma A, the σ_i are consistent.
We now prove a property concerning σ and μ: let $\mu : V_D \to N$ be the function defined in the encoding (Definition 7). We show that $\mu(v) = \mu(v')$ implies $\sigma(\psi(\phi(v))) = \sigma(\psi(\phi(v')))$. If $\mu(v) = \mu(v')$, then either $v = v'$ and the claim holds obviously, or there are indices i, j such that $v = \lfloor\chi_D\rfloor_i$, $v' = \lfloor\chi_D\rfloor_j$ and $\lfloor\tilde{t}\rfloor_i = \lfloor\tilde{t}\rfloor_j$. Then it follows with the fact that σ and $\tilde{t}$ are compatible, that $\sigma(\psi(\phi(v))) = \sigma(\psi(\phi(\lfloor\chi_D\rfloor_i))) = \sigma(\lfloor\chi_T\rfloor_i) = \sigma(\lfloor\chi_T\rfloor_j) = \sigma(\psi(\phi(\lfloor\chi_D\rfloor_j))) = \sigma(\psi(\phi(v')))$.
We show that σ_i and $\mu(\zeta_i(\chi_{\mathbf{m_i}}))$ are compatible: let $\lfloor\mu(\zeta_i(\chi_{\mathbf{m_i}}))\rfloor_j = \lfloor\mu(\zeta_i(\chi_{\mathbf{m_i}}))\rfloor_k$. This implies that $\mu(\zeta_i(\lfloor\chi_{\mathbf{m_i}}\rfloor_j)) = \mu(\zeta_i(\lfloor\chi_{\mathbf{m_i}}\rfloor_k))$ and it follows that

$$\begin{aligned}\sigma_i(\lfloor\chi_{T_i}\rfloor_j) &= \sigma(\psi(\eta_i(\lfloor\chi_{T_i}\rfloor_j))) = \sigma(\psi(\phi(\zeta_i(\lfloor\chi_{\mathbf{m_i}}\rfloor_j)))) = \sigma(\psi(\phi(\zeta_i(\lfloor\chi_{\mathbf{m_i}}\rfloor_k))))\\ &= \sigma(\psi(\eta_i(\lfloor\chi_{T_i}\rfloor_k))) = \sigma_i(\lfloor\chi_{T_i}\rfloor_k)\end{aligned}$$

Then the induction hypothesis implies that

$$\Gamma_i = \lfloor\mu(\zeta_i(\chi_{\mathbf{m_i}}))\rfloor_j : \sigma_i(\lfloor\chi_{T_i}\rfloor_j) \vdash \Theta_{\mu(\zeta_i(\chi_{\mathbf{m_i}}))}(P_i)$$

Now let Δ be a type assignment, containing all assignments of the form $\mu(v) : \sigma(\psi(\phi(v)))$ for all $v \in V_D$. Because of $(\mu(v) = \mu(v') \Rightarrow \sigma(\psi(\phi(v))) = \sigma(\psi(\phi(v'))))$ shown above it follows that Δ is well-defined and since $\sigma_i(\lfloor\chi_{T_i}\rfloor_j) = \sigma(\psi(\phi(\zeta_i(\chi_{\mathbf{m_i}}))))$, it follows that Δ can be obtained from any Γ_i by adding extra type assignments. Thus it follows from the weakening rule for the π-calculus type system, that $\Delta \vdash \Theta_{\mu(\zeta_i(\chi_{\mathbf{m_i}}))}(P_i)$. And with the rule for parallel composition it follows that

$$\Delta \vdash \Theta_{\mu(\zeta_1(\chi_{\mathbf{m_1}}))}(P_1) \mid \ldots \mid \Theta_{\mu(\zeta_n(\chi_{\mathbf{m_n}}))}(P_n)$$

Now let Γ be a type assignment containing all assignments of the form $\mu(v) : \sigma(\psi(\phi(v)))$ for $v \in \mathit{Set}(\chi_D)$. With the rule for hiding of the π-calculus type system, it follows that

$$\Gamma \vdash (\nu\ \mu(V_D \backslash \mathit{Set}(\chi_D)))(\Theta_{\mu(\zeta_i(\chi_{\mathbf{m_i}}))}(P_1) \mid \ldots \mid \Theta_{\mu(\zeta_i(\chi_{\mathbf{m_i}}))}(P_n)) = \Theta_{\tilde{t}}(P)$$

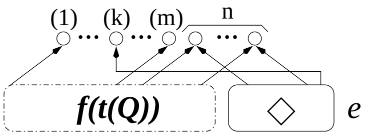
It is left to show that $\Gamma = \lfloor\tilde{t}\rfloor_i : \sigma(\lfloor\chi_T\rfloor_i)$. This is quite straightforward, since $\mu(v)$, $v \in \mathit{Set}(\chi_D)$ is exactly $\lfloor\tilde{t}\rfloor_i$ if $v = \lfloor\chi_D\rfloor_i$, and in this case $\sigma(\psi(\phi(v))) = \sigma(\psi(\phi(\lfloor\chi_D\rfloor_i))) = \sigma(\lfloor\chi_T\rfloor_i)$.

π-calculus $\to$ type graphs: now let $\Gamma \vdash \Theta_{\tilde{t}}(P)$, where $\tilde{t}$ is a duplicate-free sequence of names. We show that $T = f(t(P))$ is defined and that there is a mapping σ consistent with T, such that $\sigma(\lfloor\chi_T\rfloor_i) = \Gamma(\lfloor\tilde{t}\rfloor_i)$. We proceed by induction on P.

– $P = [M]_{n+1}$, which implies that $\Theta_{\tilde{t}}(P) = \overline{\lfloor\tilde{t}\rfloor_{n+1}}\langle\lfloor\tilde{t}\rfloor_1,\ldots,\lfloor\tilde{t}\rfloor_n\rangle$.
Since $\Gamma \vdash p$, it holds that $\Gamma(\lfloor\tilde{t}\rfloor_{n+1}) = [\Gamma(\lfloor\tilde{t}\rfloor_1),\ldots,\Gamma(\lfloor\tilde{t}\rfloor_n)]$.
We set $\overline{T} = t(P) = [\Diamond]_{n+1}$ and furthermore we define a mapping $\overline{\sigma}$ on the nodes of $\overline{T}$ with $\overline{\sigma}(\lfloor\chi_{\overline{T}}\rfloor_i) = \Gamma(\lfloor\tilde{t}\rfloor_i)$. $\overline{\sigma}$ is obviously consistent, and it follows with Lemma B that $T = f(\overline{T}) = f(t(P))$ is defined and has a consistent mapping σ such that $\sigma(\lfloor\chi_T\rfloor_i) = \overline{\sigma}(\lfloor\chi_{\overline{T}}\rfloor_i) = \Gamma(\lfloor\tilde{t}\rfloor_i)$.

– $P = [!Q]_m$, which implies that $\Theta_{\tilde{t}}(P) = !\Theta_{\tilde{t}}(Q)$.
The typing rules of the π-calculus imply that $\Gamma \vdash \Theta_{\tilde{t}}(Q)$. Then it follows from the induction hypothesis that $\overline{T} = f(t(Q))$ is defined, there is a mapping $\overline{\sigma}$ on the nodes of $\overline{T}$ which is consistent and $\overline{\sigma}(\lfloor\chi_{\overline{T}}\rfloor_i) = \Gamma(\lfloor\tilde{t}\rfloor_i)$.
Since $T = f(t(P)) = f(t(Q)) = \overline{T}$, T is also defined and we can use $\sigma = \overline{\sigma}$ as a consistent mapping with the appropriate properties.

– $P = [(k,n)Q]_m$, which implies that $\Theta_{\tilde{t}}(P) = \lfloor\tilde{t}\rfloor_k(x_1,\dots,x_n).\Theta_{\tilde{t}x_1\dots x_n}(Q)$.
It follows from the typing rules of the π-calculus that $\Gamma, x_1 : t_1, \dots, x_n : t_n \vdash \Theta_{\tilde{t}x_1\dots x_n}(Q)$ where $\Gamma(\lfloor\tilde{t}\rfloor_k) = [t_1,\dots,t_n]$. Then the induction hypothesis implies that $\overline{T} = f(t(Q))$ is defined and that there is a consistent mapping $\overline{\sigma}$ for $\overline{T}$ such that $\overline{\sigma}(\lfloor\chi_{\overline{T}}\rfloor_i) = \Gamma(\lfloor\tilde{t}\rfloor_i)$ if $1 \le i \le m$ and $\sigma(\lfloor\chi_{\overline{T}}\rfloor_{m+i}) = t_i$ if $1 \le i \le n$.
Now let $\hat{T}$ be defined as in the figure below.



