A Model of Mobile Networks with Synchronous and Asynchronous Operators

Tobias Mayr
Technische Universität München
mayrt@informatik.tu-muenchen.de

August 7, 1996

Contents

Chapter 1

Introduction

1.1 Motivation

This thesis is based upon an article written by Radu Grosu and Ketil Stølen [GS96]. They describe a denotational model for mobile networks whose components share channels. It models nondeterministic data-flow networks by sets of stream processing functions which are contractive with respect to the metric of streams. This view is based on the works of Kahn [Kah74], who describes static networks, of Park [Par83], of Broy [Bro87], and of Russell [Rus90]. Contrary to all these, Grosu and Stølen describe dynamically changing networks, i.e., the channel access of each component may change dependently on the received channel names.

There were three motivations for our further work on this model; first, we tried to formalize the theory within the theorem prover ISABELLE (see [Pau94a], [Pau94b]) using the object logic HOL (see [Pau94c]). The idea was to realize a development environment on which some of the design techniques of the project FOCUS (see [BDD+95]) could be based. There are already formalizations for that purpose, but none are using HOL and none are realizing mobility. This thesis can be seen as preparatory work for a future formalization of mobile networks in ISABELLE [1].

The second motivation stems from examinations of term algebras for mobile networks and their axiomatisation (compare [BŞ95]). Some of the considerations about renaming, name abstractness, and the operators where guided by these algebraic goals.

The last and most important inspiration was the search for a denotational model of the π-calculus of Milner ([MPW92a],[MPW92b]). We tried to make the theory apt to model synchronous communication as in the π-calculus. This was realized through the synchronous operations, the name abstractness principle, the equivalence of components, and the concatenation operation.

[1] Some sources implementing real numbers, metric spaces, and the original model of [GS96] are available. Please contact the author.

1.2 Survey of Contents

The second chapter begins by defining the basics of networks: named message streams and contractive functions on such streams. Due to the contractivity with respect to the complete metric space of streams, these functions, called behaviors, have unique fixed points according to Bannach's fixed point theorem (T.B.4).

After that we define genericity, the correspondence to dynamic channel access rules. Here we use a different approach than [GS96], based on an idea of Radu Grosu, privacy of channel names is no longer explicitly controlled by each component's genericity rules, rather it is guaranteed in an assumption/commitment style. Private channels are simply active channels which cannot be initially used by the other components. This change brought remarkable simplifications in most of the proofs and definitions below.

The second new approach lays in our concept of partial behaviors, which are functions on prefixes of streams. They represent finite processes which are followed up by arbitrary behaviors, which possibly use access rights acquired by the prefix. Compare these prefixes with the prefixes of the λ-calculus for which they can serve as interpretations.

As a base for name abstractness and equivalence of components, we introduce the renaming of components. Renaming was motivated by the structural congruence rule of the π-calculus which states that bound variables can be renamed. The two kinds of bound variables, hidden names and received names, appear as private and as received (or available) names in our model. The renameability of private names is realized through the equivalence of privately renamed components. Received names, in our case, available names, have to be present in all possible variations because of name abstractness. [2]

A component is name abstract if it is independent of the concrete names it receives additionally to its initially active channels, or in our case, if it is closed with respect to the name abstract closure. After proofing the properties of these concepts, we can define the set of components and structure it with an equivalence relation which identifies components which only differ by the choice of their private names.

The third chapter examines the operations on the defined components. Besides the operators which correspond to the ones given in [GS96] (in the context of our changed model), we introduce the union, which corresponds to the sum operator of the π-calculus. Furthermore we define the concatenation of partial components, giving expression to our concept of time partiality in acordance to the π prefix operator '..

The biggest part of the chapter however, is devoted to the three main operators: feedback, parallel composition, and mutual feedback composition. All three are given in an asynchronous version, like the ones given in [GS96] and in a

[2] Name abstractness however, does not include the possiblity of receiving an already active channel name. This case has to be included explicitly, which is a clear weakness of our model.

synchronous one. The explicit discussion of synchronizity with its fair split operator was necessary to form a model for the also synchronous π-calculus.

For each operator we give proofs of genericity, which includes genericity of the result, but also full abstractness, name abstractness, and the congruence of the operator to our equivalence relation.

1.3 Results

In our opinion, we made some important steps towards a denotational model of the π-calculus, but as yet this task is unfinished in two aspects: Firstly, there are still some technical difficulties in our modelling. Secondly, we need to demonstrate the validity of the π-reduction relation in our model. That would mean some kind of reduction relation on components, which has the interpreted π-reduction as a subrelation.

Nevertheless, our attempt proves the expressiveness of the used model and it could also help to learn something about Milner's calculus. In any case, this is just the starting point for a more thorough and outreaching examination of the relation between the denotational and the operational models of mobile networks. We hope that this thesis can serve as a contribution upon which such examinations could be based.

Our concept of name abstractness, although motivated by the π-calculus, can play an important role independently of its motivation. Renaming in general, not only in this application, is a necessary tool for solving the problems arising when using names. And, talking about mobile networks, it will be difficult to do without names. Like in logic formulae, we have to handle identifiers in a very abstract manner by ignoring all inessential aspects of a name. Perhaps, name abstractness and renaming are first steps in this direction.

While deepening our understanding of mobile networks and their denotational models, there were important results as by-products. For example, the composition of [GS96] is *not* associative. That was our main motivation for turning down the list representation in favor of multisets as representation for messages sent during a time unit on one channel.

We hope to be able to present proofs for basic algebraic properties and for the validity of our π-interpretation, and also a data-flow term language for the description of networks in a later, extended version of this paper.

Chapter 2

Components

We will now introduce our understanding of Mobile Networks, which are, in our terminology, *components*. This indicates that they can be elements of greater networks which, due to their abstractness, form components again.

2.1 Behaviors

Behaviors are stream processing functions which deterministically model the reactions of a component to all possible inputs. At first, the arguments of such functions are defined. We will see that they are uniform to any component, regardless of its initial access rights. Mobility is modeled by a dynamic restriction on the channels that may influence the reaction of a behavior, i.e., its values. Therefore, the outward appeareance of such a function, its functionality, allows arbitrary communications.

2.1.1 Message Streams

Messages

The objects of the communication in a network are messages. They can be divided into names and *proper* messages, that are carrying information which is unrelated to channel access. Thus, they are not interesting to our model – we simply assume them to be an arbitrary set: D is the set of proper messages. Completely ignoring these messages would also be possible, as, e.g., it is done in the context of the π-*Calculus*.

Central to our model is the set of channel names N. We do not assume it to be structured; all distinctions within the set are done by the components, which choose their specific initial channels (private and public).

Based on channel names, we define port names: each channel has an input and an output port, representing read and write access to the channel.

! and ? are constructors on N, for each channel name $n \in \mathrm{N}$ $!n$ denotes the output port and $?n$ the input port of the channel. Certainly a channel cannot be an output port and an input port at the same time. More formally:

$$\forall n, m \in \mathrm{N} : n \neq m \Rightarrow !n \neq !m \wedge ?n \neq ?m$$
$$\mathrm{N} \cap !\mathrm{N} = \mathrm{N} \cap ?\mathrm{N} = !\mathrm{N} \cap ?\mathrm{N} = \emptyset$$
$$M \subseteq \mathrm{N} : ?!M := ?M \cup !M$$
$$M \subseteq ?!\mathrm{N} : \tilde{M} := \{n \in \mathrm{N} | ?n \in M \vee !n \in M\}$$
$$\mathrm{M} := \mathrm{D} \cup !\mathrm{N} \cup ?\mathrm{N}$$

Care has to be taken in the distinction of channel names and port names, for an example, see section 2.3.4.

Named Message Streams

Proper messages and port names form the contents of input and output streams. These contents will be structured by the channel names and by time units. The communications of each unit and each name are represented by a multiset of messages.

$$\mathrm{NMS} := \mathbb{N} \to (\mathrm{N} \to \mathrm{M}^{+})$$
$$\mathrm{PNMS} := \{\phi \downarrow_n | \phi \in \mathrm{NMS} \wedge n \in \mathbb{N}^{\infty}\}$$

Along with the named message streams we also introduced *partial named message streams*, which can be seen as prefixes of a complete stream. The following definitions will consequently be formulated for partial streams, which include complete streams as a special case. With the idea of time partiality we aim at defining partial components which react to certain prefixes and which are followed up by the behaviors of a complete component.

2.1.2 Contractive Behaviors

Behaviors are simply functions on NMS. In contrast to this simplicity, we cannot accept every function on PNMS as a partial behavior. It should react partially but deterministically to any complete input, which means it takes a definite prefix of the input and outputs a stream of the same length.

Behaviors

Definition 2.1 (Behaviors)

$$\begin{aligned} \mathrm{B} &:= \mathrm{NMS} \to \mathrm{NMS} \\ \mathrm{PB} &:= \{f \in \mathrm{PNMS} \rightharpoonup \mathrm{PNMS} | \forall \phi \in dom(f) : |\phi| = |f(\phi)| \wedge \\ &\qquad \forall \phi \in \mathrm{NMS} : \exists_1 \phi' \in dom(f) : \phi' \sqsubseteq \phi\}. \end{aligned}$$

Remark: $\mathrm{B} \subseteq \mathrm{PB}$ by $\mathrm{B} \equiv \{f \in \mathrm{PB} | dom(f) = \mathrm{NMS}\}$

Contractivity

Proper behaviors of a component also have to follow a rule imposed by time flow: there has to be a delay between stimulus and reaction. Formally, this means that an input prefix of length n determines an output prefix of length $n+1$. Because of its equivalence to contractivity in the metric space of streams, we call such functions *contractive* (see B.3,B.4).

Definition 2.2 (Contractivity)

$$
\begin{aligned}
f \in \mathrm{PB}: \ \mathrm{ctrct}(f) \ \ :\Leftrightarrow \ \ & \forall \phi_1, \phi_2 \in dom(f), n+1 < |\phi_1|, |\phi_2| : \\
& \phi_1 \downarrow_n = \phi_2 \downarrow_n \Rightarrow f(\phi_1) \downarrow_{n+1} = f(\phi_2) \downarrow_{n+1}
\end{aligned}
$$

Nonexpansiveness

A weaker form of correspondence with the direction of time flow is nonexpansivity, which allows reaction within the same time unit, but not earlier. Again the terminology stems from the metric view of the functions.

Definition 2.3 (Nonexpansiveness)

$$
\begin{aligned}
f \in \mathrm{PB}: \ \mathrm{nexp}(f) \ \ :\Leftrightarrow \ \ & \forall \phi_1, \phi_2 \in dom(f), n < |\phi_1|, |\phi_2| : \\
& \phi_1 \downarrow_n = \phi_2 \downarrow_n \Rightarrow f(\phi_1) \downarrow_n = f(\phi_2) \downarrow_n
\end{aligned}
$$

Concatenation of Behaviors

Now we become more concrete with our idea of stream prefixes. Partial behaviors, i.e., partial functions defined on exactly one prefix of every stream, can be composed time sequentially, or in our terminology: *concatenated*:

Definition 2.4 (Concatenation of Behaviors)

$$
\begin{aligned}
& f, g \in \mathrm{PB}; \alpha \in dom(f) \ \wedge \ \beta \in dom(g) \Rightarrow \alpha \circ \beta \in dom(f \circ g): \\
& \quad (f \circ g)(\alpha \circ \beta) \ \ := \ \ f(\alpha) \circ f(\beta)
\end{aligned}
$$

The set of contractive functions is closed with respect to this operation. Consider this and its welldefinedness as a first justification for our definition.

Theorem 2.1 (Concatenation preserves Contractivity)

$$f, g \in \mathrm{PB} : \mathrm{ctrct}(f) \wedge \mathrm{ctrct}(g) \Rightarrow \mathrm{ctrct}(f \circ g)$$

Proof:

Assume $f, g \in \mathrm{PB}$, both contractive: $\mathrm{ctrct}(f) \wedge \mathrm{ctrct}(g)$.
Let $n \in \mathbb{N}, \phi_1, \phi_2 \in dom(f \circ g)$ and $|\phi_1|, |\phi_2| > n+1$ and $\phi_1 \downarrow_n = \phi_2 \downarrow_n$.
Since $\phi_1, \phi_2 \in dom(f \circ g)$, $\phi_1 = \phi_1' \circ \phi_1''$ and $\phi_2 = \phi_2' \circ \phi_2''$ with $\phi_1', \phi_2' \in dom(f)$ and $\phi_1'', \phi_2'' \in dom(g)$.

1st case: $\phi_1' = \phi_2'$ $(\Leftrightarrow |\phi_1'| = |\phi_2'| \leq n)$
$m < n+1, c \in \mathrm{N}$:

$$f \circ g(\phi_1)(m)(c) = \begin{cases} f(\phi_1')(m)(c) & \text{if } m < |\phi_1'| \\ g(\phi_1'')(m - |\phi_1'|)(c) & \text{if } m \geq |\phi_1'| \end{cases} =$$

$$= \begin{cases} f(\phi_2')(m)(c) & \text{if } m < |\phi_2'| \\ g(\phi_2'')(m - |\phi_2'|)(c) & \text{if } m \geq |\phi_2'| \end{cases} = f \circ g(\phi_2)(m)(c)$$

The only difficult point here is, that $g(\phi_1'')(m - |\phi_2'|)(c) = g(\phi_2'')(m - |\phi_2'|)(c)$ for $|\phi_1'| \leq m < n+1$. It results from $\phi_1 \downarrow_n = \phi_2 \downarrow_n \Rightarrow \phi_1'' \downarrow_{n-|\phi_1'|} = \phi_2'' \downarrow_{n-|\phi_1'|} \Rightarrow g(\phi_1'') \downarrow_{n-|\phi_1'|+1} = g(\phi_2'') \downarrow_{n-|\phi_1'|+1}$.

2nd case: $|\phi_1'|, |\phi_2'| > n$ $m < n+1$ $(\Rightarrow m < |\phi_1'|, |\phi_2'|); c \in \mathrm{N}$:

$$f \circ g(\phi_1)(m)(c) \;=\; f(\phi_1')(m)(c) \overset{(\mathrm{ctrct(f)})}{=} f(\phi_2')(m)(c) \;=\; f \circ g(\phi_2)(m)(c)$$

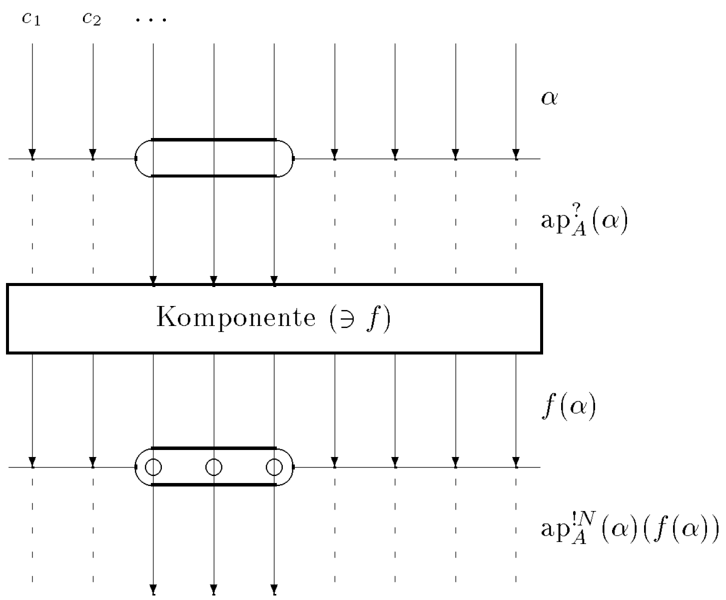
$\square$

2.2 Genericity

Now we move from defining the statics of behaviors to their dynamic constraints which characterize the mobility aspect of our networks. Contractive behaviors so far could interact on any channel. Their functionality allows them universal channel access, because arbitrary access can be acquired by each component in this model.