We know that $f(\hat{T}) = f(t(P))$ if both are defined. In order to show that $f(\hat{T})$ is defined, we construct a consistent mapping $\hat{\sigma}$: let $\phi : \overline{T} \to \hat{T}$ be the standard embedding of $\overline{T}$ into $\hat{T}$, ϕ is bijective on the node sets. So we can define $\hat{\sigma}(\phi(v)) = \overline{\sigma}(v)$. The mapping $\hat{\sigma}$ is consistent for all edges of $\overline{T} = f(t(Q))$, so it suffices to show that $\hat{\sigma}$ is consistent for the new edge e: $s_{\hat{T}}(e) = \phi(\lfloor\chi_{\overline{T}}\rfloor_{m+1})\dots\phi(\lfloor\chi_{\overline{T}}\rfloor_{m+n})\phi(\lfloor\chi_{\overline{T}}\rfloor_k)$ and therefore

$$\begin{aligned}\hat{\sigma}(\phi(\lfloor\chi_{\overline{T}}\rfloor_k)) &= \overline{\sigma}(\lfloor\chi_{\overline{T}}\rfloor_k) = \Gamma(\lfloor\tilde{t}\rfloor_k) = [t_1,\dots,t_n] \\ &= [\overline{\sigma}(\lfloor\chi_{\overline{T}}\rfloor_{m+1}),\dots,\overline{\sigma}(\lfloor\chi_{\overline{T}}\rfloor_{m+n})] \\ &= [\hat{\sigma}(\phi(\lfloor\chi_{\overline{T}}\rfloor_{m+1})),\dots,\hat{\sigma}(\phi(\lfloor\chi_{\overline{T}}\rfloor_{m+n}))]\end{aligned}$$

Thus $\hat{\sigma}$ is consistent for $\hat{T}$ and with Lemma B it follows that $T = f(\hat{T})$ is defined and that there is a consistent mapping σ for T such that $\sigma(\lfloor\chi_T\rfloor_i) = \hat{\sigma}(\lfloor\chi_{\hat{T}}\rfloor_i) = \overline{\sigma}(\lfloor\chi_{\overline{T}}\rfloor_i) = \Gamma(\lfloor\tilde{t}\rfloor_i)$.

– $P = Ⓓ_{i=1}^{n}(P_i,\zeta_i)$ which implies that

$$\Theta_{\tilde{t}}(P) = (\nu\ \mu(V_D\backslash Set(\chi_D)))(\Theta_{\mu(\zeta_1(\chi_{\mathbf{m_1}}))}(P_1) \mid \dots \mid \Theta_{\mu(\zeta_n(\chi_{\mathbf{m_n}}))}(P_n))$$

and $\mu : V_D \to N$ and $\mu(\chi_D) = \tilde{t}$. (Note that this case also covers discrete graphs, i.e. the case where $n = 0$.)
It follows from the typing rules of the π-calculus that there is a type assignment Δ, which is exactly Γ enriched by type assignments of the form $\mu(v) : t_v$ for all $v \in V_D\backslash Set(\chi_D)$ and that $\Delta \vdash \Theta_{\mu(\zeta_i(\chi_{\mathbf{m_i}}))}(P_i)$.
The induction hypothesis thus implies that all $T_i = f(t(P_i))$ are defined and that there are mappings σ_i consistent with T_i such that $\sigma_i(\lfloor\chi_{T_i}\rfloor_j) = \Delta(\lfloor\mu(\zeta_i(\chi_{\mathbf{m_i}}))\rfloor_j)$.
Now we define $\hat{T} = Ⓓ_{i=1}^{n}(T_i,\zeta_i)$. We know that $T = f(t(P))$ is defined if and only if $f(\hat{T})$ is defined. In order to show this, we define a consistent mapping $\hat{\sigma}$ for $\hat{T}$. We assume that $\eta_i : T_i \to \hat{T}$, $\phi : D \twoheadrightarrow \hat{T}$ are the standard embeddings generated by the graph construction and we define $\hat{\sigma}$ in the following way:

$$\hat{\sigma}(\hat{v}) = \begin{cases} \sigma_i(v) & \text{if } \hat{v} = \eta_i(v) \\ \Delta(\mu(v)) = t_v & \text{if } \hat{v} = \phi(v) \end{cases}$$

We first have to show that $\hat{\sigma}$ is well-defined: let $\phi_i : \mathbf{m_i} \twoheadrightarrow T_i$ be the unique strong morphisms from $\mathbf{m_i}$ into T_i. When we restrict all morphisms to the node sets, then the $(\eta_i)_V$ and ϕ_V are still the colimit of the $(\phi_i)_V$ and the $(\zeta_i)_V$. We show that $\Delta \circ \mu \circ (\zeta_i)_V = \sigma_i \circ (\phi_i)_V$:

$$\Delta(\mu(\zeta_i(\lfloor\chi_{\mathbf{m_i}}\rfloor_j))) = \sigma_i(\lfloor\chi_{T_i}\rfloor_j) = \sigma_i(\phi_i(\lfloor\chi_{\mathbf{m_i}}\rfloor_j))$$

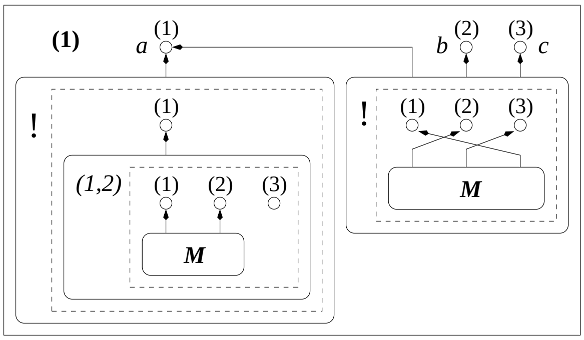
Because of the colimit property, there must be a unique mapping σ' from the nodes of $\hat{T}$ into the set of type trees such that $\sigma' \circ (\eta_i)_V = \sigma_i$ and $\sigma' \circ \phi_V = \Delta \circ \mu$. So σ' is exactly the σ defined above.
It is left to show that $\hat{\sigma}$ is consistent with $\hat{T}$: let e be one of the edges of $\hat{T}$ and it follows that there must be a $e' \in E_{T_i}$ such that $e = \eta_i(e')$. If $s_{\hat{T}}(e) = v_1 \dots v_{n+1}$, it follows that

$v_j = \eta_i(\lfloor s_{T_i}(e')\rfloor_j)$. Thus

$$\begin{aligned}\hat\sigma(v_{n+1}) &= \hat\sigma(\eta_i(\lfloor s_{T_i}(e')\rfloor_{n+1})) = \sigma_i(\lfloor s_{T_i}(e')\rfloor_{n+1})\\ &= [\sigma_i(\lfloor s_{T_i}(e')\rfloor_1),\ldots,\sigma_i(\lfloor s_{T_i}(e')\rfloor_n)]\\ &= [\hat\sigma(\eta_i(\lfloor s_{T_i}(e')\rfloor_1)),\ldots,\hat\sigma(\eta_i(\lfloor s_{T_i}(e')\rfloor_n))] = [\hat\sigma(v_1),\ldots,\hat\sigma(v_n)]\end{aligned}$$

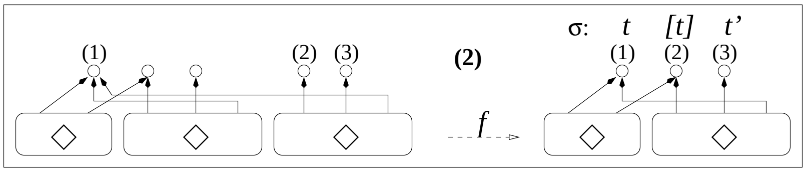
Lemma B implies that $T = f(\hat T) = f(t(P))$ is defined and that there is a mapping σ consistent with T such that $\sigma(\lfloor\chi_T\rfloor_i) = \hat\sigma(\lfloor\chi_{\hat T}\rfloor_i)$. Thus it holds that $\sigma(\lfloor\chi_T\rfloor_i) = \hat\sigma(\lfloor\chi_{\hat T}\rfloor_i) = \hat\sigma(\phi(\lfloor\chi_D\rfloor_i)) = \Delta(\mu(\lfloor\chi_D\rfloor_i)) = \Delta(\lfloor\tilde t\rfloor_i)$.

Example: as an example, look at the process graph P depicted in Figure (1) below. It consists of two edges, both able to replicate themselves, where the edge on the left-hand side waits for incoming messages on its first and only node. Each message should be equipped with two nodes, to the first of which another message is sent. The edge on the right-hand side produces arbitrarily many messages to be received by the edge on the left-hand side. Note that this process graph has an infinite reduction sequence (not even counting the replication steps) and if we denote the innermost process graph of arity 3 in the left-hand side edge by P', then $P \to^* P\Box P'\Box\ldots\Box P'$.



If we set $\tilde t = abc$, its π-calculus counterpart $\Theta_{\tilde t}(P) = !\,a(x,y).\overline{x}\langle a\rangle \mid !\,\overline{a}\langle b,c\rangle = p$, which reduces to $!\,a(x,y).\overline{x}\langle a\rangle \mid !\,\overline{a}\langle b,c\rangle \mid \overline{b}\langle c\rangle \mid \ldots \mid \overline{b}\langle c\rangle$. The process p can be typed under the type assignment $\Gamma = a : t, b : [t], c : t'$ where t' is an arbitrary type tree and t is the solution of the fixed-point equation $t = [[t],t']$. Note that the infinity of the tree is not caused by replication, but rather by the fact that the left-hand side process emits its own name as the content of a message.

Now, computing $t(P)$ yields the type graph depicted in Figure (D) below, where the edge in the middle is generated by applying t to the process abstraction $[(1,2)Q]_1$, and the other two edges are generated by $t([M]_2)$ respectively $t([M]_3)$. Computing $f(t(P))$ fuses the two rightmost edges. We indicate a consistent mapping σ by mapping the nodes to appropriate type trees. This consistent mapping corresponds exactly to the type assignment Γ given above.



4.2 Concurrent Object-Oriented Programming

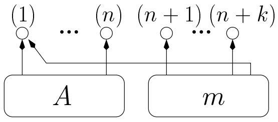
We now show how to model a concurrent object-oriented system by graph rewriting and then present a type system. In our model, several objects may compete in order to receive a message, and several messages might be waiting at the same object. Typically, type systems in object-oriented programming are there to ensure that an object that receives a message is able to process it.

Definition 8. (Concurrent object-oriented rewrite system) *Let $(\mathcal{C}, <:)$ be a lattice of classes with a top class*[6] $\top$ *and a bottom class* $\bot$. *We denote classes by the letters* $A, B, C, \dots$ *Furthermore let* $\mathcal{M}$ *be a set of method names. The function* $ar : \mathcal{C} \cup \mathcal{M} \to \mathbb{N} \backslash \{0\}$ *assigns an arity to every class or method name.*

An object graph G is a hypergraph with a duplicate-free string of external nodes, labelled with elements of $\mathcal{C}\backslash\{\bot\} \cup \mathcal{M}$ where for every edge e it holds that $ar(e) = ar(l_G(e))$. A concurrent object-oriented rewrite system (specifying the semantics) consists of a set of rules $\mathcal{R}$ satisfying the following conditions:

- *the left-hand side of a rule always has the form shown in Figure (C) below (where $A \in \mathcal{C}\backslash\{\bot\}$, $ar(A) = n$, $m \in \mathcal{M}$, $ar(m) = k+1$).*

  The right-hand side is again an object graph of arity $n+k$. If a left-hand side $R_{A,m}$ exists, we say that A understands m.



(C) $= R_{A,m}$

- *If $A <: B$, $A \neq \bot$ and B understands m, then A also understands m.*
- *For all $m \in \mathcal{M}$, either $\{A \mid A \text{ understands } m\}$ is empty or it contains a greatest element.*

An object graph G contains a "message not understood"-error if G contains a subgraph $R_{A,m}$, but A does not understand m.

Thus the predicate X for this section is defined as follows: $X(G)$ if and only if G does not contain a "message not understood"-error.

In contrast to the previous section, we now use annotated type graphs: the annotation mapping $\mathcal{A}$ assigns a lattice $(\{a : V_H \to \mathcal{C} \times \mathcal{C}\}, \leq))$ to every hypergraph H. The partial order is defined as follows: $a_1 \leq a_2 \iff \forall v\colon (a_1(v) = (A_1, B_2) \wedge a_2(v) = (A_2, B_2) \Rightarrow A_1 <: A_2 \wedge B_1 :> B_2)$, i.e. we have covariance in the first and contravariance in the second position. If a node v is labelled (A, B), this has the following intuitive meaning: we can accept at least as many messages as an object of class A on this node *and* we can send at most as many messages as an object of class B can accept.

Furthermore we define $\mathcal{A}_\phi(a)(v') = \bigvee_{\phi(v)=v'} a(v)$ where $\phi : H \to H'$, a is an element of $\mathcal{A}(H)$ and $v' \in V_{H'}$.

We now define the operator f: let $T[a]$ be a type graph of arity n where it holds for all nodes v that $a(v) = (A, B)$ implies $A <: B$ (otherwise f is undefined). Then f reduces the graph to its string of external nodes, i.e $f(T[a]) = \mathbf{n}[b]$ where $b(\lfloor\chi_{\mathbf{n}}\rfloor_i) = a(\lfloor\chi_T\rfloor_i)$. We accept a type graph, if it is defined, i.e. if f is successful. So we define $Y(T[a]) = \mathit{true}$ for all type graphs T.

The linear mapping t determines the type of a class or method. It is necessary to choose a linear mapping that preserves the interface of left-hand and right-hand sides, i.e. we can use any t that satisfies condition (10) and the following two conditions below for $A \in \mathcal{C}\backslash\{\bot\}$ and $m \in \mathcal{M}$:

$$t([A]_n) = [A]_n[a] \text{ where } a(\lfloor\chi_{[A]_n}\rfloor_1) \geq (A, \top)$$
$$t([m]_n) = [m]_n[a] \text{ where } a(\lfloor\chi_{[m]_n}\rfloor_n) \geq (\bot, \max\{B \mid B \text{ understands } m\})$$

Proposition 6. *The annotation mapping $\mathcal{A}$, the mappings f and t and the predicates X and Y defined above satisfy conditions (6)–(10) of Theorem 1. Thus if $G \triangleright T$ and $Y(G)$, then G will never produce a "message not understood"-error during reduction.*

Proof. We show that $\mathcal{A}, f, t, Y, X$ satisfy conditions (6)–(10) of Theorem 1.

(6) This property does trivially hold since $Y(T[a]) = \mathit{true}$ for all T.

(8) Let $\phi : T[a] \twoheadrightarrow_{\mathcal{A}} T'[a']$. It holds that $f(T[a]) = \mathbf{n}[b]$ and $f(T'[a']) = \mathbf{n}[b']$ where $n = ar(T) = ar(T')$, $b(\lfloor\chi_{\mathbf{n}}\rfloor_i) = a(\lfloor\chi_T\rfloor_i)$ and $b'(\lfloor\chi_{\mathbf{n}}\rfloor_i) = a'(\lfloor\chi_{T'}\rfloor_i)$. There is trivially a strong morphism from $\mathbf{n}$ into $\mathbf{n}$ (the identity) and furthermore $b'(\lfloor\chi_{\mathbf{n}}\rfloor_i) = a'(\lfloor\chi_{T'}\rfloor_i) \geq \mathcal{A}_\phi(a)(\lfloor\chi_{T'}\rfloor_i) = \bigvee_{\phi(v)=\lfloor\chi_{T'}\rfloor_i} a(v) \geq a(\lfloor\chi_T\rfloor_i) = b(\lfloor\chi_{\mathbf{n}}\rfloor_i)$.
And thus $f(T[a]) \twoheadrightarrow_{\mathcal{A}} f(T'[a'])$.

[6] This corresponds to the class Object in Java

(7) let $T_i[a_i]$ be type graphs and we set

$$T = \textcircled{\scriptsize D}_{i=1}^{n}(T_i[a_i], \zeta_i), \quad \overline{T_i}[\overline{a_i}] = f(T_i[a_i]), \quad \overline{T}[\overline{a}] = \textcircled{\scriptsize D}_{i=1}^{n}(\overline{T_i}[\overline{a_i}], \zeta_i)$$

We prove that $f(T) \cong f(\overline{T})$.
It is clear from the definition of f that $f(T) \twoheadrightarrow_{\mathcal{A}} T$, if $f(T)$ is defined. So we have the situation depicted in the figure below

$$\begin{tikzcd}
\mathbf{m_i} \arrow[r, "\zeta_i"] \arrow[d, "\overline{\phi}_i"'] \arrow[dr, phantom, "\text{(a)}"] & D \arrow[d, "\overline{\phi}"] \arrow[dd, bend left, "\phi"] & \\
\overline{T_i}[\overline{a_i}] \arrow[r, "\eta_i'"] \arrow[d, "\psi_i"'] \arrow[dr, phantom, "\text{(b)}"] & \overline{T}[\overline{a}] \arrow[d, dashed, "\psi"] & f(\overline{T}[\overline{a}]) = \mathbf{n}[\overline{b}] \arrow[l, two heads] \\
T_i[a_i] \arrow[r, "\eta_i"] & T[a] & f(T[a]) = \mathbf{n}[b] \arrow[l, two heads]
\end{tikzcd}$$

(a) is a colimit, and the $\psi_i \circ \phi_i$ are the unique strong morphisms from $\mathbf{m_i}$ into T_i. We assume that the η_i and ϕ are the colimit of the ζ_i and the $\psi_i \circ \phi_i$. It follows from standard properties of colimits that there is a morphism $\psi : \overline{T}[\overline{a}] \twoheadrightarrow_{\mathcal{A}} T[a]$ such that $\psi \circ \overline{\phi} = \phi$ and (b) is also a colimit.
We first show that $a(\psi(v)) = \overline{a}(v)$ holds for all $v \in V_{\overline{T}}$: note that whenever $\psi(v) = \eta_i(v')$ for any $v' \in V_{T_i}$, then it follows from the definition of graph construction that v' is an external node of T_i and therefore $a_j(v') = \bigvee_{\psi_j(v'')=v'} \overline{a_j}(v'')$, where all $\overline{a_j}(v'')$ are equal.

$$\begin{aligned}
a(\psi(v)) &= \bigvee_{j=1}^{n} \bigvee_{\eta_j(v')=\psi(v)} a_j(v') = \bigvee_{j=1}^{n} \bigvee_{\eta_j(v')=\psi(v)} \bigvee_{\psi_j(v'')=v'} \overline{a_j}(v'') \\
&= \bigvee_{j=1}^{n} \bigvee_{\eta_j(\psi_j(v''))=\psi(v)} \overline{a_j}(v'') = \bigvee_{j=1}^{n} \bigvee_{\psi(\eta_j'(v''))=\psi(v)} \overline{a_j}(v'') \\
&\overset{\psi \text{ inj.}}{=} \bigvee_{j=1}^{n} \bigvee_{\eta_j'(v'')=v} \overline{a_j}(v'') = \overline{a}(v)
\end{aligned}$$

In the next step we prove that $f(T[a])$ is defined, i.e. $T[a]$ satisfies condition (12), if and only if $f(\overline{T}[\overline{a}])$ is defined.

$$\forall\, v \in V_T \colon (a(v) = (A, B) \;\Rightarrow\; A <: B) \tag{12}$$

We first assume that $T[a]$ satisfies (12). Since $\overline{T}[\overline{a}] \twoheadrightarrow_{\mathcal{A}} T[a]$ and (12) is preserved by inverse morphisms, it follows that $f(T[a])$ is also defined.
Now let $f(\overline{T}[\overline{a}])$ be defined, i.e. the $f(T_i[a_i]) = \overline{T_i}[\overline{a_i}]$ are defined and $\overline{T}[\overline{a}]$ satisfies (12). Let $v \in V_T$. We distinguish two cases:

- v is not in the range of ϕ: in this case there is a unique i and a unique node $v' \in V_{T_i}$ such that $\eta_i(v') = v$ and $a_i(v') = a(v)$. Therefore $a(v) = (A, B)$ implies $A <: B$ since $T_i[a_i]$ satisfies condition (12).
- v is in the range of ϕ: in this case there is a node $v' \in V_{\overline{T}}$ such that $v = \psi(v')$ and (see above) $a(v) = a(\psi(v')) = \overline{a}(v')$. Since $\overline{T}[\overline{a}]$ satisfies (12) it holds that $a(v) = (A, B)$ implies $A <: B$.