2.2.1 Genericity Filters

The constraints are described by filter functions on the input and the output of behaviors. The filters delete all inactive channel contents and all names of inactive channels in the output. Basic to this is the dynamically changing set of active ports.



Active Ports

The active ports are a stream of port sets defined with respect to an initially active set and an input stream. In each period the set is extended by all ports whose name was received the unit before on any active input channel.

Definition 2.5 (Active Ports: ap**)**

$$\begin{aligned}
&\phi \in \mathrm{PNMS}, A \subseteq !?\mathrm{N}, n < |\phi| : \\
&\quad \mathrm{ap}_A(\phi)(0) \quad\;\; := \; A \\
&\quad \mathrm{ap}_A(\phi)(n+1) := \mathrm{ap}_A(\phi)(n) \cup \bigcup_{?c \in \mathrm{ap}_A(\phi)(n)} \{p \in ?!\mathrm{N} \mid p \in \phi(n)(c)\}
\end{aligned}$$

Filter Functions

The input of a function can now be filtered with respect to an initially active set and the input itself. The possible distinction between the message stream defining the active ports and the filtered one is omitted here.

Definition 2.6 (Input Filter: $\mathrm{ap}^?$**)**

$$\begin{aligned}
&\phi \in \mathrm{PNMS}, A \subseteq !?\mathrm{N}, c \in \mathrm{N}, n < |\phi| : \\
&\quad \mathrm{ap}_A^?(\phi)(n)(c) := \begin{cases} \phi(n)(c) & \mathit{if}\ ?c \in \mathrm{ap}_A(\phi)(n) \\ \hat{\emptyset} & \mathit{else} \end{cases}
\end{aligned}$$

Output is filtered in the same way, but with a separate stream which determines the input. Additionally, all inactive ports are deleted from the result; a component should only confer access rights it already owns itself.

Definition 2.7 (Output Filter: $\mathrm{ap}^{!N}$)

$\phi, \psi \in \mathrm{PNMS}, A \subseteq !?\mathrm{N}, c \in \mathrm{N}, n < |\phi|$:

$$\mathrm{ap}^{!N}_A(\phi)(\psi)(n)(c) := \begin{cases} (\mathrm{ap}_A(\phi)(n) \cup \mathrm{D})©\psi(n)(c) & \textit{if } !c \in \mathrm{ap}_A(\phi)(n) \\ \hat{\emptyset} & \textit{else} \end{cases}$$

Generic Behaviors

Now we can define genericity as corespondence with the rules given above. Genericity is established if the function behaves as if it were filtered.

Definition 2.8 (Genericity)

$f \in \mathrm{PB}$:

$$\begin{aligned} \mathrm{generic}_A(f) :\Leftrightarrow\ & \forall \phi \in \mathrm{PNMS} : \phi \in dom(f) \Leftrightarrow \mathrm{ap}^?_A(\phi) \in dom(f) \wedge \\ & \forall \phi \in dom(f) : f(\phi) = f(\mathrm{ap}^?_A(\phi)) = \mathrm{ap}^{!N}_A(\phi)(f(\phi)) \end{aligned}$$

Set of Generic Contractive Behaviors

Definition 2.9 (Generic Contractive Behaviors)

$$\begin{aligned} \mathrm{GCPB}_A &:= \{f \in \mathrm{PB} | \, \mathrm{ctrct}(f) \wedge \mathrm{generic}_A(f)\} \\ \mathrm{GCB}_A &:= \{f \in \mathrm{B} | \, \mathrm{ctrct}(f) \wedge \mathrm{generic}_A(f)\} \end{aligned}$$

2.2.2 Properties

Nonexpansiveness

It is easy to see that the active set develops delayed with respect to the input stream and that, as a consequence, the filter functions are nonexpansive in their arguments.

Lemma 2.2 (ap **is contractive)**

$A \subseteq ?!\mathrm{N}$; $\phi, \psi \in \mathrm{PNMS}$; $m \leq |\phi|, |\psi|$:

$$\phi \downarrow_m = \psi \downarrow_m \Longrightarrow \mathrm{ap}_A(\phi) \downarrow_{m+1} = \mathrm{ap}_A(\psi) \downarrow_{m+1}$$

Proof:

$\mathrm{ap}_A(\phi)(n)$ is defined without refering to the values of ϕ at time n or at any time after ($> n$). So an inductive proof is straightforward.

□

Theorem 2.3 ($\mathrm{ap}^?$ **is nonexpansive)**

$A \subseteq ?!\mathrm{N}$; $\phi, \psi \in \mathrm{PNMS}$; $m \leq |\phi|, |\psi|$:

$$\phi \downarrow_m = \phi \downarrow_m \Longrightarrow \mathrm{ap}^?_A(\phi) \downarrow_m = \mathrm{ap}^?_A(\psi) \downarrow_m$$

Proof:

$\mathrm{ap}^?_A(\phi)(n)$ is defined in terms of the contractive, and thereby nonexpansive, ap applied on n and in terms of $\phi(n)$.

□

Theorem 2.4 ($\mathrm{ap}^{!N}$ **is nonexpansive)**

$$A \subseteq ?!\mathrm{N};\ \phi_1, \psi_1, \phi_2, \psi_2 \in \mathrm{PNMS};\ m \le |\phi_1|, |\psi_1|, |\phi_2|, |\psi_2| :$$
$$\phi_1 \downarrow_m = \psi_1 \downarrow_m,\ \phi_2 \downarrow_m = \psi_2 \downarrow_m \Longrightarrow \mathrm{ap}^{!N}_A(\phi_1)(\phi_2) \downarrow_m = \mathrm{ap}^{!N}_A(\psi_1)(\psi_2) \downarrow_m$$

Proof:

Trivial. Compare T.2.3.

Monotonicity

We state some hardly surprising but necessary facts about the monotonicity of the defined functions and sets.

Lemma 2.5 (ap **is monotonic)**

$$A_1, A_2 \subseteq \mathrm{N}; n_1, n_2 \in I\!N; \phi_1, \phi_2 \in \mathrm{PNMS} :$$
$$A_1 \subseteq A_2 \wedge n_1 \le n_2 \le |\phi_1| + 1, |\phi_2| + 1 \wedge \phi_1 \subseteq \phi_2 \Rightarrow$$
$$\Rightarrow \mathrm{ap}_{A_1}(\phi_1)(n_1) \subseteq \mathrm{ap}_{A_2}(\phi_2)(n_2)$$

Proof:

Obvious for each premise separately. And with that:

$$\mathrm{ap}_{A_1}(\phi_1)(n_1) \subseteq \mathrm{ap}_{A_2}(\phi_1)(n_1) \subseteq \mathrm{ap}_{A_2}(\phi_2)(n_1) \subseteq \mathrm{ap}_{A_2}(\phi_2)(n_2)$$

□

Lemma 2.6 ($\mathrm{ap}^?$ **is monotonic)**

$$A_1 \subseteq A_2 \subseteq ?!\mathrm{N}, \phi \in \mathrm{PNMS} :$$
$$\forall c \in \mathrm{N}, n < |\phi| :\ \mathrm{ap}^?_{A_1}(\phi)(n)(c) = \hat{\emptyset} \vee \mathrm{ap}^?_{A_1}(\phi)(n)(c) = \mathrm{ap}^?_{A_2}(\phi)(n)(c)$$

Lemma 2.7 ($\mathrm{ap}^{!\mathrm{N}}$ **is monotonic)**

$$A_1 \subseteq A_2 \subseteq ?!\mathrm{N}, \phi_1, \phi_2 \in \mathrm{PNMS}, |\phi_1| = |\phi_2|, \phi_1 \subseteq \phi_2 :$$
$$\forall \psi \in \mathrm{PNMS} :\ \mathrm{ap}^{!\mathrm{N}}_{A_1}(\phi_1)(\psi) \hat{\subseteq} \mathrm{ap}^{!\mathrm{N}}_{A_2}(\phi_2)(\psi)$$

Due to this monotonicity, we will call a filter based on a larger initial set or a larger input stream *less restrictive*. As we will see in the next section, its effect on any stream will be neglectable in combination with a more restrictive one.

Theorem 2.8 (generic **is monotonic)**

$$A_1 \subseteq A_2 \subseteq ?!\mathrm{N}, f \in \mathrm{PB} : \mathrm{generic}_{A_1}(f) \Rightarrow \mathrm{generic}_{A_2}(f)$$

Theorem 2.9 (GCPB **is monotonic)**

$$A_1 \subseteq A_2 \subseteq ?!\mathrm{N} : \mathrm{GCPB}_{A_1} \subseteq \mathrm{GCPB}_{A_2}$$

Idempotence

Here we prove what happens when we apply the filters several times.

ap is self correcting, that means it does not need a correct input. It rather behaves as if the input was corrected before.

Lemma 2.10

$A_2 \subseteq A_1 \subseteq ?!\mathrm{N}$:
$\forall \phi \in \mathrm{PB}, n \leq |\phi| : \mathrm{ap}_{A_2}(\mathrm{ap}^?_{A_1}(\phi))(n) = \mathrm{ap}_{A_2}(\phi)(n)$

Proof:

We give a proof by induction over n:

n=0: (for $|\phi| \neq 0$)

$\mathrm{ap}_{A_2}(\mathrm{ap}^?_{A_1}(\phi))(0) = A_2 = \mathrm{ap}_{A_2}(\phi)(0)$

n+1: (for $|\phi| > n+1$)

$\mathrm{ap}_{A_2}(\mathrm{ap}^?_{A_1}(\phi))(n+1) =$

$$= \mathrm{ap}_{A_2}(\mathrm{ap}^?_{A_1}(\phi))(n) \cup \bigcup_{?c \in \mathrm{ap}_{A_2}(\mathrm{ap}^?_{A_1}(\phi))(n)} \{p \in ?!\mathrm{N} | \, p \in \mathrm{ap}^?_{A_1}(\phi)(n)(c)\} =$$

$$\stackrel{(\mathrm{I.A.})}{=} \mathrm{ap}_{A_2}(\phi)(n) \cup \bigcup_{?c \in \mathrm{ap}_{A_2}(\phi)(n)} \{p \in ?!\mathrm{N} | \, p \in \phi(n)(c) \wedge ?c \in \mathrm{ap}_{A_1}(\phi)(n)\} =$$

$$\stackrel{(\mathrm{T.2.5})}{=} \mathrm{ap}_{A_2}(\phi)(n+1)$$

□

If a stream is input filtered twice, first less and then more restrictive, the first filter is superfluous.

Theorem 2.11 (Multiple Input Filtering)

$A_2 \subseteq A_1 \subseteq ?!\mathrm{N}$:
$\forall \phi \in \mathrm{PNMS} : \mathrm{ap}^?_{A_2}(\mathrm{ap}^?_{A_1}(\phi)) = \mathrm{ap}^?_{A_2}(\phi)$

Proof:

$n < |\phi|, c \in \mathrm{N}$:

$\mathrm{ap}^?_{A_2}(\mathrm{ap}^?_{A_1}(\phi))(n)(c) =$

$$= \begin{cases} \mathrm{ap}^?_{A_1}(\phi)(n)(c) \text{ if } ?c \in \mathrm{ap}_{A_2}(\mathrm{ap}^?_{A_1}(\phi))(n) \\ \hat{\emptyset} \qquad \text{else} \end{cases} =$$

$$\stackrel{(\mathrm{T.2.10})}{=} \begin{cases} \phi(n)(c) \text{ if } ?c \in \mathrm{ap}_{A_1}(\phi)(n) \cap \mathrm{ap}_{A_2}(\phi)(n) \\ \hat{\emptyset} \qquad \text{else} \end{cases} =$$

$$\stackrel{(\mathrm{T.2.5})}{=} \mathrm{ap}^?_{A_2}(\phi)(n)(c)$$

□

For output filtering, a prior less restrictive input filter is irrelevant.

Theorem 2.12 (Output Filtering with respect to Filtered Input)

$A_2 \subseteq A_1 \subseteq ?!\mathrm{N}$:
$\forall \phi, \psi \in \mathrm{PB} : \mathrm{ap}^{!N}_{A_2}(\mathrm{ap}^{?}_{A_1}(\phi))(\psi) \;=\; \mathrm{ap}^{!\mathrm{N}}_{A_2}(\phi)(\psi)$

Proof:

$n < |\phi|, |\psi|; c \in \mathrm{N}$:

$$\mathrm{ap}^{!N}_{A_2}(\mathrm{ap}^{?}_{A_1}(\phi))(\psi)(n)(c) =$$

$$= \begin{cases} \mathrm{ap}_{A_2}(\mathrm{ap}^{?}_{A_1}(\phi))(n) \cup \mathrm{D}©\psi(n)(c) & \text{if } !c \in \mathrm{ap}_{A_2}(\mathrm{ap}^{?}_{A_1}(\phi))(n) \\ \hat{\emptyset} & \text{else} \end{cases} =$$

$$\overset{(\text{T.2.5,2.10})}{=} \mathrm{ap}^{!N}_{A_2}(\phi)(\psi)(n)(c)$$

□

Analogous to multiple input filtering, a second less restrictive filtering of the output can be omitted.

Theorem 2.13 (Multiple Output Filtering)

$A_1 \subseteq A_2 \subseteq ?!\mathrm{N}, \phi_1, \phi_2 \in \mathrm{PNMS}, \phi_1 \subseteq \phi_2$:
$\forall \psi \in \mathrm{PB} : \mathrm{ap}^{!N}_{A_2}(\phi_2)(\mathrm{ap}^{!N}_{A_1}(\phi_1)(\psi))(\psi) \;=\; \mathrm{ap}^{!N}_{A_1}(\phi_1)(\psi)$

Proof:

$n < |\phi|, |\psi|; c \in \mathrm{N}$:

$$\mathrm{ap}^{!N}_{A_2}(\phi_2)(\mathrm{ap}^{!N}_{A_1}(\phi_1)(\psi))(\psi)(n)(c) =$$

$$= \begin{cases} \mathrm{ap}_{A_2}(\phi_2)(n) \cup \mathrm{D}©\mathrm{ap}^{!N}_{A_1}(\phi_1)(\psi)(n)(c) & \text{if } !c \in \mathrm{ap}_{A_2}(\phi_2)(n) \\ \hat{\emptyset} & \text{else} \end{cases} =$$

$$= \begin{cases} \mathrm{ap}_{A_2}(\phi_2)(n) \cup \mathrm{D}©\mathrm{ap}_{A_1}(\phi_1)(n) \cup \mathrm{D}©\psi(n)(c) & \\ \quad \text{if } !c \in \mathrm{ap}_{A_2}(\phi_2)(n) \cap \mathrm{ap}_{A_1}(\phi_1)(n) & \\ \hat{\emptyset} & \text{else} \end{cases} =$$

$$\overset{(\text{T.2.5})}{=} \begin{cases} \mathrm{ap}_{A_1}(\phi_1)(n) \cup \mathrm{D}©\psi(n)(c) & \text{if } !c \in \mathrm{ap}_{A_1}(\phi_1)(n) \\ \hat{\emptyset} & \text{else} \end{cases} =$$

$$= \mathrm{ap}^{!N}_{A_1}(\phi_1)(\psi)$$

□

Concatenation

As we saw in T.2.1, concatenation preserves contractivity. In this section we prove that it also preserves genericity with a special regard to '*availability*'.