It is left to show that $f(\overline{T}[\overline{a}]) = \mathbf{n}[\overline{b}]$ and $f(T[a]) = \mathbf{n}[b]$ are isomorphic. The underlying graphs are clearly isomorphic and it is left to prove that $\overline{b} = b$:

$$b(\lfloor \chi_{\mathbf{n}} \rfloor_i) = a(\lfloor \chi_T \rfloor_i) = a(\psi(\lfloor \chi_{\overline{T}} \rfloor_i)) = \overline{a}(\lfloor \chi_{\overline{T}} \rfloor_i) = \overline{b}(\lfloor \chi_{\mathbf{n}} \rfloor_i)$$

(9) we assume that $Y(f(t(G)))$ holds, which implies that $f(t(G))$ is defined, or—in other words—if $T[a] = t(G)$, $v \in V_T$ and $a(v) = (A, B)$, then it holds that $A <: B$.
We now assume that G has a subgraph $R_{A,m}$ and we show that A understands m. It follows that there is a morphism $\eta : t(R_{A,m}) \to_{\mathcal{A}} t(G)$. We set $\overline{T}[\overline{a}] = t(R_{A,m})$.
From the conditions imposed on t it follows that

$$\overline{a}(\lfloor \chi_{\overline{T}} \rfloor_1) \geq (A, \bot) \quad \text{and} \quad \overline{a}(\lfloor \chi_{\overline{T}} \rfloor_1) \geq (\bot, \max\{B \mid B \text{ understands } m\})$$

Now let $\overline{a}(\lfloor\chi_{\overline{T}}\rfloor_1) = (C, D)$ and $a(\eta(\lfloor\chi_{\overline{T}}\rfloor_1)) = (C', D')$. Since η is an $\mathcal{A}$-morphism, it holds that $(C, D) \le (C', D')$ and therefore $C <: C'$ and $D :> D'$. Furthermore

$$\bot \neq A <: C <: C' <: D' <: D <: \max\{B \mid B \text{ understands } m\}$$

Then it follows from Definition 8 that A understands m.

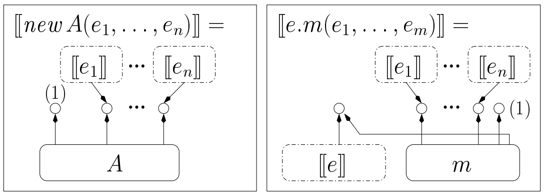
(10) this is satisfied because of the condition imposed on t.

In this case we do not prove that this type systems corresponds to an object-oriented type system, but rather present a semi-formal argument: we give the syntax and a type system for a small object calculus, and furthermore an encoding into hypergraphs, without really defining the semantics. For the formal semantics of object calculi see [20, 9], among others.

An expression e in the object calculus either has the form $new\ A(e_1, \ldots, e_n)$ where $A \in \mathcal{C}\backslash\{\bot\}$ and $ar(A) = n + 1$ or $e.m(e_1, \ldots, e_n)$ where $m \in \mathcal{M}$ and $ar(m) = n + 2$. The e_i are again expressions. Every class A is assigned an $(ar(A) - 1)$-tuple of classes defining the type of the fields of A $(A : (A_1, \ldots, A_n))$ and every method m with $ar(m) = n + 2$ defined in class B is assigned a type $B.m : C_1, \ldots, C_n \to C$. If a method is overwritten in a subclass it is required to have the same type. A simple type systems looks as follows:

$$\frac{e : A,\ A <: B}{e : B} \quad \frac{A : (A_1, \ldots, A_n),\ e_i : A_i}{new\ A(e_1, \ldots, e_n) : A} \quad \frac{e : B,\ B.m : C_1, \ldots, C_n \to C,\ e_i : C_i}{e.m(e_1, \ldots, e_n) : C}$$

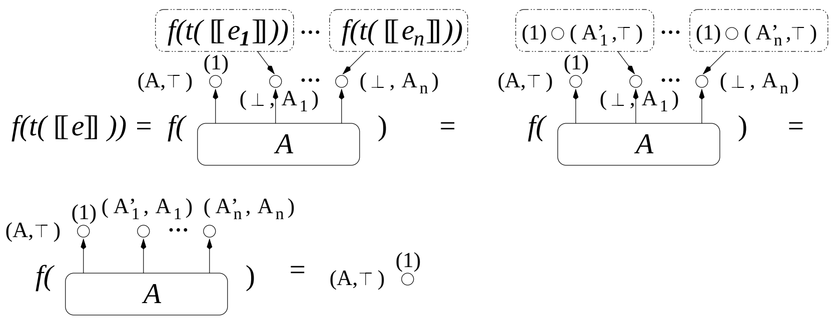
Now an encoding $[\![\cdot]\!]$ can be defined as shown in the figure to the right. We introduce the convention that the penultimate node of a message can be used to access the result after the rewriting step.



If $A : (A_1, \ldots, A_n)$ we define t in such a way that the $n+1$ external nodes of $t([A]_{n+1})$ are annotated by $(A, \top)$, $(\bot, A_1)$, ..., $(\bot, A_n)$. And if $B.m : C_1, \ldots, C_n \to C$ (where B is the maximal class which understands method m), we annotate the external nodes of $t([m]_{n+2})$ by $(\bot, C_1)$, ..., $(\bot, C_n)$, $(C, \top)$, $(\bot, B)$. Now we can show by induction on the typing rules that if $e : A$, then there exists a type graph $T[a]$ such that $[\![e]\!] \rhd T[a]$ and $a(\lfloor\chi_T\rfloor_1) = (A, \top)$.

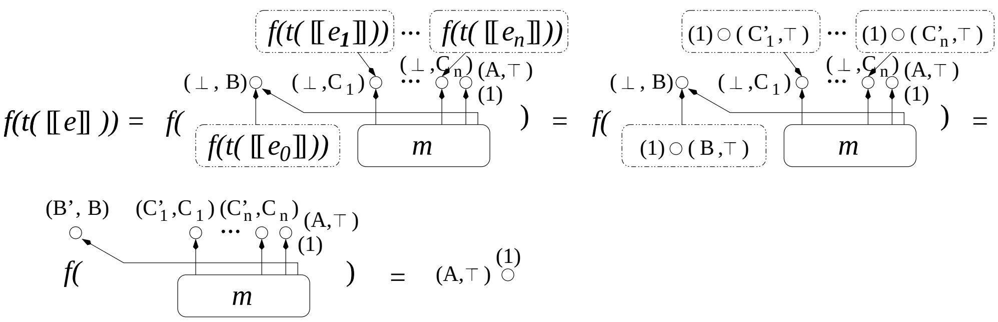
Proof: we show a stronger induction hypothesis. If $e : A$, then $f(t([\![e]\!]))$ is defined and $f(t([\![e]\!])) = \mathbf{1}[a]$ where $a(\lfloor\chi_{\mathbf{1}}\rfloor_1) = (A', \top) \le (A, \top)$. With the weakening rule (3) we then obtain the original claim.

- Let $e : B$ where $e : A$ and $A <: B$. With the induction hypothesis it follows that $f(t([\![e]\!])) = \mathbf{1}[a]$ where $a(\lfloor\chi_{\mathbf{1}}\rfloor_1) = (B', \top) \le (B, \top)$. And since $B <: A$, we also obtain $(B', \top) \le (A, \top)$.
- Let $e = new\ A(e_1, \ldots, e_n) : A$ where $A : (A_1, \ldots, A_n)$ and $e_i : A_i$. From the induction hypothesis it follows that $f(t([\![e_i]\!])) = \mathbf{1}[a_i]$ is defined and that $a_i(\lfloor\chi_{\mathbf{1}}\rfloor_1) = (A'_i, \top) \le (A_i, \top)$. Now we can conclude that



The last step is defined since $A'_i <: A_i$.

– Let $e = e_0.m(e_1, \ldots, e_n) : A$ where $e_0 : B$, $B.m : C_1, \ldots, C_n \to A$ and $e_i : C_i$. From the induction hypothesis it follows that $f(t([\![e_i]\!]))$, $0 \le i \le n$ is defined and that $f(t([\![e_i]\!])) = \mathbf{1}[a_i]$ with $a_0(\lfloor \chi_{T_0} \rfloor_1) = (B', \top) \le (B, \top)$ and $a_i(\lfloor \chi_{T_i} \rfloor_1) = (C_i', \top) \le (C_i, \top)$ for $1 \le i \le n$.
So we can conclude that:



The last step is defined since $B' <: B$ and $C_i' <: C_i$.