Availability This term denotes the possibility that one partial behavior can make some ports available to its follow up. These ports will not be part of the behavior's initially active ports, but they will be active in case of a break off of the input (and output). For every finite stream in the behavior's domain the active ports at the end of the stream will include the available ports. Intuitively speaking, the behavior may not break off as long as the ports which should be available were not received.

Genericity Preservation We start with basic congruence rules about filtering with respect to prefixes and suffixes. The prefix/suffix of a filtered stream is the filtered prefix/suffix of the stream:

Lemma 2.14 (Prefixes of $\mathrm{ap}^?$**)**

$A \subseteq ?!\mathrm{N}$; $\phi \in \mathrm{PNMS}$; $m < |\phi|$:

$$\mathrm{ap}^?_A(\phi)\downarrow_m = \mathrm{ap}^?_A(\phi\downarrow_m)$$

Proof:

$$\phi\downarrow_m = (\phi\downarrow_m)\downarrow_m \overset{(\mathrm{T.2.3})}{\Rightarrow} \mathrm{ap}^?_A(\phi)\downarrow_m = \mathrm{ap}^?_A(\phi\downarrow_m)\downarrow_m \overset{(|\mathrm{ap}^?_A(\phi\downarrow_m)| = m)}{=} \mathrm{ap}^?_A(\phi\downarrow_m)$$

□

Lemma 2.15 (Prefixes of $\mathrm{ap}^{!N}$**)**

$A \subseteq ?!\mathrm{N}$; $\phi, \psi \in \mathrm{PNMS}$; $m < |\phi|, |\psi|$:

$$\mathrm{ap}^{!N}_A(\phi)\downarrow_m = \mathrm{ap}^{!N}_A(\phi\downarrow_m)$$

Proof:

$$\phi\downarrow_m = (\phi\downarrow_m)\downarrow_m \wedge \psi\downarrow_m = (\psi\downarrow_m)\downarrow_m \Rightarrow$$

$$\overset{(\mathrm{T.2.4})}{\Rightarrow} \quad \mathrm{ap}^{!N}_A(\phi)(\psi)\downarrow_m = \mathrm{ap}^{!N}_A(\phi\downarrow_m)(\psi\downarrow_m)\downarrow_m =$$

$$\overset{(|\mathrm{ap}^{!N}_A(\phi\downarrow_m)(\psi\downarrow_m)| = m)}{=} \quad \mathrm{ap}^{!N}_A(\phi\downarrow_m)(\psi\downarrow_m)$$

□

Lemma 2.16 (Suffixes of ap**)**

$A \subseteq ?!\mathrm{N}$; $\phi \in \mathrm{PNMS}$; $m < |\phi|$:

$$\mathrm{ap}_A(\phi)\uparrow^m = \mathrm{ap}_{\mathrm{ap}_A(\phi)(m)}(\phi\uparrow^m)$$

Proof:

Let $m < |\phi|$, we will perform an inductive proof over n in $\mathrm{ap}_A(\phi)\uparrow^m (n)$:

n=0: $(0 < |\phi| - m)$

$\mathrm{ap}_A(\phi)\uparrow^m (0) = \mathrm{ap}_A(\phi)(m) = \mathrm{ap}_{\mathrm{ap}_A(\phi)(m)}(\phi\uparrow^m)(0)$

n+1: $(0 < |\phi| - m)$

$$\mathrm{ap}_A(\phi) \uparrow^m (n+1) =$$

$$\begin{aligned}
&= \mathrm{ap}_A(\phi)(m+n+1) && = \\
&= \mathrm{ap}_A(\phi)(m+n) \cup \bigcup_{?c \in \mathrm{ap}_A(\phi)(m+n)} \{p \in ?!\mathrm{N} \mid p \in \phi(m+n)(c)\} && = \\
&= \mathrm{ap}_A(\phi) \uparrow^m (n) \cup \bigcup_{?c \in \mathrm{ap}_A(\phi) \uparrow^m (n)} \{p \in ?!\mathrm{N} \mid p \in \phi \uparrow^m (n)(c)\} && = \\
&\overset{\text{(I.A.)}}{=} \mathrm{ap}_{\mathrm{ap}_A(\phi)(m)}(\phi \uparrow^m)(n) \cup \bigcup_{?c \in \mathrm{ap}_{\mathrm{ap}_A(\phi)(m)}(\phi \uparrow^m)(n)} \{p \in ?!\mathrm{N} \mid p \in \phi \uparrow^m (n)(c)\} && = \\
&= \mathrm{ap}_{\mathrm{ap}_A(\phi)(m)}(\phi \uparrow^m)(n+1)
\end{aligned}$$

□

Lemma 2.17 (Suffixes of $\mathrm{ap}^?$)

$A \subseteq ?!\mathrm{N}$; $\phi \in$ PNMS; $m < |\phi|$:

$$\mathrm{ap}^?_A(\phi) \uparrow^m = \mathrm{ap}^?_{\mathrm{ap}_A(\phi)(m)}(\phi \uparrow^m)$$

Proof:

$n < |\phi| - m, c \in \mathrm{N}$:

$$\begin{aligned}
\mathrm{ap}^?_A(\phi) \uparrow^m (n)(c) &= \mathrm{ap}^?_A(\phi)(m+n)(c) = \\
&= \begin{cases} \phi(m+n)(c) & \text{if } ?c \in \mathrm{ap}_A(\phi)(m+n) \\ \hat{\emptyset} & \text{else} \end{cases} = \\
&= \begin{cases} \phi \uparrow^m (n)(c) & \text{if } ?c \in \mathrm{ap}_A(\phi) \uparrow^m (n) \\ \hat{\emptyset} & \text{else} \end{cases} = \\
&= \begin{cases} \phi \uparrow^m (n)(c) & \text{if } ?c \in \mathrm{ap}_{\mathrm{ap}_A(\phi)(m)}(\phi \uparrow^m)(n) \\ \hat{\emptyset} & \text{else} \end{cases} = \\
&= \mathrm{ap}^?_{\mathrm{ap}_A(\phi)(m)}(\phi \uparrow^m)(n)(c)
\end{aligned}$$

□

Lemma 2.18 (Suffixes of $\mathrm{ap}^{!N}$)

$A \subseteq ?!\mathrm{N}$; $\phi, \psi \in$ PNMS; $m < |\phi|, |\psi|$:

$$\mathrm{ap}^{!N}_A(\phi)(\psi) \uparrow^m = \mathrm{ap}^{!N}_{\mathrm{ap}_A(\phi)(m)}(\phi \uparrow^m)(\psi \uparrow^m)$$

Proof:

$n < |\phi| - m, |\psi| - m$; $c \in \mathrm{N}$:

$$\mathrm{ap}^{!N}_A(\phi)(\psi) \uparrow^m (n)(c) = \mathrm{ap}^{!N}_A(\phi)(\psi)(m+n)(c)$$

$$= \begin{cases} \mathrm{ap}_A(\phi)(m+n) \cup \mathrm{D}©\phi(m+n)(c) & \text{if } !c \in \mathrm{ap}_A(\phi)(m+n) \\ \hat{\emptyset} & \text{else} \end{cases} =$$

$$= \begin{cases} \mathrm{ap}_A(\phi)\uparrow^m(n) \cup \mathrm{D}©\phi\uparrow^m(n)(c) & \text{if } !c \in \mathrm{ap}_A(\phi)\uparrow^m(n) \\ \hat{\emptyset} & \text{else} \end{cases} =$$

$$= \begin{cases} \mathrm{ap}_{\mathrm{ap}_A(\phi)(m)}(\phi\uparrow^m)(n) \cup \mathrm{D}©\phi\uparrow^m(n)(c) & \text{if } !c \in \mathrm{ap}_{\mathrm{ap}_A(\phi)(m)}(\phi\uparrow^m)(n) \\ \hat{\emptyset} & \text{else} \end{cases} =$$

$$= \mathrm{ap}^{!N}_{\mathrm{ap}_A(\phi)(\psi)(m)}(\phi\uparrow^m)(\psi\uparrow^m)(n)(c)$$

$\square$

Now availability comes into play: if a partial behavior makes the initial set of its follow up available, the concatenation of both is generic.

Theorem 2.19 (Concatenation preserves Genericity)

$$\begin{aligned} & f, g \in \mathrm{PB};\ A, B \subseteq ?!\mathrm{N}: \\ & \quad \mathrm{generic}_A(f) \wedge \mathrm{generic}_B(g) \wedge \forall \phi \in dom(f) \setminus \mathrm{NMS} : B \subseteq \mathrm{ap}_A(\phi)(|\phi|) \\ & \qquad \Rightarrow \mathrm{generic}_A(f \circ g) \end{aligned}$$

Proof:

Let ϕ' bet that prefix of ϕ which is in f's domain: $\phi' \sqsubseteq \phi \wedge \phi' \in dom(f)$.

1. Input Genericity

$$\begin{aligned} (f\circ g)(\mathrm{ap}^?_A(\phi)) &= (f\circ g)(\mathrm{ap}^?_A(\phi)\downarrow_{|\phi'|} \circ \mathrm{ap}^?_A(\phi)\uparrow^{|\phi'|}) = \\ &\overset{(\text{L.2.14,2.17})}{=} (f\circ g)(\underbrace{\mathrm{ap}^?_A(\phi\downarrow_{|\phi'|})}_{\in dom(f)} \circ \underbrace{\mathrm{ap}^?_{\mathrm{ap}_A(\phi)(|\phi'|)}(\phi\uparrow^{|\phi'|})}_{\in dom(g)^{(*)}}) = \\ &\overset{(\text{D.2.4})}{=} f(\mathrm{ap}^?_A(\phi\downarrow_{|\phi'|}) \circ g(\mathrm{ap}^?_{\mathrm{ap}_A(\phi)(|\phi'|)}(\phi\uparrow^{|\phi'|})) = \\ &\overset{(\mathrm{generic}_B(g))}{=} f(\mathrm{ap}^?_A(\phi\downarrow_{|\phi'|}) \circ g(\mathrm{ap}^?_B(\mathrm{ap}^?_{\mathrm{ap}_A(\phi)(|\phi'|)}(\phi\uparrow^{|\phi'|}))) = \\ &\overset{(*)}{=} f(\mathrm{ap}^?_A(\phi\downarrow_{|\phi'|}) \circ g(\mathrm{ap}^?_B(\phi\uparrow^{|\phi'|})) = \\ &\overset{\left(\substack{\mathrm{generic}_A(f), \\ \mathrm{generic}_B(g)}\right)}{=} f(\phi\downarrow_{|\phi'|}) \circ g(\phi\uparrow_{|\phi'|}) = (f\circ g)(\phi) \end{aligned}$$

(*): Here we use that with $B \subseteq \mathrm{ap}_A(\phi)(|\phi'|)$: $\mathrm{ap}^?_B(\mathrm{ap}^?_{\mathrm{ap}_A(\phi)(|\phi'|)}(\phi\uparrow^{|\phi'|})) \overset{(\text{T.2.11})}{=}$ $\mathrm{ap}^?_B(\phi\uparrow^{|\phi'|})$, which is in $dom(g)$ because $\phi\uparrow^{|\phi'|}$ is.

2. Output Genericity

$$
\begin{aligned}
&\mathrm{ap}^{!N}_A(\phi)((f \circ g)(\phi)) = \\
&\quad = \quad \mathrm{ap}^{!N}_A(\phi)((f \circ g)(\phi)) \downarrow_{|\phi'|} \circ \, \mathrm{ap}^{!N}_A(\phi)((f \circ g)(\phi)) \uparrow_{|\phi'|} = \\
&\quad \overset{(\mathrm{L.2.15,2.18})}{=} \quad \mathrm{ap}^{!N}_A(\phi \downarrow_{|\phi'|})((f \circ g)(\phi) \downarrow_{|\phi'|}) \circ \mathrm{ap}^{!N}_{\mathrm{ap}_A(\phi')(|\phi'|)}(\phi \uparrow_{|\phi'|})(g(\phi \uparrow_{|\phi'|})) = \\
&\quad \overset{\left(\substack{\mathrm{generic}_A(f),\\ \mathrm{generic}_B(g)}\right)}{=} \quad f(\phi') \circ (\mathrm{ap}^{!N}_{\mathrm{ap}_A(\phi')(|\phi'|)}(\phi \uparrow_{|\phi'|})(\mathrm{ap}^{!N}_B(\phi \uparrow_{|\phi'|})(g(\phi \uparrow_{|\phi'|}))) = \\
&\quad \overset{(\mathrm{T.2.13})}{=} \quad f(\phi') \circ \mathrm{ap}^{!N}_B(\phi \uparrow_{|\phi'|})(g(\phi \uparrow_{|\phi'|})) = \\
&\quad \overset{(\mathrm{generic}_B(g))}{=} \quad f(\phi') \circ g(\phi'') = \\
&\quad = \quad (f \circ g)(\phi)
\end{aligned}
$$

□

2.3 Components

We will now develop the component concept step by step with its properties. After contractivity and genericity, which were already introduced, we will define *full abstractness* for behavior sets and then our *renaming concept* as a prerequisite to *name abstractness* and the *equivalence of components.*

Full abstractness ensures that our representation of nondeterministic components via sets of functions is unambiguous.

Renaming serves several purposes by permutating the channel names a component uses. If it can be applied on all initially nonactive names of a component without altering it, we call that component name abstract.

Some of the channels a component accesses initially are *private*. To realize them as kinds of local variables, we see all components which only differ in the names of their private channels as equivalent.

This results in the concept of components as fully abstract sets of contractive and generic behaviors which are abstract regarding inactive names. The private names of these components can be renamed without essential effects on the resulting networks.

2.3.1 Full Abstractness

For convenience in the following proofs and definitions, both abstractness conditions are introduced via closures. Each time, abstractness is fulfilled if the set is closed, i.e., if the closure has no effect on the set.

Full abstractness is the well known property that the functions of a component represent its input output relation in all possible *contractive and generic* combinations.

To exemplify the problem, consider a simple example: A component which nondeterministically relates two inputs with two outputs. It could be represented by two functions, both constant with the two different outputs. At the same time it could be represented by the two possible bijective functions from the two inputs to the two outputs. The fully abstract set, in this trivial case the set of all functions, forms an unambiguous normal form for all components.

Fully Abstract Closure

Definition 2.10 (Fully Abstract Closure)

$$F \subseteq \mathrm{PB} : \ \mathrm{FAC}_A(F) := \\ \{f \in \mathrm{GCPB}_A \mid \forall \phi \in \mathrm{dom}(f) : \exists f' \in F : \phi \in \mathrm{dom}(f') \wedge f(\phi) = f'(\phi)\}$$

Remark: $C \subseteq \mathrm{B} :\ \mathrm{FAC}_A(C) \subseteq \mathrm{B}$ because $\forall f \in \mathrm{FAC}_A(C) : dom(f) \subseteq \mathrm{NMS}$.

Closure Properties

By definition, *closure* always has the following three properties:

Extensivity ensures that a set is always extended by a closure, never reduced. In our case, no behaviors disappear.

Theorem 2.20 (Extensivity)

$$C \subseteq \mathrm{GCPB}_A : \ C \subseteq \mathrm{FAC}_A(C)$$

Proof:

$$\mathrm{FAC}_A(C) \ =$$

$$= \{f \in \mathrm{GCPB}_A \mid \forall \phi \in \mathrm{dom}(f) : \exists f' \in C : \phi \in \mathrm{dom}(f') \wedge f(\phi) = f'(\phi)\} \supseteq \\ \supseteq \{f \in \mathrm{GCPB}_A \mid \exists f' \in C : \forall \phi \in \mathrm{dom}(f) : \phi \in \mathrm{dom}(f') \wedge f(\phi) = f'(\phi)\} = \\ = C$$

□

Theorem 2.21 (Monotonicity)

$$C_1 \subseteq C_2 : \ \mathrm{FAC}_A(C_1) \subseteq \mathrm{FAC}_A(C_2)$$

Proof:

Obvious.

Theorem 2.22 (Idempotence)

$$A_1 \subseteq A_2;\ C \subseteq \mathrm{GCPB}_{A_1} : \\ \mathrm{FAC}_{A_2}(\mathrm{FAC}_{A_1}(C)) \ = \ \mathrm{FAC}_{A_2}(C)$$

Proof:

'$\supseteq$' is implied by T.2.20 and 2.21.
'$\subseteq$':

$$\mathrm{FAC}_{A_2}(\mathrm{FAC}_{A_1}(C)) \ =$$

$$= \{f \in \mathrm{GCPB}_{A_2} | \forall \phi \in \mathrm{dom}(f) : \exists f' \in \mathrm{GCPB}_{A_1} : \forall \phi' \in \mathrm{dom}(f') :$$
$$\exists f'' \in C : \phi' \in \mathrm{dom}(f'') \wedge f'(\phi') = f''(\phi') \wedge \phi \in \mathrm{dom}(f') \wedge f(\phi) = f'(\phi)\} \subseteq$$
$$\subseteq \{f \in \mathrm{GCPB}_{A_2} | \forall \phi \in \mathrm{dom}(f) : \exists f' \in \mathrm{GCPB}_{A_1} :$$
$$\exists f'' \in C : \phi \in \mathrm{dom}(f') \wedge f'(\phi) = f''(\phi) \wedge \phi \in \mathrm{dom}(f') \wedge f(\phi) = f'(\phi)\} =$$
$$= \{f \in \mathrm{GCPB}_{A_2} | \forall \phi \in \mathrm{dom}(f) : \exists f'' \in C : \phi \in \mathrm{dom}(f'') \wedge f(\phi) = f''(\phi)\} =$$
$$= \mathrm{FAC}_{A_2}(C)$$

□

2.3.2 Renaming

A renaming is a permutation of channel names which can be applied to a component. Normally, components depend on certain names by reacting to them differently than to others or by sending them and no others. These namings can be changed or even eliminated with renamings.

Renamings play a similar but not equal role as substitutions in the use of local variables. As any such variable can be substituted by a unused one without altering its meaning, we can rename certain channels for a component without altering it essentially. But in contrast to substitution renamings have to be bijective; we cannot unify two channels by giving them the same name.

Definitions

A *renaming function* τ is generally a *bijective* function on N, so its applicability on channel names and on port names is evident:

$\tau \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}$.

τ can also be seen as a function on port names $!?N$: $\tau(!n) := !\tau(n)$, $\tau(?n) := ?\tau(n)$

Applying a renaming on a stream renames the single multisets of each time unit and channel, and exchanging the contents of the channels according to the permutation. A renamed behavior renames its input reversely, applies the behavior on it, and renames the result.

Definition 2.11 (Renaming)

$\phi \in \mathrm{PNMS};\ n < |\phi|;\ c \in \mathrm{N} :$

$$\tau(\phi)(n)(c) := \tau(\phi(n)(\tau^{-1}(c)))$$

$f \in \mathrm{PB} :$

$$\mathrm{dom}(\tau(f)) := \tau(\mathrm{dom}(f)) :\ \forall \phi \in \mathrm{dom}(\tau(f)) : \tau(f)(\phi) := \tau(f(\tau^{-1}))$$

These definitions are justified by the following theorems about preservation of important behavior properties.

Contractivity

The renaming of a contractive behavior will still be contractive.

Theorem 2.23 (Renaming preserves Contractivity)

$f \in \mathrm{PB};\ \tau \in \mathrm{N} \stackrel{inj.}{\longrightarrow} \mathrm{N}:$

$\quad \mathrm{ctrct}(f) \Rightarrow \mathrm{ctrct}(\tau(f))$

Proof:

Let $\tau \in \mathrm{N} \stackrel{inj.}{\longrightarrow} \mathrm{N}; f \in \mathrm{PB}; \phi_1, \phi_2 \in dom(f); n+1 < |\phi_1|, |\phi_2|$:

$$\begin{array}{rcl}
\phi_1 \downarrow_n = \phi_2 \downarrow_n & \Rightarrow & \tau^{-1}(\phi_1) \downarrow_n = \tau^{-1}(\phi_2) \downarrow_n \Rightarrow \\
& \stackrel{(\mathrm{ctrct(f)})}{\Rightarrow} & f(\tau^{-1}(\phi_1)) \downarrow_{n+1} = f(\tau^{-1}(\phi_2)) \downarrow_{n+1} \Rightarrow \\
& \Rightarrow & \tau(f(\tau^{-1}(\phi_1))) \downarrow_{n+1} = \tau(f(\tau^{-1}(\phi_2))) \downarrow_{n+1} \Leftrightarrow \\
& \Leftrightarrow & \tau(f)(\phi_1) \downarrow_{n+1} = \tau(f)(\phi_2) \downarrow_{n+1}
\end{array}$$

□

Genericity

The renaming of a generic function will be generic with respect to the renamed initial set. This is central as a justification of our definitions.

We start by showing the effect of the filter functions on renamed streams.

Lemma 2.24 (ap of a Renamed Stream)

$A \subseteq \mathrm{N}, \phi \in \mathrm{PNMS}:$

$\quad \forall n \leq |\phi| : \mathrm{ap}_A(\phi)(n) \ = \ \tau^{-1}(\mathrm{ap}_{\tau(A)}(\tau(\phi))(n))$

Proof:

Proof by induction on n:

n=0: $(|\phi| \geq 0)$

$\tau^{-1}(\mathrm{ap}_{\tau(A)}(\tau(\phi))(0)) \ = \ A \ = \ \mathrm{ap}_A(\phi)(0)$

n+1: $(|\phi| \geq n+1)$

$\tau^{-1}(\mathrm{ap}_{\tau(A)}(\tau(\phi))(n+1)) \ =$

$$\begin{array}{rll}
= & \tau^{-1}(\mathrm{ap}_{\tau(A)}(\tau(\phi))(n)) \cup \tau^{-1}(\displaystyle\bigcup_{?c \in \mathrm{ap}_{\tau(A)}(\tau(\phi))(n)} \{p \in ?!\mathrm{N} |\, p \in \tau(\phi)(n)(c)\}) & = \\
\stackrel{(\mathrm{I.A.})}{=} & \mathrm{ap}_A(\phi)(n) \cup \displaystyle\bigcup_{?c \in \mathrm{ap}_{\tau(A)}(\tau(\phi))(n)} \{\tau^{-1}(p) \in ?!\mathrm{N} |\, p \in \tau(\phi(n)(\tau^{-1}(c)))\} & = \\
= & \mathrm{ap}_A(\phi)(n) \cup \displaystyle\bigcup_{?c \in \tau^{-1}(\mathrm{ap}_{\tau(A)}(\tau(\phi))(n))} \{p \in ?!\mathrm{N} |\, p \in \phi(n)(c)\} & = \\
\stackrel{(\mathrm{I.A.})}{=} & \mathrm{ap}_A(\phi)(n+1) &
\end{array}$$

□

Lemma 2.25 ($\mathrm{ap}^?$ **of a Renamed Stream)**

$A \subseteq \mathrm{N}, \phi \in \mathrm{PNMS}:$

$$\mathrm{ap}^?_A(\phi) = \tau^{-1}(\mathrm{ap}^?_{\tau(A)}(\tau(\phi)))$$

Proof:

$c \in \mathrm{N}, n < |\phi|:$

$\tau^{-1}(\mathrm{ap}^?_{\tau(A)}(\tau(\phi)))(n)(c) =$

$$\begin{aligned}
&= \tau^{-1}(\mathrm{ap}^?_{\tau(A)}(\tau(\phi))(n)(\tau(c))) &&= \\
&= \begin{cases} \tau^{-1}(\tau(\phi(n)(\tau(\tau^{-1}(c))))) \text{ if } \tau(?c) \in \mathrm{ap}_{\tau(A)}(\tau(\phi))(n) \\ \hat{\emptyset} \qquad \text{else} \end{cases} &&= \\
&\overset{(\text{L.2.24})}{=} \begin{cases} \phi(n)(c) \text{ if } ?c \in \mathrm{ap}_A(\phi)(n) \\ \hat{\emptyset} \qquad \text{else} \end{cases} &&= \\
&= \mathrm{ap}^?_A(\phi)(n)(c)
\end{aligned}$$

Lemma 2.26 ($\mathrm{ap}^{!N}$ **of a Renamed Stream)**

$A \subseteq \mathrm{N}, \phi, \psi \in \mathrm{PNMS}:$

$$\mathrm{ap}^{!N}_A(\phi)(\psi) = \tau^{-1}(\mathrm{ap}^{!N}_{\tau(A)}(\tau(\phi))(\tau(\psi)))$$

Proof:

$c \in \mathrm{N}, n < |\phi|:$

$\tau^{-1}(\mathrm{ap}^{!N}_{\tau(A)}(\tau(\phi))(\tau(\psi)))(n)(c) =$

$$\begin{aligned}
&= \tau^{-1}(\mathrm{ap}^{!N}_{\tau(A)}(\tau(\phi))(\tau(\psi))(n)(\tau(c))) &&= \\
&= \begin{cases} \tau^{-1}(\mathrm{ap}_{\tau(A)}(\tau(\phi))(n) \cup \mathrm{D}©\tau(\psi(n)(c)) \text{ if } !\tau(c) \in \mathrm{ap}_{\tau(A)}(\tau(\phi))(n) \\ \hat{\emptyset} \qquad \text{else} \end{cases} &&= \\
&\overset{(\text{L.2.24})}{=} \mathrm{ap}^{!N}_A(\phi)(\psi)(n)(c)
\end{aligned}$$

□

Theorem 2.27 (Renaming preserves Genericity)

$A \subseteq \mathrm{N}, f \in \mathrm{PB}: \quad \mathrm{generic}_A(f) \Rightarrow \mathrm{generic}_{\tau(A)}(\tau(f))$

Proof:

1. Input

$$\begin{aligned}
\tau(f)(\mathrm{ap}^?_{\tau(A)}(\phi)) &= \tau(f(\tau^{-1}(\mathrm{ap}^?_{\tau(A)}(\phi)))) = \\
&\overset{(\text{L.2.25})}{=} \tau(f(\mathrm{ap}^?_A(\tau^{-1}(\phi)))) = \\
&\overset{(\mathrm{generic}_A(f))}{=} \tau(f(\tau^{-1}(\phi))) = \\
&= \tau(f)(\phi)
\end{aligned}$$

2. Output

$$
\begin{aligned}
\mathrm{ap}^{!N}_{\tau(A)}(\phi)(\tau(f)(\phi)) &= \mathrm{ap}^{!N}_{\tau(A)}(\phi)(\tau(f(\tau^{-1}(\phi))) = \\
&\overset{(\mathrm{L2.26})}{=} \tau(\mathrm{ap}^{!N}_{A}(\tau^{-1}(\phi))(f(\tau^{-1}(\phi)))) = \\
&\overset{(\mathrm{generic}_A(f))}{=} \tau(f(\tau^{-1}(\phi))) = \\
&= \tau(f)(\phi)
\end{aligned}
$$

□

Concatenation

Concatenation is compatible with renamings: the renaming of a concatenation is the same as the concatenation of the renamed streams. With this it is easy to show that also function concatenation is congruent.

Renaming congruence of this kind will be used very often to show the compatibility of operations with renaming. If an operation is congruent it can be regarded as independent of concrete namings.

Lemma 2.28 (Concatenation of Streams is Renaming Congruent)
$\alpha, \beta \in \mathrm{NMS}: \ \tau(\alpha \circ \beta) \ = \ \tau(\alpha) \circ \tau(\beta)$

Proof:
Let $n \in \mathbb{N}, c \in \mathrm{N}$:

$$
\begin{aligned}
\tau(\alpha \circ \beta)(n)(c) &= \tau(\alpha \circ \beta(n)(\tau^{-1}(c))) && = \\
&= \begin{cases} \tau(\alpha(n)(\tau^{-1}(c))) & \text{if } n < |\tau(\alpha)| \\ \tau(\beta(n - |\tau(\alpha)|)(\tau^{-1}(c))) & \text{else} \end{cases} && = \\
&= (\tau(\alpha) \circ \tau(\beta))(n)(c)
\end{aligned}
$$

□

Theorem 2.29 (Concatenation is Renaming Congruent)

$f, g \in \mathrm{PB}: \ \tau(f \circ g) \ = \ \tau(f) \circ \tau(g)$

Proof:
Let $\alpha \in dom(f), \beta \in dom(g)$: Then and only then $\tau(\alpha \circ \beta) \in dom(\tau(f \circ g))$.

$$
\begin{aligned}
\tau(f \circ g)(\tau(\alpha \circ \beta)) &= \tau((f \circ g)(\alpha \circ \beta)) && = \\
&\overset{(\mathrm{L.2.28})}{=} \tau(f(\tau^{-1}(\tau(\alpha)))) \circ \tau(g(\tau^{-1}(\tau(\beta))))
\end{aligned}
$$

□

Full Abstractness

Full abstractness is preserved by renaming with respect to the renamed initial set. This is a consequence of the congruence of the closure.

Theorem 2.30 (FAC **is Renaming Congruent)**

$C \subseteq \mathrm{GCPB}_A;\ \tau \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}:$

$\tau(\mathrm{FAC}_A(C)) = \mathrm{FAC}_{\tau(A)}(\tau(C))$

Proof:

$$\begin{aligned}
&\tau(\mathrm{FAC}_A(C)) = \\
&= \{\tau(f) \in \mathrm{GCPB}_{\tau(A)} | \forall \phi \in \mathrm{dom}(f) : \exists f' \in C : \phi \in \mathrm{dom}(f') \wedge f(\phi) = f'(\phi)\} = \\
&= \{f \in \mathrm{GCPB}_{\tau(A)} | \forall \phi \in \mathrm{dom}(f) : \exists f' \in C : \phi \in \mathrm{dom}(f') \wedge \tau^{-1}(f)(\phi) = f'(\phi)\} = \\
&= \{f \in \mathrm{GCPB}_{\tau(A)} | \forall \phi \in \mathrm{dom}(f) : \exists f' \in C : \phi \in \mathrm{dom}(f') \wedge f(\phi) = \tau(f')(\phi)\} = \\
&= \{f \in \mathrm{GCPB}_{\tau(A)} | \forall \phi \in \mathrm{dom}(f) : \exists f' \in \tau(C) : \phi \in \mathrm{dom}(f') \wedge f(\phi) = f'(\phi)\} = \\
&= \mathrm{FAC}_{\tau(A)}(\tau(C))
\end{aligned}$$

□

Corollary 2.31 (Renaming preserves Full Abstractness)

$C \subseteq \mathrm{GCPB}_A;\ \tau \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}: \quad \mathrm{FAC}_A(C) = C \Rightarrow \mathrm{FAC}_{\tau(A)}(\tau(C)) = \tau(C)$

2.3.3 Name Abstractness

If we represent components by fully abstract sets of contractive and generic functions, we also model an annoying and superfluous feature. Components could distinguish incoming new port names by their name, but they should be independent of the concrete names of *inactive*, or better: unknown, names. Certainly, they should identify incoming new names in some way, for example, recognize if they are different or read and write ports of the same channel. Nevertheless, new channels should be anonymous, i.e., a new channel name is as good as any other.