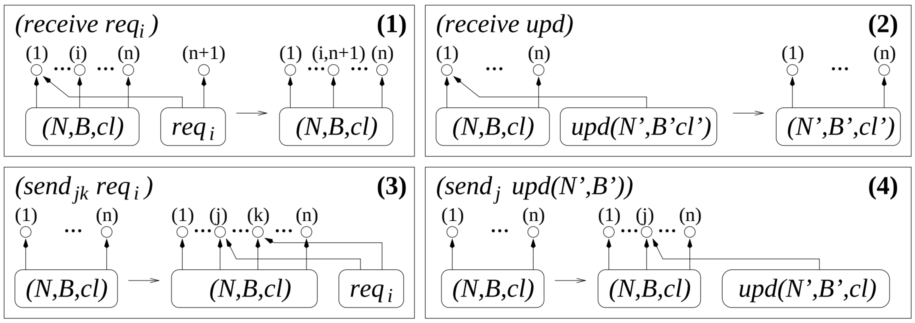
4.3 A Security Policy for Untrustworthy Applets

In this last case study we show how to construct a new type system rather than simulate an existing one. We have the following scenario: an applet is received and is executed in an environment containing trustworthy and untrustworthy objects. By definition, all objects in the applet are untrustworthy. Objects may interact with each other and modify each other. An untrustworthy object must not modify a trustworthy one.

In order to achieve this, two solutions are applicable: the first solution is that every object has to authenticate itself by some sort of protocol during runtime, which can be rather costly. The second solution is that, after reception of the applet, a static type check is performed which excludes the possibility of trust violation for at least a subset of trustworthy objects. For these objects there is no need to conduct the authentification protocol. Since typing is compositional, it is sufficient to compute the type of the environment only once. Thus, if a rather small applet is received, the computation of its type and the composition of the two types is cheap.

The system model is as follows: both, environment and applet are represented by hypergraphs and can be combined by some suitable construction operation. There are three forms of edges: objects, request messages and update messages.

Each object (represented by an edge) has a unique name $N \in \mathcal{N}$, a behaviour $B \in \mathcal{B}$, a clearance $cl \in \{t, u\}$ indicating if the object is trustworthy or untrustworthy and a fixed arity $n \in \mathbb{N}$. Objects may spontaneously send request or update messages. A request message is an edge of arity 2 which is labelled req_i, meaning that the i-th field of an object is requested and should be merged with the second node of the message. An update message is an edge of arity 1 which is labelled $upd(N, B, cl)$, meaning that the receiving object should change its name to N, its behaviour to B and its clearance to cl. According to its behaviour B, an object only sends and accepts a subset of all messages, so $\mathcal{R}_B$ (the set of all rewrite rules for behaviour B) is a subset of the rules depicted in the figure below.



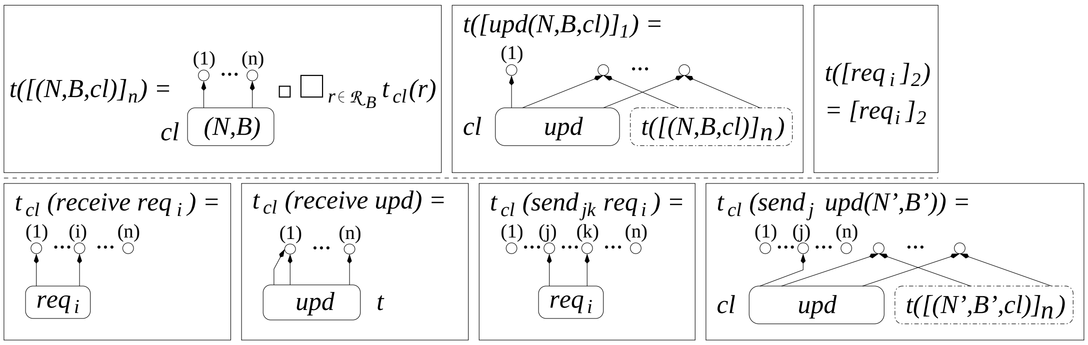
The clearance labels are only there for clarification of the problem, but they are not yet the solution. In practice nothing would keep an object from sending a message with a faked label.

We say that B emits B' if the rule $(send_j\ upd(N', B', cl'))$ is contained in $\mathcal{R}_B$. For technical reasons we demand that the "emits"-relation does not have cycles.

The set $\mathcal{R}$ of all rewrite rules is the union of the $\mathcal{R}_B$. We want to make sure that for a certain object of name N no program will ever contain a subgraph in the form of the left-hand side of rule (2) where $cl = t$ and $cl' = u$. We call this a trust violation for N. The predicate X_N holds for a program P, if P does not contain a trust violation for N.

We can regard the set $\{t, u\}$ as a lattice with elements t and u where $t < u$. Then our type graphs have the following form: edges are labelled either by $(N, B) \in \mathcal{N} \times \mathcal{B}$ (then they are annotated by an element from $\{t, u\}$) or by req_i (trivial annotations) or by upd (annotations from $\{t, u\}$). Let a be an annotation assigning lattice elements to the respective edges. We define $\mathcal{A}_\phi(a)(e') = \bigvee_{\phi(e)=e'} a(e)$ if $\phi : H \to H'$ and $e' \in E_{H'}$.

The linear mapping t—which anticipates all components that might be introduced into the graph during reduction and creates information for f on how they should be merged—is defined as depicted in the figure below. We define $\square_{1 \le i \le n} T_i = T_1 \square \ldots \square T_n$. The image of an object edge is defined via the auxiliary mapping t_{cl} which takes into account all the rules which may be applied to the object edge. Because of the condition on the "emits"-relation, the construction of t terminates and is well-defined.



The operation f can now be defined according to Proposition 2 where an annotated hypergraph $T[a]$ satisfies property C if and only if it holds for all $e_1, e_2 \in E_T$ that

$$l_T(e_1) = l_T(e_2) \in \{upd\} \cup \bigcup_{i \in \mathbb{N}} \{req_i\} \ \wedge\ \lfloor s_T(e_1) \rfloor_1 = \lfloor s_T(e_2) \rfloor_1 \ \Rightarrow\ e_1 = e_2$$

In other words: if two message edges are sent to the same node, they are merged. This is basically equivalent to the condition C imposed in section 4.1.

It is left to define a predicate Y_N in order to infer the absence of trust violation for a trustworthy object of name N: let $T[a]$ be a type graph. We say that $Y_N(T[a])$ holds if no untrustworthy update message is attached to an edge labelled (N, B) for any behaviour B:

$$\begin{aligned} \forall e, e' \in E_T\colon\ ((\exists B : l_T(e) = (N, B)) \wedge l_T(e') = upd \wedge \lfloor s_T(e') \rfloor_1 = \lfloor s_T(e) \rfloor_1 \\ \Rightarrow\ a(e') = t) \end{aligned}$$

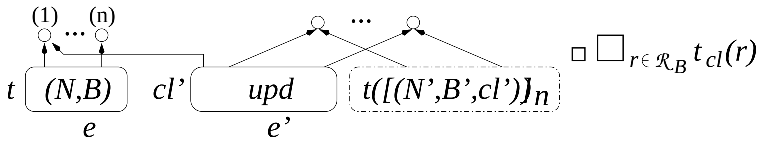
Proposition 7. *The annotation mapping $\mathcal{A}$, the mappings f and t and the predicates X_N and Y_N defined above satisfy conditions (6)–(10) of Theorem 1. Thus if $P \triangleright T$ and $Y_N(P)$, then there will never be a trust violation for an object of name N in P.*

Proof. We show that $\mathcal{A}, f, t, X, Y$ satisfy the conditions of Theorem 1.

(6) We show that Y_N is preserved by invariant morphisms: let $\phi : T[a] \to_{\mathcal{A}} T'[a']$ and $Y_N(T'[a'])$. Now let $e, e' \in E_T$ such that there is a behaviour description B with $l_T(e) = (N, B)$, furthermore $l_T(e') = upd$ and $\lfloor s_T(e') \rfloor_1 = \lfloor s_T(e) \rfloor_1$. Since labels and the connection function are preserved by morphisms, it follows that $l_{T'}(\phi(e)) = (N, B)$, $l_{T'}(\phi(e')) = upd$ and $\lfloor s_{T'}(\phi(e')) \rfloor_1 = \lfloor s_{T'}(\phi(e)) \rfloor_1$. Since $Y_N(T'[a'])$ holds, we conclude that $a'(\phi(e')) = t$.
Since ϕ is an $\mathcal{A}$-morphism it follows that $a(e) \le a'(\phi(e)) = t$ and therefore $a(e') = t$.

(7) & (8) Showing that f satisfies the universal property (as defined in Proposition 2) with respect to condition C is analogous to proving the same fact for process graphs (see the proof of Proposition 3).