This ignorance of the concrete names of inactive channels, which we call *name abstractness*, is realized through a closure. We demand a component to be closed with respect to that closure.

Name Abstract Closure

Definition 2.12 (Name Abstract Closure)

$F \subseteq \mathrm{PB} : \mathrm{NAC}_A(F) :=$

$\{f \in \mathrm{PB} | \exists \tau \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}, f' \in F : \tau|_A = id \wedge f = \tau(f')\}$

Remark: $C \subseteq \mathrm{B} : \mathrm{NAC}_A(C) \subseteq \mathrm{B}$

The effect of an application of this closure on a not already name abstract component is to add all behaviors which are necessary to fulfill name abstractness.

For example, consider a component which reacts with some output only to a certain, new channel name c (which is not in its active set). For each channel name c' outside of the initial set, there is a renaming which maps c to c' and which maps the initial set identically. Each such renaming adds a new behavior to the component set, resulting in a component which does not distinguish c from any other inactive channel name.

Closure Properties

Theorem 2.32 (Extensivity)

$C \subseteq \mathrm{PB} : C \subseteq \mathrm{NAC}_A(C)$

Proof:

Obvious.Consider $\tau = id$.

Theorem 2.33 (Monotonicity)

$C_1 \subseteq C_2 : \ \mathrm{NAC}_A(C_1) \subseteq \mathrm{NAC}_A(C_2)$

Proof:

Obvious.

Theorem 2.34 (Idempotence)

$C \subseteq \mathrm{PB} :$

$\mathrm{NAC}_A(\mathrm{NAC}_A(C)) \ = \ \mathrm{NAC}_A(C)$

Proof:

$$
\begin{aligned}
&\mathrm{NAC}_A(\mathrm{NAC}_A(C)) \ = \\
&= \{f \in \mathrm{PB} \,|\, \exists \tau \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}, f' \in \mathrm{PB} : \exists \tau' \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}, f'' \in F : \\
&\quad \tau'|_A = id \wedge f' = \tau'(f'') \wedge \tau|_A = id \wedge f = \tau(f')\} \qquad = \\
&= \{f \in \mathrm{PB} \,|\, \exists \tau'' \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}, f' \in F : \tau''|_A = id \wedge f = \tau''(f')\}
\end{aligned}
$$

We can see τ'' as $\tau \circ \tau'$.

□

Properties and Consequences

The following property is necessary for our closure; it has to result in correct component functions.

Theorem 2.35 (NAC **preserves Genericity and Contractivity)**

$C \subseteq \mathrm{GCPB}_A, A \subseteq A' : \ \mathrm{NAC}_{A'}(C) \subseteq \mathrm{GCPB}_A$

Proof:

Renaming generally preserves Contractivity.

Input Genericity:

$$\tau(f(\tau^{-1}(\mathrm{ap}^{?}_{\tau(A)}(\phi)))) \stackrel{(\mathrm{L.2.25})}{=} \tau(f(\mathrm{ap}^{?}_{A}(\tau^{-1}(\phi)))) = \stackrel{(\mathrm{generic}_A(f))}{=} \tau(f(\tau^{-1}(\phi)))$$

Output Genericity:

$$\mathrm{ap}^{!N}_{\tau(A)}(\phi)(\tau(f(\tau^{-1}(\phi)))) \stackrel{(\mathrm{L.2.26})}{=} \tau(\mathrm{ap}^{!N}_{A}(\tau^{-1}(\phi))(f(\tau^{-1}(\phi)))) = \stackrel{(\mathrm{generic}_A(f))}{=} \tau(f(\tau^{-1}(\phi)))$$

□

The next theorem shows that renaming does not change anything about name abstractness of a component, but it is also important for other proofs, like that of T.2.38.

Theorem 2.36 (NAC **is Renaming Congruent)**

$C \subseteq \mathrm{GCPB}_A;\, \tau \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}:$
$\quad \tau(\mathrm{NAC}_A(C)) \,=\, \mathrm{NAC}_{\tau(A)}(\tau(C))$

Proof:

$$\begin{aligned}
&\tau(\mathrm{NAC}_A(C)) \,= \\
&= \{\tau(f) \in \mathrm{PB} | \, \exists \tau' \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}, f' \in F : \tau'|_A = id \wedge f = \tau'(f')\} = \\
&= \{f \in \mathrm{PB} | \, \exists \tau' \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}, f' \in F : \tau'|_A = id \wedge \tau^{-1}(f) = \tau'(f')\} = \\
&= \{f \in \mathrm{PB} | \, \exists \tau'' \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}, f'' \in \tau(F) : \tau''|_{\tau(A)} = id \wedge f = \tau''(f'')\} = \\
&= \mathrm{NAC}_A(\tau(C))
\end{aligned}$$

where τ'' can be seen as $\tau \circ \tau' \circ \tau^{-1}$.

□

Corollary 2.37 (Renaming preserves Name Abstractness)

$C \subseteq \mathrm{GCPB}_A;\, \tau \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}: \quad \mathrm{NAC}_A(C) \,=\, C \Rightarrow \mathrm{NAC}_{\tau(A)}(\tau(C)) \,=\, \tau(C)$

Based on the renaming congruence we can proof a very interesting property, which shows the special importance of name abstractness for the role of renamings. Two renamings that behave identical on the active ports are identical in their application on the whole component. In other words, renamings can be characterized by their effect on active ports. On name abstract components, renamings are partial injective functions on the active channels. Now we are closer to the idea of substitution, where the variables of the context also do not matter. But still, injectivity is an unpleasent constraint on renamings which will stay with us.

Theorem 2.38 (Characterization of Renamings)

$$C \subseteq \mathrm{GCPB}_A;\ \mathrm{NAC}_A(C) = C;\ \tau_1, \tau_2 \in \mathrm{N} \xrightarrow{inj.} \mathrm{N} :$$
$$\tau_1|_A = \tau_2|_A \Rightarrow \tau_1(C) = \tau_2(C)$$

Proof:

We define for given renamings $\tau_1, \tau_2 \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}$ with $\tau_1|_A = \tau_2|_A$: $\tau := \tau_2 \circ \tau_1^{-1}$.

$$\begin{aligned} \tau_1|_A = \tau_2|_A &\Rightarrow (\tau_2|_A) \circ (\tau_1|_A)^{-1} = id \\ &\Rightarrow \tau_2|_A \circ (\tau_1^{-1})|_{\tau_1(A)} = id \\ &\Rightarrow \tau|_{\tau_1(A)} = id \end{aligned}$$

Since C is name abstract and with that by C.2.37 also $\tau_1(C)$ is name abstract, we can use τ on $\tau_1(C)$ without effect: $\tau(\tau_1(C)) = \tau_1(C)$. By τ's definition, this guides us to: $\tau_1(C) = \tau(\tau_1(C)) = \tau_2(\tau_1^{-1}(\tau_1(C)) = \tau_2(C)$.

□

2.3.4 Components

Now we have all instruments to specify components. We demand *contractivity* and *genericity* from each function. The set of functions has to be *fully abstract* and *name abstract*.

A component is indexed with two initial sets: the set of active ports and the set of private channels. The union of the active ports and the ports of the private channels form the initially active port set we discussed above. A private channel is only known to the component itself, it should never appear in its environment. This will be ensured by the definitions of the operators, which embed a component in an environment.

For our purposes so far, private channels are simply initially active ports which always appear as pairs. But we will see the sense of privacy when defining the equivalence relation.

There are two more formal conditions for components: First, for the unambiguity of our representation, we demand active ports and the ports of the private channels to be *disjunct*. Second, the component is not partial, so it has to have one output to each input at least. This is equivalent to the existence of at least one behavior: the behavior set of a component has to be *nonempty*.

Set of Components

Definition 2.13 (Set of Components)

$$
\begin{array}{lll}
\multicolumn{3}{l}{C \subseteq \mathrm{B};\ I \subseteq ?!\mathrm{N};\ P \subseteq \mathrm{N}:} \\
\quad C_{I,P} \in \mathrm{Comp} & :\Leftrightarrow & \\
\quad\quad I \cap ?!P = \emptyset & \wedge & \textit{(Disjointness)} \\
\quad\quad C \neq \emptyset & \wedge & \textit{(Nonemptiness)} \\
\quad\quad C \subseteq \mathrm{GCB}_{I \cup ?!P} & \wedge & \textit{(Genericity)} \\
\quad\quad C = \mathrm{FAC}_{I \cup ?!P}(C) & \wedge & \textit{(Full Abstractness)} \\
\quad\quad C = \mathrm{NAC}_{I \cup ?!P}(C) & & \textit{(Name Abstractness)}
\end{array}
$$

Set of Partial Components

Partial behaviors can also form components. Additionally to active ports and private channels, they also have an *available* set of ports, which are guaranteed to be active whenever a behavior ends after a final input. This available set also has to be disjunct to the ports of the private channels.

Definition 2.14 (Set of Partial Components)

$$
\begin{array}{lll}
\multicolumn{3}{l}{C \subseteq \mathrm{PB};\ I, A \subseteq ?!\mathrm{N};\ P \subseteq \mathrm{N}:} \\
\quad C_{I,P,A} \in \mathrm{PComp} & :\Leftrightarrow & \\
\quad\quad I \cap ?!P = A \cap ?!P = \emptyset & \wedge & \textit{(Disjointness)} \\
\quad\quad C \neq \emptyset & \wedge & \textit{(Nonemptiness)} \\
\quad\quad C \subseteq \mathrm{GCPB}_{I \cup ?!P} & \wedge & \textit{(Genericity)} \\
\quad\quad C = \mathrm{FAC}_{I \cup ?!P}(C) & \wedge & \textit{(Full Abstractness)} \\
\quad\quad C = \mathrm{NAC}_{I \cup ?!P \cup A}(C) & \wedge & \textit{(Name Abstractness)} \\
\quad\quad \forall f \in C : \forall \phi \in dom(f) \setminus \mathrm{NMS}: & & \\
\quad\quad\quad A \subseteq \mathrm{ap}_{I \cup ?!P}(\phi)(|\phi|) & & \textit{(Availability)}
\end{array}
$$

Remark: $\mathrm{Comp} \cong \{C_{I,P,A} \in \mathrm{PComp} |\ A = \emptyset \wedge C \subseteq \mathrm{B}\}$

Basic Prefixes We will give two examples for partial components. Taken from the π-calculus, they could be seen as the interpretation of a sender and a receiver prefix (in the context of the synchronous operators).

Definition 2.15 (Sender) $x, y \in \mathrm{N}:$

$$
\begin{aligned}
!x(y) := \{f \in \mathrm{GCPB}_{\{!x,!y,?y\}} |\ & \forall \phi \in dom(f) : \exists n < |\phi| : \\
& (f(\phi)(n)(x) = [y] \vee f(\phi)(n)(x) = \hat{\emptyset}) \wedge \\
& \forall m \neq n, m < |\phi| : \forall c \neq x : f(\phi)(m)(c) = \hat{\emptyset}\}_{\{!x,!y,?y\},\emptyset,\emptyset}
\end{aligned}
$$

Definition 2.16 (Receiver) $x, y \in \mathrm{N}$:

$$
\begin{aligned}
?x(y) := \{f \in \mathrm{GCPB}_{\{?x\}} \mid \forall \phi \in dom(f), n < |\phi|, c \in \mathrm{N} : \\
f(\phi)(n)(c) = \hat{\emptyset} \wedge \\
((\exists n < |\phi| : ?y, !y \in \phi(n)(x)) \vee |\phi| = \infty\}_{?x, \emptyset, \{?y, !y\}}
\end{aligned}
$$

Equivalence

Now we see a first use of the private set: it can be renamed without altering the component essentially. Obviously it is formally altered, but we abstract from those 'inessential' differences by identifying a component with its equivalence class of components which result from a private renaming of the component. With private renaming we denote any renaming $\tau \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}$ with $\tau|_P = id$ for the private set P. Remember that τ's effect on the initially active ports characterizes the renaming (T.2.38), which means for this case that the restriction of the renaming on the private channels is all the necessary information to connect a component with an equivalent one.

Definition 2.17 (Equivalence)

$$
\begin{aligned}
& C_{1_{I_1, P_1}} \in \mathrm{Comp}_{I_1 \cup ?!P_1}; C_{2_{I_2, P_2}} \in \mathrm{Comp}_{I_2 \cup ?!P_2}, \\
& P_1 \cap P_2 = ?!P_1 \cap I_2 = ?!P_2 \cap I_1 = \emptyset : \\
& \quad C_{1_{I_1, P_1}} \equiv C_{2_{I_2, P_2}} :\Leftrightarrow \exists \tau \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}; \tau|_I = id : \ \tau(C_{1_{I_1, P_1}}) = C_{2_{I_2, P_2}}
\end{aligned}
$$

Now we have completely defined components and their equivalence structure, but this is just half of our model. Of equal importance are the possiblilities of combining networks via operators. We will study the interaction between the defined components by discussing the methods of their combination.

Chapter 3

Operators

3.1 Genericity

Genericity is also a property of operators, it includes that the operator's result has to be generic if its arguments are, but there is more to genericity than that. The result of an operation has to be a proper component, as described by the last chapter. Our main work in this chapter will be the proof of genericity for different operator definitions.

Any operator has to fulfill two conditions to be considered *generic*: It has to result in a component when applied on components, and it has to be a congruent operation with respect to $\equiv$:

$$\forall i \in I : A_i \equiv A'_i \Rightarrow \bigoplus_{i \in I} A_i \equiv \bigoplus_{i \in I} A'_i$$

As stated before, we want to see components abstractly, independent of the concrete names of the private channels. However, a certain component has to represent its equivalence class in the following operations. We constrain this choice on components with disjunct private sets.

The necessary properties for genericity are proofed in detail by showing the following:

Disjointness of the result's initially active and private ports.

Nonemptiness of the result's behavior set.

Generictity of the result's behaviors.

Full Abstractness of the result's behavior set.

Renaming Congruence of the operation. This is commonly needed to prove the following properties.

Name Abstractness of the result's behavior set.

Congruence of the operation to the equivalence relation '$\equiv$'.

3.2 Union

The union of a set of components forms the union of the possible input output pairs of the components. For any input the union reacts like one of the argument components, which are not interacting with each other.

We cannot simply form the union of the behavior sets because there can be contractive, generic behaviors reacting like one component to one input and like another to a different one. As an example, take the union of different components which each map all their inputs constantly on different outputs. With regards for contractivity and genericity there possibly have to be non constant functions in the result. These functions are added by the fully abstract closure in the definition:

3.2.1 Definition

Definition 3.1 (Union of Components)

$I \neq \emptyset$; $\forall i \in I : C_{i\,I_i,P_i} \in \text{Comp}$; $\forall i, j \in I, i \neq j : P_i \cap P_j = I_i \cap ?!P_j = \emptyset$:

$$\bigcup_{i\in I} C_{i\,I_i,P_i} := (\text{FAC}_{\bigcup_{i\in I} I_i \cup ?!P_i}(\bigcup_{i\in I} C_i))_{\bigcup_{i\in I} I_i, \bigcup_{i\in I} P_i}$$

For $I = \emptyset$ we would need a set of behaviors as result which should be neutral with respect to the union: $\bigcup_{i\in I} C_{i\,I_i,P_i} = \bigcup_{i\in I} C_{i\,I_i,P_i} \cup \bigcup_{i\in\emptyset} C_{i\,I_i,P_i}$.

There is no general solution because such a set would have to be subset of every component. Nevertheless, there could be examples for special cases.

3.2.2 Disjointness

Because of the premise $\forall i, j \in I, i \neq j :\ I_j \cap ?!P_i = \emptyset$ and with $\forall i \in I : C_{i\,I_i,P_i} \in \text{Comp} \Rightarrow I_i \cap ?!P_i$, disjointness is obviously guaranteed:

$$(\bigcup_{i\in I} I_i) \cap ?!(\bigcup_{i\in I} P_i) = \bigcup_{i,j\in I} I_i \cap ?!P_i = \emptyset$$

3.2.3 Nonemptiness

Consider:

$$\forall i \in I : \emptyset \neq C_i \subseteq \text{GCB}_{I_i \cup ?!P_i} \overset{(\text{T.2.9})}{\Rightarrow} \emptyset \neq C_i \subseteq \text{GCB}_{\bigcup_{i\in I} I_i \cup ?!P_i} \Rightarrow$$

$$\overset{(\text{T.2.20,2.21})}{\Rightarrow} \emptyset \neq C_i \subseteq \text{FAC}_{\bigcup_{i\in I} I_i \cup ?!P_i}(C_i) \subseteq \text{FAC}_{\bigcup_{i\in I} I_i \cup ?!P_i}(\bigcup_{i\in I} C_i)$$

3.2.4 Genericity

The fully abstract closure enforces genericity (see Def.2.10).

3.2.5 Full Abstractness

Full abstractness is trivial by the form of the definition, the fully abstract closure is idempotent:

$$\mathrm{FAC}_{\bigcup\limits_{i\in I} I_i\cup ?!P_i}(\mathrm{FAC}_{\bigcup\limits_{i\in I} I_i\cup ?!P_i}(\bigcup_{i\in I} C_i)) \overset{(\mathrm{T.2.22})}{=} \mathrm{FAC}_{\bigcup\limits_{i\in I} I_i\cup ?!P_i}(\bigcup_{i\in I} C_i)$$

3.2.6 Renaming Congruence

This property is not directly needed for genericity, but its essential for the following two proofs. Name abstractness and congruence to the equivalence relation is obvious after proving renaming congruence. Besides that, the definitions of the operations (including renaming itself) are justified as long as they are in agreement with each other.

Theorem 3.1 (Union is Renaming Congruent)

$\tau \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}:$

$$\tau(\bigcup_{i\in I} C_{i\,I_i,P_i}) = \bigcup_{i\in I} \tau(C_i)_{\tau(I_i),\tau(P_i)}$$

Proof:

$$\begin{aligned}\tau(\bigcup_{i\in I} C_{i\,I_i,P_i}) &= \tau((\mathrm{FAC}_{\bigcup\limits_{i\in I} I_i\cup ?!P_i}(\bigcup_{i\in I} C_i))_{\bigcup\limits_{i\in I} I_i,\bigcup\limits_{i\in I} P_i}) \overset{(\mathrm{T.2.30})}{=} \\ &= (\mathrm{FAC}_{\tau(\bigcup\limits_{i\in I} I_i\cup ?!P_i)}(\tau(\bigcup_{i\in I} C_i)))_{\tau(\bigcup\limits_{i\in I} I_i),\tau(\bigcup\limits_{i\in I} P_i)} = \\ &= (\mathrm{FAC}_{\bigcup\limits_{i\in I} \tau(I_i)\cup\tau(?!P_i))}(\bigcup_{i\in I} \tau(C_i)))_{\bigcup\limits_{i\in I} \tau(I_i)),\bigcup\limits_{i\in I} \tau(P_i))} = \bigcup_{i\in I} \tau(C_{i\,I_i,P_i})\end{aligned}$$

□

3.2.7 Name Abstractness

Assume $\tau \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}$, $\tau|_{\bigcup\limits_{i\in I} I_i\cup ?!P_i} = id$, then $\tau|_{I_i\cup ?!P_i} = id$ for any $i \in I$:

$$\tau(\bigcup_{i\in I} C_{i\,I_i,P_i}) = \bigcup_{i\in I} \tau(C_i)_{\tau(I_i),\tau(P_i)} \overset{(\mathrm{NAC}_{I_i\cup ?!P_i}(C_i) = C_i)}{=} \bigcup_{i\in I} C_{i\,I_i,P_i}$$

This proves $\mathrm{NAC}_{\bigcup\limits_{i\in I} I_i\cup ?!P_i}(\bigcup_{i\in I} C_i) \subseteq \bigcup_{i\in I} C_i$. The opposite direction '$\supseteq$' is shown by theorem 2.32.

3.2.8 Congruence

We assume: $\forall i \in I : C_{i\,I_i,P_i} \equiv C'_{i\,I_i,P'_i}$, where

$\forall i \in I : \tau_i \in \mathrm{N} \xrightarrow{inj.} \mathrm{N} \wedge \tau_i|_{I_i} = id \wedge \tau_i(C_{i\,I_i,P_i}) = C'_{i\,I_i,P'_i}$

Since the union is applicable on the indexed components as on their indexed images, we know that: $\forall i,j \in I : i \neq j \Rightarrow P_i \cap P_j = \emptyset \wedge \tau_i(P_i) \cap \tau_j(P_j) = \emptyset$

We will have to find a $\tau \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}, \tau|_{\bigcup\limits_{i \in I} I_i \cup ?!P_i} = id$ which maps the union of the indexed components on the union of their images:

$$\tau(\bigcup_{i \in I} C_{i\,I_i,P_i}) = \bigcup_{i \in I} C'_{i\,I_i,P'_i}$$

Assume the following for τ: $\tau \in \mathrm{N} \xrightarrow{inj.} \mathrm{N} \wedge \forall i \in I : \tau|_{I_i} = id \wedge \tau|_{P_i} = \tau_i|_{P_i}$

The existence of such a τ is guaranteed by the disjointness of the private sets and by that of their images. Now we apply such a τ on the union:

$$\tau(\bigcup_{i \in I} C_{i\,I_i,P_i}) = \bigcup_{i \in I} \tau(C_i)_{\tau(I_i),\tau(P_i)} = \bigcup_{i \in I} C'_{i\,I_i,P'_i} \Rightarrow \bigcup_{i \in I} C_{i\,I_i,P_i} \equiv \bigcup_{i \in I} C'_{i\,I_i,P'_i}$$

3.3 Hiding

Hiding locally binds some variable inside a network term. The effect is that the appearances of the hidden name in the environment of the term cannot have the same meaning, i.e., denote the same channel. Disjointness of the private set of the resulting component and the initially active ports of the environment enforce a renaming of the hidden name before applying any further combinators on them.

Privacy of the component prohibits hiding an already private channel name. [1] As a consequence we also cannot hide any completely inactive name, because it could be active in a privately renamed version of the component. As an effect of this, the definition is relatively cumbersome by distinguishing the case of hiding inactive names.

3.3.1 Definition

Definition 3.2 (Hiding)

$C_{I,P} \in \mathrm{Comp},\ x \in N \setminus P$:

$$\nu x : C_{I,P} := \begin{cases} (\mathrm{FAC}_{I \cup ?!P \cup \{?x,!x\}}(C))_{I \setminus \{?x,!x\}, P \cup \{x\}} & \textit{if } x \in \tilde{I} \\ C_{I,P} & \textit{else} \end{cases}$$

We do not assume that $?x \in I$ or $!x \in I$ before hiding it. Only for the case $!x \in I$ and $?x \notin I$ the fully abstract closure in the definition above matters. In that case, since $x \notin P$ is assumed, the component should be fully abstract with respect to actually ignored inputs on x.

We omit a proof of disjointness in this and many following cases, where it is obvious from the definiton.

[1] The subtle and rather technical reason lies in the congruence of hiding. The hiding of an already private name would need to be equivalent to the same hiding applied on a privately renamed component to which the name would be completely unknown. But, the resulting private sets would be of different cardinality and so could not be equivalent via a bijective renaming function!

3.3.2 Nonemptiness

In case $x \in \tilde{I}$:

$$\emptyset \neq C \subseteq \mathrm{GCB}_{I\cup ?!P} \overset{(\mathrm{T}.2.9)}{\subseteq} \mathrm{GCB}_{I\cup ?!P\cup\{?x,!x\}} \overset{(\mathrm{T}.2.20)}{\Longrightarrow} C \subseteq \mathrm{FAC}_{I\cup ?!P\cup\{?x,!x\}}(C)$$

If $x \notin \tilde{I}$ the resulting behavior set is C, which is nonempty by assumption.

3.3.3 Renaming Congruence

Theorem 3.2 (Hiding is Renaming Congruent)

$\tau \in \mathrm{N} \overset{inj.}{\longrightarrow} \mathrm{N}$:

$$\tau(\nu x : C_{I,P}) = \nu\tau(x) : \tau(C_{I,P})$$

Proof:

$$\begin{aligned}
\tau(\nu x : C_{I,P}) &= \tau(\mathrm{FAC}_{I\cup ?!P\cup\{?x,!x\}}(C))_{\tau(I\setminus\{?x,!x\}),\tau(P\cup\{x\})} = \\
&\overset{(\mathrm{T}.2.30)}{=} \mathrm{FAC}_{\tau(I)\cup ?!\tau(P)\cup\{?\tau(x),!\tau(x)\}}(\tau(C))_{\tau(I)\setminus\{?\tau(x),!\tau(x)\},\tau(P)\cup\{\tau(x)\}} = \\
&= \nu\tau(x) : \tau(C_{I,P})
\end{aligned}$$

□

3.3.4 Name Abstractness

$\tau \in \mathrm{N} \overset{inj.}{\longrightarrow} \mathrm{N}$, $\tau|_{I\cup ?!P} = id$:

1st case: $x \in \tilde{I}$ ($\Rightarrow \tau(x) = x$): $\tau(\nu x : C_{I,P}) \overset{(\mathrm{T}.3.2)}{=} \nu x : \tau(C_{I,P}) = \nu x : C_{I,P}$ – the last equality follows from the name abstractness of $C_{I,P}$ and from $\tau|_{I\cup ?!P} = id$.

2nd case: $x \notin \tilde{I}$ ($\Rightarrow \tau(x) \notin \tilde{I}$): $\tau(\nu x : C_{I,P}) \overset{(\mathrm{T}.3.2)}{=} \nu\tau(x) : \tau(C_{I,P}) = C_{I,P} = \nu x : C_{I,P}$

3.3.5 Congruence

Again, the only interesting case is $x \in \tilde{I}$, let us assume: $\tau(C_{I,P}) = C'_{I,P'}$ with $\tau|_I = id$:

$\tau(\nu x : C_{I,P}) \overset{(\mathrm{T}.3.2)}{=} \nu x : \tau(C_{I,P}) = \nu x : C'_{I,P'}$.

This proves: $\nu x : C_{I,P} \equiv \nu x : C'_{I,P'}$.

3.4 Concatenation

We can concatenate partial components to form new components. The initially active ports of the result does not include the ports which were made available by the first operand.

3.4.1 Definition

Definition 3.3 (Concatenation of Partial Components)

$C1_{I_1,P_1,A_1}, C2_{I_2,P_2,A_2} \in \text{PComp}$, $P_1 \cap P_2 = \emptyset$, $(I_1 \cup A_1) \cap ?!P_2 = (I_2 \cup A_2) \cap ?!P_1 = \emptyset$:

$$\begin{aligned} C1_{I_1,P_1,A_1} \circ C2_{I_2,P_2,A_2} := \; & \text{FAC}_{I_1 \cup (I_2 \setminus A_1) \cup ?!(P_1 \cup P_2)}(\\ & \text{NAC}_{I_1 \cup (I_2 \setminus A_1) \cup ?!(P_1 \cup P_2) \cup A_2}(\\ & (C_1 \circ C_2)))_{I_1 \cup (I_2 \setminus A_1), P_1 \cup P_2, A_2} \end{aligned}$$

Since the result is still a partial component, we have to exclude A_2 from the name abstract closure. In contrast to this, we can abstract away from the names of A_1 because in the concatenation these names lost there exemplary role for C_2.

3.4.2 Disjointness

$$(I_1 \cup (I_2 \setminus A_1) \cup A_2) \cap (P_1 \cup P_2) \subseteq (I_1 \cup I_2 \cup A_2) \cap (P_1 \cup P_2) =$$

$$= (I_1 \cap P_2) \cup (I_1 \cap P_1) \cup ((I_2 \cup A_2) \cap (P_2)) \cup ((I_2 \cup A_2) \cap P_1) = \emptyset$$

3.4.3 Nonemptiness and Genericity

We will show that $\emptyset \neq C_1 \circ C_2 \subseteq \text{GCPB}_{I_1 \cup (I_2 \setminus A_1) \cup ?!(P_1 \cup P_2)}$ using theorem 2.1 and 2.19, which state that concatenation of behaviors preserves contractivity and genericity.

Assume $f \in C_1, g \in C_2$: Since f and g are contractive each, $f \circ g$ is contractive (see T. 2.1).

$$f \in \text{GCPB}_{I_1 \cup ?!P_1} \overset{(T.2.8)}{\Longrightarrow} \text{generic}_{I_1 \cup (I_2 \setminus A_1) \cup ?!(P_1 \cup P_2)}(f)$$

$$g \in \text{GCPB}_{I_2 \cup ?!P_2} \Longrightarrow \text{generic}_{I_2 \cup ?!P_2}(g)$$

Availabiltity of $C1_{I_1,P_1,A_1}$ ensures that

$$\forall \phi \in dom(f) \setminus \text{NMS} : I_2 \cup ?!P_2 \subseteq (I_2 \setminus A_1) \cup ?!P_2 \cup A_1 \subseteq \text{ap}_A(\phi)(|\phi|)$$

Hence all premises of theorem 2.19 are fulfilled, we can conclude that $\text{generic}_{I_1 \cup (I_2 \setminus A_1) \cup ?!(P_1 \cup P_2)}(f \circ g)$. Together with contractivity this sums up to $f \circ g \in \text{GCPB}_{I_1 \cup (I_2 \setminus A_1) \cup ?!(P_1 \cup P_2)}$.

According to theorem 2.32 and 2.20 the name abstract and the fully abstract closure are extensive, which makes the nonempty, contractive, and generic $C_1 \circ C_2$ also a subset of $\text{FAC}_{I_1 \cup (I_2 \setminus A_1) \cup ?!(P_1 \cup P_2)}(\text{NAC}_{I_1 \cup (I_2 \setminus A_1) \cup ?!(P_1 \cup P_2) \cup A_2}(C_1 \circ C_2))$. Its nonemptiness is proofed with that.