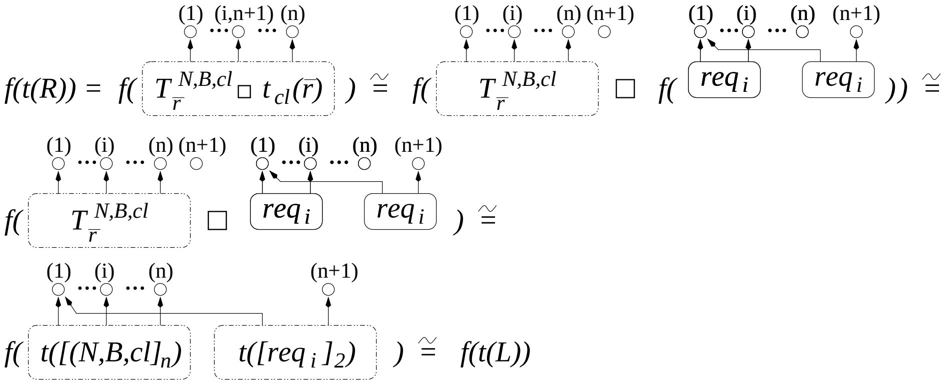
(9) Let $Y_N(f(t(H)))$. We show that this implies $X_N(H)$: let us assume that H contains a subgraph corresponding to the left-hand side L of rule (2) where $cl = t$. Then there is an $\mathcal{A}$-morphism from $f(t(L))$ into $f(t(H))$ and since Y_N is preserved by inverse $\mathcal{A}$-morphisms, it follows that $Y_N(f(t(L)))$ and also $Y_N(t(L))$ hold. The type graph $t(L)$ has the following form:



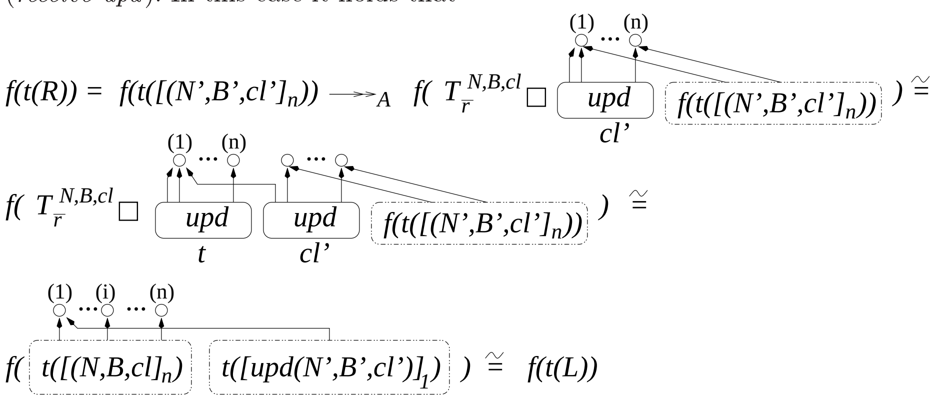
And since Y_N is satisfied, it follows that $cl' = t$.

(10) We show that this property holds for all four types of rules. For a rule $\overline{r}$, we set $T_{\overline{r}}^{N,B,cl} = [(N,B)]_n[a] \,\square \bigsqcup_{r\in\mathcal{R}_B\setminus\{r\}} t_{cl}(r)$ where $a(e) = cl$ for the only edge e of $[(N,B)]_n$. That is $T_{\overline{r}}^{N,B,cl}$ is $t([(N,B,cl)]_n)$ without $t_{cl}(\overline{r})$, or $t([(N,B,cl)]_n) = T_{\overline{r}}^{N,B,cl} \square\, t_{cl}(\overline{r})$.

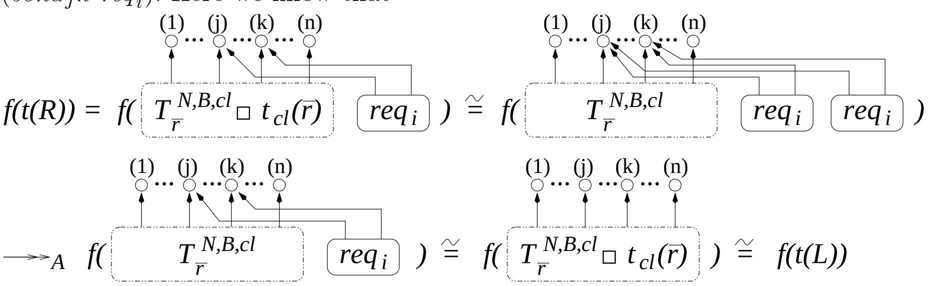
(1) $\overline{r} = (receive\ req_i)$. In this case we conclude



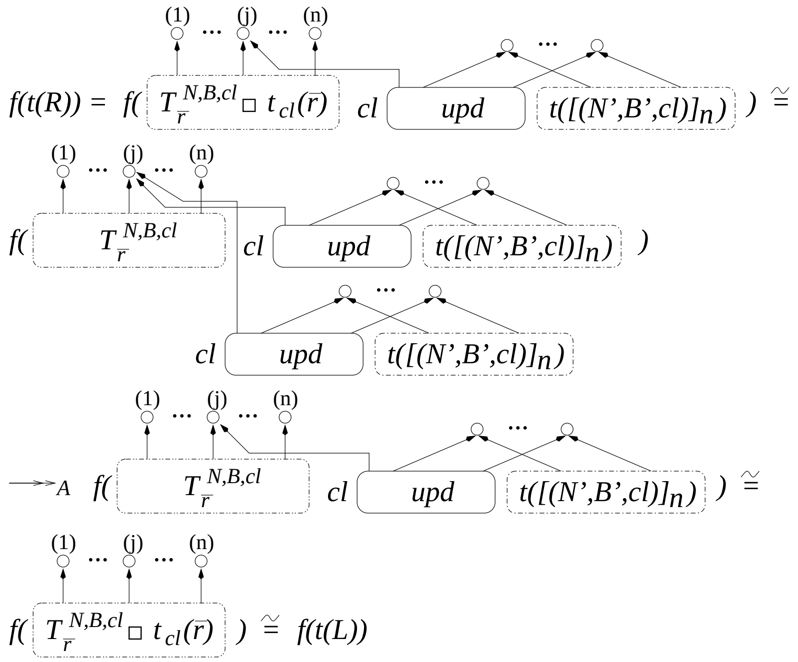
(2) $\overline{r} = (receive\ upd)$. In this case it holds that



(3) $\overline{r} = (send_j k\ req_i)$. Here we know that

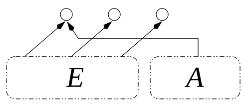


(4) $\overline{r} = (send_j\ upd(N',B'))$ and in this case we can conclude that

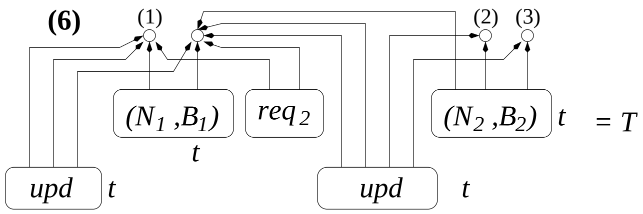


Example: we assume that there is an environment E into which an applet A should be plugged, and the combination P of environment and applet is obtained as shown in Figure (5) below. We assume that E has already been typed and that $E \triangleright T$ where T is depicted in Figure (6). Although it is normally not possible to reconstruct the original graph from a type, we can speculate how this type could have originated: for example, E has two objects labelled with (N_1, B_1, t) respectively (N_2, B_2, t) where the first accepts request and update messages while the second only accepts update messages.

(5)

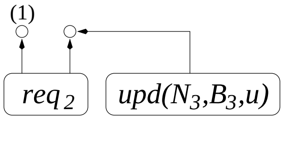
$P =$ 

(6)

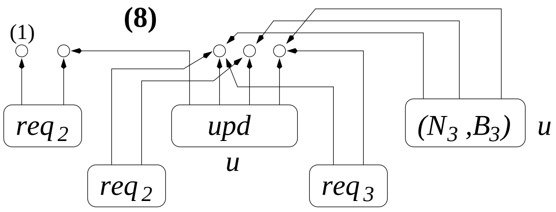


Now assume that we receive an applet A from outside, which has the form drawn in Figure (7). We have to make sure that all objects and update messages have the label “untrustworthy”, which holds in this case. In order to type A, all we need to know is the rule set $\mathcal{R}_{B_3}$ and in this case we assume that it contains the rules $(receive\ req_2)$ and $(receive\ req_3)$.

(7)

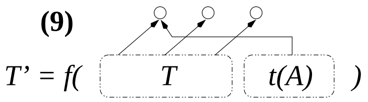


(8)



So, computing $t(A)$ yields the type graph drawn in Figure (8) and because of condition (4), the entire program P has a type T' as depicted in Figure (9). T' is rather complex and we will not draw the entire graph here, but we will describe its most important features: since the first external node of T and the first external node of $t(A)$ are merged, it follows that the two req_2-edges sent to these nodes are merged, which causes the first node of the untrusted update message in $t(A)$ and the first node of the edge labelled (N_2, B_2) to merge. Therefore $Y_{N_2}(T')$ does not hold, but $Y_{N_1}(T')$ holds. So we need an authentication protocol for the object with name N_2, but we can be sure that no untrusted update message is ever sent to N_1, although N_1 can accept update messages.

(9)



5 Conclusion and Comparison to Related Work

This is a first tentative approach aimed at developing a general framework for the static analysis of graph rewriting in the context of type systems. It is obvious that there are many type systems which do not fit well into our proposal. But since we are able to capture the essence of two important type systems, we assume to be on the right track.

Types are often used to make the connection of components and the flow of information through a system explicit (see e.g. the type system for the π-calculus, where the type trees indicate which tuple of channels is sent via which channel). Since connections are already explicit in graphs, we can use them both as type and as the expression to be typed. Via morphisms we can establish a clear connection between an expression and its type. Graphs are furthermore useful since we can easily add an extra layer of annotation (in our case: annotation by lattice elements).

Work that is very close in spirit to ours is [8] by Honda which also presents a general framework for type systems. The underlying model is closer to standard process algebras and the main focus is on the characterisation and classification of type systems.

The idea of composing graphs in such a way that they satisfy a certain property was already presented by Lafont in [14] where it is used to obtain deadlock-free nets.

In graph rewriting there already exists a concept of typed graphs [1], related to ours, but nevertheless different. In that work, 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, 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.

This paper is a continuation of the work presented in [10] where the idea of generic type systems for process graphs (as defined in section 4.1) was introduced, but no proof of the equivalence of our type system to the standard type system for the π-calculus was given. The ideas presented there are now extended to general graph rewriting systems.