The result of a name abstract closure remains a subset of $\text{GCPB}_{I_1 \cup (I_2 \setminus A_1) \cup ?!(P_1 \cup P_2)}$ according to theorem 2.35. Obviously the same is valid for the fully abstract closure (cf. its definition 2.10), which proves the genericity of concatenation.

3.4.4 Full and Name Abstractness

Keep in mind that the name abstractness of the result is, because of availability, defined with respect to the active set enlarged by the finally available set A_2. A partial component can receive a *certain* channel name, although this name is not initially active.

The following theorem shows that we applied the two closures in the right order since both, full and name abstractness, are ensured:

Theorem 3.3 (Fully and Name Abstract Closure)

$C \subseteq \mathrm{GCPB}_A, A \subseteq A'$:

$$1)\ \mathrm{NAC}_{A'}(\mathrm{FAC}_A(\mathrm{NAC}_{A'}(C))) = \mathrm{FAC}_A(\mathrm{NAC}_{A'}(C)) \wedge$$
$$2)\ \mathrm{FAC}_A(\mathrm{FAC}_A(\mathrm{NAC}_{A'}(C))) \ = \mathrm{FAC}_A(\mathrm{NAC}_{A'}(C))$$

Proof:

$1) \tau|_{A'} = id \Rightarrow$

$\tau(\mathrm{FAC}_A(\mathrm{NAC}_{A'}(C))) = \mathrm{FAC}_A(\tau(\mathrm{NAC}_{A'}(C))) = \mathrm{FAC}_A(\mathrm{NAC}_{A'}(C))$

$2)\ \mathrm{NAC}_{A'}(C) \subseteq \mathrm{GCPB}_A \overset{(\mathrm{T}.2.22)}{\Rightarrow}$

$\mathrm{FAC}_A(\mathrm{FAC}_A(\mathrm{NAC}_{A'}(C))) = \mathrm{FAC}_A(\mathrm{NAC}_{A'}(C))$

□

For convenience and in compliance with the theorem we abbreviate $I_1 \cup (I_2 \setminus A_1) \cup ?!(P_1 \cup P_2)$ with A and its superset $I_1 \cup (I_2 \setminus A_1) \cup ?!(P_1 \cup P_2) \cup A_2$ with A'.

As we saw in the last section, $\mathrm{NAC}_{A'}(C_1 \circ C_2)$ is a subset of GCPB_A. This makes theorem 3.3 applicable:

$$\mathrm{FAC}_A(\mathrm{FAC}_A(\mathrm{NAC}_{A'}(C_1 \circ C_2))) \overset{(\mathrm{T}.3.3)}{=} \mathrm{FAC}_A(\mathrm{NAC}_{A'}(C_1 \circ C_2))$$

Now we apply theorem 3.3 to our special case of name abstractness:

$$\mathrm{NAC}_{A \cup A_2}(\mathrm{FAC}_A(\mathrm{NAC}_{A \cup A_2}(C_1 \circ C_2))) \overset{(\mathrm{T}.3.3)}{=} \mathrm{FAC}_A(\mathrm{NAC}_{A \cup A_2}(C_1 \circ C_2))$$

3.4.5 Availability

In this case, where the result is a partial component, availability (D.2.14) has to be shown:

Assume that $h \in C_1 \circ C_2$ with $h = f \circ g, f \in C_1$ and $g \in C_2$ and that $\phi \in dom(h)$. There exist $\phi_1 \in dom(f)$ and $\phi_2 \in dom(g)$ such that $\phi = \phi_1 \circ \phi_2$.

Now according to lemma 2.16 (A stands for $I_1 \cup (I_2 \setminus A_1) \cup ?!(P_1 \cup P_2)$) : $ap_A(\phi)(|\phi_1| + |\phi_2|) = ap_A(\phi) \uparrow^{|\phi_1|} (|\phi_2|) = ap_{ap_A(\phi_1)(|\phi_1|)}(\phi_2)(|\phi_2|)$, which is, by the availability, of C_1 a superset of $ap_{I_2 \cup ?!P_2}(\phi_2)(|\phi_2|)$ which includes A_2 because of the availability of C_2.

By this, each domain element with its value for each function in $C_1 \circ C_2$ corresponds to our availability demand. The applied name abstract closure adds functions with changed names, but it still respects the initial sets *together with* A_2. Theorem 2.24 makes it easy to see why A_2 will still be available:

$$A_2 \subseteq ap_A(\phi)(|\phi_1| + |\phi_2|) \overset{(\tau|_{A_2} = id)}{\Rightarrow}$$
$$A_2 \subseteq \tau(ap_A(\phi)(|\phi_1| + |\phi_2|)) \overset{(\mathrm{T.2.24})}{=} ap_{\tau(A)}(\tau(\phi))(|\phi_1| + |\phi_2|))$$

The application of the fully abstract closure does not interfere with availability because it is a property of domain elements and their values. FAC does not add pairs to this input output relation, it just recombines them.

Thus, each function in $\mathrm{FAC}_A(\mathrm{NAC}_{A'}(C_1 \circ C_2))$ makes the names in A_2 available.

3.5 Merging, Copying and Splitting

One of the main design decisions we are confronted with lays in our treatment of stream ramifications. When several components interact with each other, we have to handle multiple input and output streams for each component. Under the constraint of a fixed functionality we have to explicitly model the transition from several to one stream and reverse.

For the 'two on one' case we simply use the multi set union on each channel and in each unit, which *merges* the streams without adding or dropping any messages. There are alternative solutions like, for example, nondeterministically choosing one multiset per channel and time unit and dropping the other. In fact this example has some algebraic merits, depending on other properties of the chosen design. Nevertheless we constrain our examination on the 'fair merge' above [2]

For the 'one on two' case we offer two different solutions which, in combination with the fair merge, realize *asynchronous* and *synchronous* communication.

3.5.1 Asynchronous Communication

In the context of the distinction we are trying to make, *asynchronous communication* will be understood as an exclusive procedure of interaction.

After *broadcasting* its message, the sending component continues directly without awaiting acknowledgements (this is the normal case). Every other component with read access on the respective channel receives the message, i.e., it may causally be influenced in its further behavior by the message. As the term 'broadcast suggests, there is neither a sole receiver nor, in consequence, a standardized acknowledgement procedure.

Asynchronous interaction does not involve the possibility of a *failure*: Communication includes nothing more than the act of sending a message — a reaction is always possible but contingent upon the concrete behaviors of the partner components.

[2] In the case of list representation instead of multisets there are more alternatives which we have to consider. Especially with regard to the algebraic problems arising.

3.5.2 Realization

We model this process by networks based on a 'copy split'. In ramifications where one stream is divided into two we simply 'copy' the stream by using it as input for different functions. The subterm which denotes the stream appears several times within one term.

As an example consider the asynchronous parallel composition (Def.3.9):

$f(\phi) = f_1(\phi) \uplus f_2(\phi))$

The input stream ϕ is simply used twice, each time as argument for a different function.

3.5.3 Synchronous Communication

In a synchronous network, messages are *never* duplicated. A message has one sender and one receiver with the necessary channel access. Other components which read the channel besides the receiver will not be influenced by the message.

Synchronizity in another respect, time synchronizity, can easily be introduced through a protocol which demands a correct acknowledgment transmission for each message.

The described message synchronizity (one message – one sender – one receiver) is realized by an explicit stream splitting function which will neither add nor drop any message. It will nondeterministically distribute messages on both output streams. If a message is put on a channel which cannot be received by the processing component, it is lost. Synchronous communication may fail.

3.5.4 Realization

Because there are many possibilities to split a stream, the distributing function is realized as a set of functions on streams.

The following two properties are needed to ease later proofs. They do not constrain networks in any way because the functions are used pointwise in defining the networks. For each input there will have to be a splitting function – not just one function for every input.

Pointwise defined functions are functions that work on each channel's and each time unit's content of the input independent of all other channels and time units. Its result on one channel and unit can be predicted from the input on exactly this channel and unit.

Definition 3.4 (Pointwise Defined Functions: PWD**)**

$n, m \in I\!N \setminus \{0\},\ f \in \mathrm{NMS}^n \longrightarrow \mathrm{NMS}^m$:

$$\begin{aligned} \mathrm{PWD}(f) :\Leftrightarrow\ & \forall \phi_1, \ldots, \phi_n, \psi_1, \ldots, \psi_n, m \in I\!N, c \in \mathrm{N} : \\ & (\forall i \leq n : \phi_i(m)(c) = \psi_i(m)(c)) \Rightarrow \\ & f(\phi_1, \ldots, \phi_n)(m)(c) = f(\psi_1, \ldots, \psi_n)(m)(c) \end{aligned}$$

A pointwise defined function can be seen as a bundle of functions on multisets, indexed by channel names and time units:

$\mathrm{PWD}(f) : \exists (f_{n,c})_{n \in \mathbb{N}, c \in \mathrm{N}} :$

$\forall n \in \mathbb{N}, c \in \mathrm{N} : \forall \phi \in \mathrm{NMS} : f(\phi)(n)(c) = f_{n,c}(\phi(n)(c))$

Filter Congruence is the more interesting constraint for splits. It connects the behavior of any split on any given input with its behavior on the respective filtered inputs. Again, this constraint does not ban any argument value pairs, it only limits the possible combinations within one function.

Definition 3.5 (Filter Congruent Functions: FC**)**

$n, m \in \mathbb{N} \setminus \{0\},\ f \in \mathrm{NMS}^n \longrightarrow \mathrm{NMS}^m :$

$\mathrm{FC}(f) :\Leftrightarrow$

$$\forall \phi_1, \ldots, \phi_n \in \mathrm{NMS}, F \subseteq M : F \copyright f(\phi_1, \ldots, \phi_n) = f(F \copyright \phi_1, \ldots, F \copyright \phi_n)$$

Notation We will write f^n for $\pi_n \circ f$, i.e the nth projection which can be applied on the tuple result of a function f. For splits s that means: $s(\phi) = (s^1(\phi), s^2(\phi))$.

Definition 3.6 (Splits)

$$\begin{aligned} \mathrm{Splits} := \{ & s \in \mathrm{NMS} \longrightarrow \mathrm{NMS} \times \mathrm{NMS} \mid \mathrm{PWD}(f) \wedge \mathrm{FC}(f) \wedge \\ & \forall \phi \in \mathrm{NMS}, n \in \mathbb{N}, c \in \mathrm{N} : \phi(n)(c) = f^1(\phi)(n)(c) \uplus f^2(\phi)(n)(c) \} \end{aligned}$$

Theorem 3.4 (Splits are nonexpansive)

$s \in \mathrm{Splits}, \phi, \phi' \in \mathrm{NMS}, n \in \mathbb{N} :$

$$\phi \downarrow_n = \phi' \downarrow_n \implies s^1(\phi) \downarrow_n = s^1(\phi') \downarrow_n \wedge s^2(\phi) \downarrow_n = s^2(\phi') \downarrow_n$$

Proof:

Obvious because splits are pointwise defined.

□

Theorem 3.5 (Splits **is name abstract)**

$\tau \in \mathrm{N} \xrightarrow{inj.} \mathrm{N} :$

$\forall s \in \mathrm{Splits} : \exists s' \in \mathrm{Splits} : \forall \phi \in \mathrm{NMS} :$
$s'^1(\phi) = \tau(s^1(\tau^{-1}(\phi)) \wedge s'^2(\phi) = \tau(s^2(\tau^{-1}(\phi))$

Remark: $\tau(s^i(\alpha)) = s^{i'}(\tau(\alpha)) \Rightarrow$
$\tau^{-1}(s^{i'}(\alpha)) = \tau^{-1}(s^{i'}(\tau(\tau^{-1}(\alpha)))) = s^{i'}(\tau^{-1}(\alpha))$

Proof:

Let $\tau \in \mathrm{N} \xrightarrow{inj.} \mathrm{N}; s \in \mathrm{Splits}; \phi_1, \phi_2 \in \mathrm{NMS}; n \in \mathbb{N}; c \in \mathrm{N}; i \in \{1, 2\}$:

1) Pointwise Definedness

Assume $\phi_1(n)(c) = \phi_2(n)(c)$:

$$\begin{array}{cl} \Rightarrow & \tau^{-1}(\phi_1)(n)(\tau^{-1}(c)) = \tau^{-1}(\phi_2)(n)(\tau^{-1}(c))) \quad \Rightarrow \\ \stackrel{(\mathrm{PWD}(s^i))}{\Rightarrow} & s^i(\tau^{-1}(\phi_1))(n)(\tau^{-1}(c)) = s^i(\tau^{-1}(\phi_2))(n)(\tau^{-1}(c)) \Rightarrow \\ \Rightarrow & \tau(s^i(\tau^{-1}(\phi_1)))(n)(c) = \tau(s^i(\tau^{-1}(\phi_2)))(n)(c) \end{array}$$

2) Filter Congruence

$$\begin{array}{cll} & F©\tau(s^i(\tau^{-1}(\phi))) & = \\ = & \tau(\tau^{-1}(F)©s^i(\tau^{-1}(\phi))) & = \\ \stackrel{(\mathrm{FC}(s^i))}{=} & \tau(s^i(\tau^{-1}(F)©\tau^{-1}(\phi))) & = \\ = & \tau(s^i(\tau^{-1}(F©\phi))) & \end{array}$$

3) Merging of Lists

$$\begin{array}{lll} \tau^{-1}(\phi)(n)(\tau^{-1}(c)) & \in \mathrm{LM}(s^1(\tau^{-1}(\phi))(n)(\tau^{-1}(c)), s^1(\tau^{-1}(\phi))(n)(\tau^{-1}(c))) & \Rightarrow \\ \tau(\tau^{-1}(\phi)(n)(\tau^{-1}(c))) & \in \tau(\mathrm{LM}(s^1(\tau^{-1}(\phi))(n)(\tau^{-1}(c)), s^1(\tau^{-1}(\phi))(n)(\tau^{-1}(c))) & \Rightarrow \\ \phi(n)(c) & \in \mathrm{LM}(\tau(s^1(\tau^{-1}(\phi)))(n)(c)), \tau(s^1(\tau^{-1}(\phi)))(n)(c)))) & \end{array}$$

□

3.5.5 Properties

We show two properties of merges and splits, which illustrate their congruence relation to output filters. Strong use of these theorems will be made later to prove genericity of operations using splits and merges.