Further work will consist in better understanding the underlying mechanism of the type system. An interesting question in this context is the following: given a set of rewrite rules, is it possible to automatically derive mappings f and t satisfying the conditions of Theorem 1? The crucial point here is the fact that cyclic dependencies may appear (i.e. there are two rules (L, R) and (L', R') where L and R' contain an edge labelled A and R and L' contain an edge labelled B), we avoided such a situation in section 4.3 by imposing an additional condition on the "emits"-relation, otherwise the definition of the linear mapping t would not have been well-founded. Is there some way to uniformly treat such situations? Probably some results concerning fixed-points are needed.

Acknowledgements: I would like to thank Reiko Heckel and Andrea Corradini for their comments on drafts of this paper, and Tobias Nipkow for his advice. I am also grateful to the anonymous referees for their valuable comments.

Remark: this report is the extended version of [11].

References

1. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26(3/4):241–265, 1996.
2. 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.
3. H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol.3: Concurrency, Parallellism, and Distribution*. World Scientific, 1999.
4. F. Gadducci and U. Montanari. Comparing logics for rewriting: Rewriting logic, action calculi and tile logic. *Theoretical Computer Science*, 2000. to appear.
5. Philippa Gardner. Closed action calculi. *Theoretical Computer Science (in association with the conference on Mathematical Foundations in Programming Semantics)*, 1998.
6. Annegret Habel. *Hyperedge Replacement: Grammars and Languages*. Springer-Verlag, 1992. LNCS 643.
7. Masahito Hasegawa. *Models of Sharing Graphs (A Categorical Semantics of Let and Letrec)*. PhD thesis, University of Edingburgh, 1997. available in Springer Distinguished Dissertation Series.
8. Kohei Honda. Composing processes. In *Proc. of POPL'96*, pages 344–357. ACM, 1996.
9. Atsushi Igarashi, Benjamin Pierce, and Philip Wadler. Featherweight Java: A core calculus for Java and GJ. In *Proc. of OOPSLA 1999*, 1999.
10. Barbara König. Generating type systems for process graphs. In *Proc. of CONCUR '99*, pages 352–367. Springer-Verlag, 1999. LNCS 1664.
11. Barbara König. A general framework for types in graph rewriting. In *Proc. of FST&TCS 2000*. Springer-Verlag, 2000. to appear.
12. Barbara König. A graph rewriting semantics for the polyadic pi-calculus. In *Workshop on Graph Transformation and Visual Modeling Techniques (Geneva, Switzerland), ICALP Workshops 2000*, pages 451–458. Carleton Scientific, 2000.
13. Barbara König. Hypergraph construction and its application to the compositional modelling of concurrency. In *GRATRA 2000: Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems*, 2000. Proceedings available as Report Nr. 2000-2 (Technische Universität Berlin).
14. Yves Lafont. Interaction nets. In *Proc. of POPL '90*, pages 95–108. ACM Press, 1990.
15. James J. Leifer and Robin Milner. Deriving bisimulation congruences for reactive systems. In *Proc. of CONCUR 2000*, 2000. LNCS 1877.
16. José Meseguer. Rewriting logic as a semantic framework for concurrency: A progress report. In *Concurrency Theory*, pages 331–372. Springer-Verlag, 1996. LNCS 1119.
17. 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.
18. Robin Milner. Calculi for interaction. *Acta Informatica*, 33(8):707–737, 1996.
19. Benjamin Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. In *Proc. of LICS '93*, pages 376–385, 1993.
20. David Walker. Objects in the π-calculus. *Information and Computation*, 116:253–271, 1995.
21. Nobuko Yoshida. Graph notation for concurrent combinators. In *Proc. of TPPP '94*. Springer-Verlag, 1994. LNCS 907.

A Semantics of the Asynchronous Polyadic π-Calculus

In section 4.1 we have given an alternative semantics for the asynchronous polyadic π-calculus in terms of process graphs. We have also presented an encoding from process graphs into the π-calculus (more details can be found in [12]). Here we give the syntax and semantics for this variant of the π-calculus, where we use the semantics given for its synchronous version in [19].

The asynchronous polyadic π-calculus without choice and matching can be described by the following syntax where we assume that $\mathcal{N}$ is a fixed set of names, $c \in \mathcal{N}$ and $\tilde{a}, \tilde{x} \in \mathcal{N}^*$:

$$\begin{array}{lll} p := \mathbf{0} & & \text{(nil process)} \\ & \mid (\nu c)p & \text{(restriction)} \\ & \mid \overline{c}\langle \tilde{a} \rangle & \text{(output)} \\ & \mid c(\tilde{x}).p & \text{(input)} \\ & \mid p_1 | p_2 & \text{(parallel composition)} \\ & \mid\ !p & \text{(replication)} \\ & \mid \mathit{wrong} & \text{(error)} \end{array}$$

The operational semantics of the π-calculus is defined as follows: structural congruence $\equiv$ is the smallest congruence closed under renaming of bound names (α-conversion) and under the rules given in the table below. The reduction relation $\rightarrow$ is generated by the rules listed below. By $p\{\tilde{a}/\tilde{x}\}$ we denote the substitution of the names $\lfloor \tilde{x} \rfloor_i$ by $\lfloor \tilde{a} \rfloor_i$ in p (with possible α-conversion in order to avoid capture).

Rules of Structural Congruence:

$$p_1|p_2 \equiv p_2|p_1 \qquad p_1|(p_2|p_3) \equiv (p_1|p_2)|p_3 \qquad (\nu c)\mathbf{0} \equiv \mathbf{0} \qquad (\nu c)(\nu b)p \equiv (\nu b)(\nu c)p$$

$$((\nu c)p_1)|p_2 \equiv (\nu c)(p_1|p_2) \text{ if } c \notin \mathit{fn}(p_2) \qquad p|\mathbf{0} \equiv p \qquad !p \equiv !p|p$$

$$!\,\mathit{wrong} \equiv \mathit{wrong} \qquad \mathit{wrong}|p \equiv \mathit{wrong} \qquad (\nu c)\,\mathit{wrong} \equiv \mathit{wrong}$$

Reduction Rules:

$$c(\tilde{x}).p \mid \overline{c}\langle \tilde{a} \rangle \rightarrow p\{\tilde{a}/\tilde{x}\} \quad \text{if } |\tilde{a}| = |\tilde{x}| \qquad c(\tilde{x}).p \mid \overline{c}\langle \tilde{a} \rangle \rightarrow \mathit{wrong} \quad \text{if } |\tilde{a}| \neq |\tilde{x}|$$

$$\frac{p \rightarrow p'}{p|q \rightarrow p'|q} \qquad \frac{p \rightarrow p'}{(\nu c)p \rightarrow (\nu c)p'} \qquad \frac{q \equiv p, p \rightarrow p', p' \equiv q'}{q \rightarrow q'}$$

Note that replication, which is part of the structural congruence, is simulated by two reduction rules in the case of process graphs. Therefore the operational correspondence in Proposition 4 does not hold for one-step reduction, but for the transitive closure $\rightarrow^*$ of the reduction relations.

With the π-calculus type system presented in section 4.1, the same kind of mismatching arities as denoted by the term “bad redex” (introduced in section 4.1) are avoided. So if $\Gamma \vdash p$, then $p \not\rightarrow^* \mathit{wrong}$.

SFB 342: Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

bisher erschienen :

Reihe A

Liste aller erschienenen Berichte von 1990-1994
auf besondere Anforderung

342/01/95 A Hans-Joachim Bungartz: Higher Order Finite Elements on Sparse Grids
342/02/95 A Tao Zhang, Seonglim Kang, Lester R. Lipsky: The Performance of Parallel Computers: Order Statistics and Amdahl's Law
342/03/95 A Lester R. Lipsky, Appie van de Liefvoort: Transformation of the Kronecker Product of Identical Servers to a Reduced Product Space
342/04/95 A Pierre Fiorini, Lester R. Lipsky, Wen-Jung Hsin, Appie van de Liefvoort: Auto-Correlation of Lag-k For Customers Departing From Semi-Markov Processes
342/05/95 A Sascha Hilgenfeldt, Robert Balder, Christoph Zenger: Sparse Grids: Applications to Multi-dimensional Schrödinger Problems
342/06/95 A Maximilian Fuchs: Formal Design of a Model-N Counter
342/07/95 A Hans-Joachim Bungartz, Stefan Schulte: Coupled Problems in Microsystem Technology
342/08/95 A Alexander Pfaffinger: Parallel Communication on Workstation Networks with Complex Topologies
342/09/95 A Ketil Stølen: Assumption/Commitment Rules for Data-flow Networks - with an Emphasis on Completeness
342/10/95 A Ketil Stølen, Max Fuchs: A Formal Method for Hardware/Software Co-Design
342/11/95 A Thomas Schnekenburger: The ALDY Load Distribution System
342/12/95 A Javier Esparza, Stefan Römer, Walter Vogler: An Improvement of McMillan's Unfolding Algorithm
342/13/95 A Stephan Melzer, Javier Esparza: Checking System Properties via Integer Programming
342/14/95 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Point-to-Point Dataflow Networks
342/15/95 A Andrei Kovalyov, Javier Esparza: A Polynomial Algorithm to Compute the Concurrency Relation of Free-Choice Signal Transition Graphs
342/16/95 A Bernhard Schätz, Katharina Spies: Formale Syntax zur logischen Kernsprache der Focus-Entwicklungsmethodik
342/17/95 A Georg Stellner: Using CoCheck on a Network of Workstations
342/18/95 A Arndt Bode, Thomas Ludwig, Vaidy Sunderam, Roland Wismüller: Workshop on PVM, MPI, Tools and Applications
342/19/95 A Thomas Schnekenburger: Integration of Load Distribution into ParMod-C
342/20/95 A Ketil Stølen: Refinement Principles Supporting the Transition from Asynchronous to Synchronous Communication
342/21/95 A Andreas Listl, Giannis Bozas: Performance Gains Using Subpages for Cache Coherency Control
342/22/95 A Volker Heun, Ernst W. Mayr: Embedding Graphs with Bounded Treewidth into Optimal Hypercubes

Reihe A

342/23/95 A Petr Jančar, Javier Esparza: Deciding Finiteness of Petri Nets up to Bisimulation

342/24/95 A M. Jung, U. Rüde: Implicit Extrapolation Methods for Variable Coefficient Problems

342/01/96 A Michael Griebel, Tilman Neunhoeffer, Hans Regler: Algebraic Multigrid Methods for the Solution of the Navier-Stokes Equations in Complicated Geometries

342/02/96 A Thomas Grauschopf, Michael Griebel, Hans Regler: Additive Multilevel-Preconditioners based on Bilinear Interpolation, Matrix Dependent Geometric Coarsening and Algebraic-Multigrid Coarsening for Second Order Elliptic PDEs

342/03/96 A Volker Heun, Ernst W. Mayr: Optimal Dynamic Edge-Disjoint Embeddings of Complete Binary Trees into Hypercubes

342/04/96 A Thomas Huckle: Efficient Computation of Sparse Approximate Inverses

342/05/96 A Thomas Ludwig, Roland Wismüller, Vaidy Sunderam, Arndt Bode: OMIS — On-line Monitoring Interface Specification

342/06/96 A Ekkart Kindler: A Compositional Partial Order Semantics for Petri Net Components

342/07/96 A Richard Mayr: Some Results on Basic Parallel Processes

342/08/96 A Ralph Radermacher, Frank Weimer: INSEL Syntax-Bericht

342/09/96 A P.P. Spies, C. Eckert, M. Lange, D. Marek, R. Radermacher, F. Weimer, H.-M. Windisch: Sprachkonzepte zur Konstruktion verteilter Systeme

342/10/96 A Stefan Lamberts, Thomas Ludwig, Christian Röder, Arndt Bode: PFSLib – A File System for Parallel Programming Environments

342/11/96 A Manfred Broy, Gheorghe Ştefănescu: The Algebra of Stream Processing Functions

342/12/96 A Javier Esparza: Reachability in Live and Safe Free-Choice Petri Nets is NP-complete

342/13/96 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Many-to-Many Data-flow Networks

342/14/96 A Giannis Bozas, Michael Jaedicke, Andreas Listl, Bernhard Mitschang, Angelika Reiser, Stephan Zimmermann: On Transforming a Sequential SQL-DBMS into a Parallel One: First Results and Experiences of the MIDAS Project

342/15/96 A Richard Mayr: A Tableau System for Model Checking Petri Nets with a Fragment of the Linear Time μ -Calculus

342/16/96 A Ursula Hinkel, Katharina Spies: Anleitung zur Spezifikation von mobilen, dynamischen Focus-Netzen

342/17/96 A Richard Mayr: Model Checking PA-Processes

342/18/96 A Michaela Huhn, Peter Niebert, Frank Wallner: Put your Model Checker on Diet: Verification on Local States

342/01/97 A Tobias Müller, Stefan Lamberts, Ursula Maier, Georg Stellner: Evaluierung der Leistungsfähigkeit eines ATM-Netzes mit parallelen Programmierbibliotheken

342/02/97 A Hans-Joachim Bungartz and Thomas Dornseifer: Sparse Grids: Recent Developments for Elliptic Partial Differential Equations

342/03/97 A Bernhard Mitschang: Technologie für Parallele Datenbanken - Bericht zum Workshop

342/04/97 A nicht erschienen

342/05/97 A Hans-Joachim Bungartz, Ralf Ebner, Stefan Schulte: Hierarchische Basen zur effizienten Kopplung substrukturierter Probleme der Strukturmechanik

Reihe A

Reihe A

342/04/98 A Stefan Bischof, Ernst W. Mayr: On-Line Scheduling of Parallel Jobs with Runtime Restrictions
342/05/98 A St. Bischof, R. Ebner, Th. Erlebach: Load Balancing for Problems with Good Bisectors and Applications in Finite Element Simulations: Worst-case Analysis and Practical Results
342/06/98 A Giannis Bozas, Susanne Kober: Logging and Crash Recovery in Shared-Disk Database Systems
342/07/98 A Markus Pizka: Distributed Virtual Address Space Management in the MoDiS-OS
342/08/98 A Niels Reimer: Strategien für ein verteiltes Last- und Ressourcenmanagement
342/09/98 A Javier Esparza, Editor: Proceedings of INFINITY'98
342/10/98 A Richard Mayr: Lossy Counter Machines
342/11/98 A Thomas Huckle: Matrix Multilevel Methods and Preconditioning
342/12/98 A Thomas Huckle: Approximate Sparsity Patterns for the Inverse of a Matrix and Preconditioning
342/13/98 A Antonin Kucera, Richard Mayr: Weak Bisimilarity with Infinite-State Systems can be Decided in Polynomial Time
342/01/99 A Antonin Kucera, Richard Mayr: Simulation Preorder on Simple Process Algebras
342/02/99 A Johann Schumann, Max Breitling: Formalisierung und Beweis einer Verfeinerung aus FOCUS mit automatischen Theorembeweisern – Fallstudie –
342/03/99 A M. Bader, M. Schimper, Chr. Zenger: Hierarchical Bases for the Indefinite Helmholtz Equation
342/04/99 A Frank Strobl, Alexander Wisspeintner: Specification of an Elevator Control System
342/05/99 A Ralf Ebner, Thomas Erlebach, Andreas Ganz, Claudia Gold, Clemens Harlfinger, Roland Wismüller: A Framework for Recording and Visualizing Event Traces in Parallel Systems with Load Balancing
342/06/99 A Michael Jaedicke, Bernhard Mitschang: The Multi-Operator Method: Integrating Algorithms for the Efficient and Parallel Evaluation of User-Defined Predicates into ORDBMS
342/07/99 A Max Breitling, Jan Philipps: Black Box Views of State Machines
342/08/99 A Clara Nippl, Stephan Zimmermann, Bernhard Mitschang: Design, Implementation and Evaluation of Data Rivers for Efficient Intra-Query Parallelism
342/09/99 A Robert Sandner, Michael Mauderer: Integrierte Beschreibung automatisierter Produktionsanlagen - eine Evaluierung praxisnaher Beschreibungstechniken
342/10/99 A Alexander Sabbah, Robert Sandner: Evaluation of Petri Net and Automata Based Description Techniques: An Industrial Case Study
342/01/00 A Javier Esparza, David Hansel, Peter Rossmanith, Stefan Schwoon: Efficient Algorithm for Model Checking Pushdown Systems
342/02/00 A Barbara König: Hypergraph Construction and Its Application to the Compositional Modelling of Concurrency
342/03/00 A Max Breitling and Jan Philipps: Verification Diagrams for Dataflow Properties
342/04/00 A Günther Rackl: Monitoring Globus Components with MIMO
342/05/00 A Barbara König: Analysing Input/Output Capabilities of Mobile Processes with a Generic Type System
342/06/00 A Michael Bader, Christoph Zenger: A Parallel Solver for Convection Diffusion Equations based on Nested Dissection with Incomplete Elimination

Reihe A

342/07/00 A Clara Nippl, Angelika Reiser, Bernhard Mitschang: Extending Database Functionality to Support Frequent Itemset Processing
342/08/00 A Clara Nippl, Angelika Reiser, Bernhard Mitschang: Conquering the Search Space for the Calculation of the Maximal Frequent Set
342/09/00A A Max Breitling, Jan Philipps: Transitions into Black Box Views

SFB 342 : Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Reihe B

342/1/90 B Wolfgang Reisig: Petri Nets and Algebraic Specifications
342/2/90 B Jörg Desel: On Abstraction of Nets
342/3/90 B Jörg Desel: Reduction and Design of Well-behaved Free-choice Systems
342/4/90 B Franz Abstreiter, Michael Friedrich, Hans-Jürgen Plewan: Das Werkzeug runtime zur Beobachtung verteilter und paralleler Programme
342/1/91 B Barbara Paech: Concurrency as a Modality
342/2/91 B Birgit Kandler, Markus Pawlowski: SAM: Eine Sortier- Toolbox - Anwenderbeschreibung
342/3/91 B Erwin Loibl, Hans Obermaier, Markus Pawlowski: 2. Workshop über Parallelisierung von Datenbanksystemen
342/4/91 B Werner Pohlmann: A Limitation of Distributed Simulation Methods
342/5/91 B Dominik Gomm, Ekkart Kindler: A Weakly Coherent Virtually Shared Memory Scheme: Formal Specification and Analysis
342/6/91 B Dominik Gomm, Ekkart Kindler: Causality Based Specification and Correctness Proof of a Virtually Shared Memory Scheme
342/7/91 B W. Reisig: Concurrent Temporal Logic
342/1/92 B Malte Grosse, Christian B. Suttner: A Parallel Algorithm for Set-of-Support
Christian B. Suttner: Parallel Computation of Multiple Sets-of-Support
342/2/92 B Arndt Bode, Hartmut Wedekind: Parallelrechner: Theorie, Hardware, Software, Anwendungen
342/1/93 B Max Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung
342/2/93 B Ekkart Kindler: Sicherheits- und Lebendigkeitseigenschaften: Ein Literaturüberblick
342/1/94 B Andreas Listl; Thomas Schnekenburger; Michael Friedrich: Zum Entwurf eines Prototypen für MIDAS