Theorem 3.6 ($\mathrm{ap}^{!N}$ **is congruent to merges)**

$\phi, \alpha, \beta \in \mathrm{NMS}, A \subseteq \mathrm{N}$:

$$\mathrm{ap}^{!N}_A(\phi)(\alpha \uplus \beta) = \mathrm{ap}^{!N}_A(\phi)(\alpha) \uplus \mathrm{ap}^{!N}_A(\phi)(\beta)$$

Proof:

Let $n \in \mathbb{N}, c \in \mathrm{N}$:

$$\begin{array}{cll} & \mathrm{ap}^{!N}_A(\phi)(\alpha \uplus \beta)(n)(c) & = \\ = & \begin{cases} \mathrm{ap}_A(\phi)(n)©(\alpha \uplus \beta)(n)(c) & \text{if } !c \in \mathrm{ap}_A(\phi)(n) \\ \epsilon & \text{else} \end{cases} & = \\ \stackrel{(\mathrm{FC}(m))}{=} & \begin{cases} (\mathrm{ap}_A(\phi)(n)©\alpha) \uplus (\mathrm{ap}_A(\phi)(n)©\beta)(n)(c) & \text{if } !c \in \mathrm{ap}_A(\phi)(n) \\ \epsilon & \text{else} \end{cases} & = \\ = & (\mathrm{ap}^{!N}_A(\phi)(\alpha) \uplus \mathrm{ap}^{!N}_A(\phi)(\beta))(n)(c) & \end{array}$$

□

Theorem 3.7 ($\mathrm{ap}^{!N}$ **is congruent to splits)**

$s \in \mathrm{Splits}, i \in \{1, 2\}; \phi, \psi \in \mathrm{NMS}; A \subseteq \mathrm{N}$:

$\mathrm{ap}_A^{!N}(\phi)(s^i(\psi)) = s^i(\mathrm{ap}_A^{!N}(\phi)(psi))$

Proof:

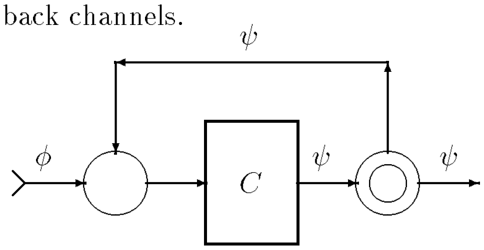
Let $n \in \mathbb{N}, c \in \mathrm{N}$:

$$
\begin{array}{ll}
 & \mathrm{ap}_A^{!N}(\phi)(s^i(\psi))(n)(c) \qquad = \\
= & \begin{cases} \mathrm{ap}_A(\phi)(n) \copyright s^i(\psi)(n)(c) & \text{if } !c \in \mathrm{ap}_A(\phi)(n) \\ \epsilon & \text{else} \end{cases} = \\
\stackrel{(\mathrm{FC}(s))}{=} & \begin{cases} s^i(\mathrm{ap}_A(\phi)(n) \copyright \psi)(n)(c) & \text{if } !c \in \mathrm{ap}_A(\phi)(n) \\ \epsilon & \text{else} \end{cases} = \\
\stackrel{(\mathrm{PWD}(s))}{=} & s^i(\mathrm{ap}_A^{!N}(\phi)(\psi))(n)(c)
\end{array}
$$

3.6 Asynchronous Feedback

With our components' behaviors being contractive, we have exactly one solution for recursive equations. This means that a behavior can be applied on its own output, or seen as a parameterized fixed point equation, it can be applied to any input which *nonexpansively* depends on the output. Consider that the composition of a nonexpansive and a contractive function is contractive.

The *asynchronous feedback* applied on a component merges the output of the component with its input. For any input, the output will be the fixed point of a behavior of the component. The initial interface of a component remains the same, especially the component will still be able to communicate with its environment on feedback channels.



3.6.1 Definition

Definition 3.7 (Asynchronous Feedback)

$C_{I,P} \in \mathrm{Comp}$:

$$\mu(C_{I,P}) := \{f \in \mathrm{GCB}_{I\cup?!P} | \forall \phi : \exists g \in C : f(\phi) = \psi, \psi = f'(\phi \uplus \psi)\}_{I,P}$$

3.6.2 Nonemptiness

We will show that all functions that are actually defined by one $g \in C$ for all inputs $\phi \in \mathrm{NMS}$ are contractive and generic. Since C is nonempty, this implies that the asynchronous feedback of C is also nonempty.

The lemmas below will serve to prove the asynchronous and the more complicated synchronous case. Therefore they might seem exaggeratedly general for the easy asynchronous case.

The first lemma will show that for the active ports stream the feedback stream does not matter because it accords to the active ports constraint itself.

Lemma 3.8

$\phi, \psi, \psi' \in \mathrm{NMS}, \psi' \subseteq \psi;\ A_1 \subseteq A_2 \subseteq \mathrm{N}:$

$$\psi = \mathrm{ap}^{!N}_{A_1}(\phi \uplus \psi')(\psi) \Rightarrow \mathrm{ap}_{A_2}(\phi \uplus \psi') = \mathrm{ap}_{A_2}(\phi)$$

Proof:

We assume $\psi = \mathrm{ap}^{!N}_{A_1}(\phi \uplus \psi')(\psi)$ and prove inductively.

n=0:

$$\mathrm{ap}_{A_2}(\phi \uplus \psi')(0) = A_2 = \mathrm{ap}_{A_2}(\phi)(0)$$

n+1:

$$\begin{aligned} & \mathrm{ap}_{A_2}(\phi \uplus \psi')(n+1) = \\ = {} & \mathrm{ap}_{A_2}(\phi \uplus \psi')(n) \cup \bigcup_{?c \in \mathrm{ap}_{A_2}(\phi \uplus \psi')(n)} \{p \in !?\mathrm{N} \mid p \in \phi(n)(c) \vee p \in \psi'(n)(c)\} = \end{aligned}$$

Now we use that $\psi' \subseteq \mathrm{ap}^{!N}_{A_1}(\phi \uplus \psi')(\psi) = \mathrm{ap}_{A_1}(\phi \uplus \psi')(n) \copyright \ldots$ which means that the ports which stem from ψ' do not add anything new.

$$\begin{aligned} = {} & \mathrm{ap}_{A_2}(\phi \uplus \psi')(n) \cup \bigcup_{?c \in \mathrm{ap}_{A_2}(\phi \uplus \psi')(n)} \{p \in !?\mathrm{N} \mid p \in \phi(n)(c)\} = \\ \stackrel{\text{(I.A.)}}{=} {} & \mathrm{ap}_{A_2}(\phi)(n) \cup \bigcup_{?c \in \mathrm{ap}_{A_2}(\phi)(n)} \{p \in !?\mathrm{N} \mid p \in \phi(n)(c)\} = \\ = {} & \mathrm{ap}_{A_2}(\phi)(n+1) \end{aligned}$$

□

As a consequence of Lemma 3.8 the output filter can be determined without reference to the feedback stream.

Lemma 3.9

$\phi, \psi, \psi' \in \mathrm{NMS};\ \psi' \subseteq \psi;\ A_1 \subseteq A_2 \subseteq \mathrm{N}:$

$$\psi = \mathrm{ap}^{!N}_{A_1}(\phi \uplus \psi')(\psi) \Rightarrow \mathrm{ap}^{!N}_{A_2}(\phi \uplus \psi')(\psi) = ap^{!N}_{A_2}(\phi)(\psi)$$

Proof:

$n \in \mathbb{N}, c \in \mathrm{N}:$

$$\begin{aligned} & \mathrm{ap}^{!N}_{A_2}(\phi \uplus \psi')(\psi)(n)(c) = \\ \stackrel{\text{(L.3.8)}}{=} {} & \begin{cases} \mathrm{ap}_{A_2}(\phi)(n) \copyright \psi(n)(c) & \text{if } !c \in \mathrm{ap}_{A_2}(\phi)(n) \\ \epsilon & \text{else} \end{cases} = \\ = {} & ap^{!N}_{A_2}(\phi)(\psi)(n)(c) \end{aligned}$$

For determining the active ports of a feedbacked component, filtering the input beforehands less restrictively is superfluous.

Lemma 3.10

$\phi, \psi, \psi' \in \mathrm{NMS}, \psi' \subseteq \psi;\ A_1, A_2 \subseteq A_3 \subseteq \mathrm{N}:$

$$\psi = \mathrm{ap}^{!N}_{A_2}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(\psi) \Rightarrow \mathrm{ap}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi') = \mathrm{ap}_{A_1}(\phi \uplus \psi')$$

Proof:

Inductive proof.

n=0

$\mathrm{ap}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(0) \ = \ A_1 \ = \ \mathrm{ap}_{A_1}(\phi \uplus \psi')(0)$

n+1:

$$\mathrm{ap}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(n+1) \ = \ \mathrm{ap}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(n) \cup \bigcup_{?c \in \mathrm{ap}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(n)} \{p \in !?\mathrm{N} \mid (p \in \phi(n)(c) \wedge ?c \in \mathrm{ap}_{A_3}(\phi)(n)) \vee p \in \psi'(n)(c)\}$$

Let us have a closer look on $\mathrm{ap}_{A_3}(\phi)(n)$:

$$\begin{array}{lll} & \mathrm{ap}_{A_3}(\phi)(n) & \stackrel{(\mathrm{T.2.10})}{=} \ \mathrm{ap}_{A_3}(\mathrm{ap}^{?}_{A_3}(\phi))(n) \quad = \\ \stackrel{(\mathrm{L.3.8})}{=} & \mathrm{ap}_{A_3}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(n) & \supseteq \ \mathrm{ap}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(n) \end{array}$$

This means the condition $?c \in \mathrm{ap}_{A_3}(\phi)(n)$ is fulfilled for any set of the union, and can thus be dropped.

$$\mathrm{ap}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(n) \cup \bigcup_{?c \in \mathrm{ap}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(n)} \{p \in !?\mathrm{N} \mid p \in \phi(n)(c) \vee p \in \psi'(n)(c)\} =$$

$$\stackrel{(\mathrm{I.A.})}{=} \mathrm{ap}_{A_1}(\phi \uplus \psi')(n) \cup \bigcup_{?c \in \mathrm{ap}_{A_1}(\phi \uplus \psi')(n)} \{p \in !?\mathrm{N} \mid p \in \phi \uplus \psi'\}$$

□

As a consequence, filtering the input of a feedbacked component less restrictively has no effect on its output filtering.

Lemma 3.11

$\phi, \psi, \psi', \omega \in \mathrm{NMS}, \psi' \subseteq \psi;\ A_1, A_2 \subseteq A_3 \subseteq \mathrm{N}:$

$$\psi = \mathrm{ap}^{!N}_{A_2}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(\psi) \Rightarrow \mathrm{ap}^{!N}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(\omega) = ap^{!N}_{A_1}(\phi \uplus \psi')(\omega)$$

Proof:

Let $n \in \mathbb{N}, c \in \mathrm{N}$:

$$\begin{aligned}
& \mathrm{ap}^{!N}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(\omega)(n)(c) & = \\
= & \begin{cases} \mathrm{ap}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(n)©\omega(n)(c) & \text{if } !c \in \mathrm{ap}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(n) \\ \epsilon & \text{else} \end{cases} & = \\
\stackrel{(\text{L.3.10})}{=} & \begin{cases} \mathrm{ap}_{A_1}(\phi \uplus \psi')(n)©\omega(n)(c) & \text{if } !c \in \mathrm{ap}_{A_1}(\phi \uplus \psi')(n) \\ \epsilon & \text{else} \end{cases} & = \\
= & \ ap^{!N}_{A_1}(\phi \uplus \psi')(\omega)(n)(c)
\end{aligned}$$

The analogue of the last lemma for input filtering:

Lemma 3.12

$\psi \in \mathrm{NMS}, A_1, A_2 \subseteq A_3 \subseteq \mathrm{N}$:

$$\psi = \mathrm{ap}^{!N}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(\psi) \Rightarrow \mathrm{ap}^{?}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi') = ap^{?}_{A_1}(\phi \uplus \psi')$$

Proof:

Let $n \in \mathbb{N}, c \in \mathrm{N}$:

$$\begin{aligned}
& \mathrm{ap}^{?}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(n)(c) & = \\
\stackrel{(\text{PWD}(m))}{=} & \begin{cases} (\phi \uplus \psi')(n)(c) & \text{if } ?c \in \mathrm{ap}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(n) \cap \mathrm{ap}_{A_3}(\phi)(n) \\ \psi'(n)(c) & \text{if } ?c \in \mathrm{ap}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(n) \wedge ?c \notin \mathrm{ap}_{A_3}(\phi)(n) \\ \epsilon & \text{else} \end{cases} & = \\
\stackrel{(\text{L3.10,3.8})}{=} & \begin{cases} (\phi \uplus \psi')(n)(c) & \text{if } ?c \in \mathrm{ap}_{A_1}(\phi \uplus \psi')(n) \\ \epsilon & \text{else} \end{cases} & = \\
= & \ ap^{?}_{A_1}(\phi \uplus \psi')
\end{aligned}$$

We used $\mathrm{ap}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(n) \subseteq \mathrm{ap}_{A_3}(\phi)$ which can be shown:

$$\begin{aligned}
& \psi = \mathrm{ap}^{!N}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi')(\psi) & \Rightarrow \\
\stackrel{(\text{L.3.11})}{\Rightarrow} & \psi = \mathrm{ap}^{!N}_{A_1}(\phi \uplus \psi')(\psi) & \Rightarrow \\
\stackrel{(\text{L.3.10,3.8})}{\Rightarrow} & \left\{ \begin{array}{rcl} \mathrm{ap}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi') & = & \mathrm{ap}_{A_1}(\phi \uplus \psi') \\ \mathrm{ap}_{A_1}(\phi \uplus \psi') & = & ap_{A_1}(\phi) \end{array} \right\} & \Rightarrow \\
\stackrel{(\text{T.2.5})}{\Rightarrow} & \mathrm{ap}_{A_1}(\mathrm{ap}^{?}_{A_3}(\phi) \uplus \psi') \subseteq \mathrm{ap}_{A_3}(\phi)
\end{aligned}$$

□

Theorem 3.13 (Asynchronous Feedbacks are nonempty)

$f' \in \mathrm{GCB}_A, f \in \mathrm{B}, m \in \mathrm{Merges}$:

$$\forall \phi \in \mathrm{NMS} : f(\phi) = \psi \ \textit{where} \ \psi = f'(\phi \uplus \psi) \Rightarrow f \in \mathrm{GCB}_A$$

Proof:

According to Bannach's fixed point theorem (T.B.4), f is welldefined as a parameterized fixed point: $f := \mu h$ where $h(\psi, \phi) = f'(\phi \uplus \psi)$.

The resulting function is also contractive, since h is contractive (see T.B.5).

For genericity we have to prove that $f(\mathrm{ap}^?_A(\phi)) = f(\phi)$ and that $\mathrm{ap}^{!N}_A(\phi)(f(\phi)) = f(\phi)$:

$f(\mathrm{ap}^?_A(\phi)) = \psi$ with $\psi = f'(\mathrm{ap}^?_A(\phi) \uplus \psi)$

$$\stackrel{(\mathrm{generic}_A(f'))}{\Longrightarrow} \left\{ \begin{array}{l} \psi = \mathrm{ap}^{!N}_A(\mathrm{ap}^?_A(\phi) \uplus \psi)(\psi) \\ \psi = f'(\mathrm{ap}^?_A(\mathrm{ap}^?_A(\phi) \uplus \psi)) \end{array} \right\} \stackrel{(\mathrm{L.3.12})}{\Longrightarrow}$$

$$\psi = f'(\mathrm{ap}^?_A(\phi \uplus \psi)) \stackrel{(\mathrm{generic}_A(f'))}{=} f'(\phi \uplus \psi)$$

We showed that the fixed point of $\psi = f'(\mathrm{ap}^?_A(\phi) \uplus \psi)$ is the same as of $\psi = f'(\phi \uplus \psi)$. This implies that $f(\mathrm{ap}^?_A(\phi)) = f(\phi)$.

$$\begin{array}{rcl} \mathrm{ap}^{!N}_A(\phi)(f(\phi)) & = & ap^{!N}_A(\phi)(\psi) \text{ where } \psi = f'(\phi \uplus \psi) \\ & \stackrel{(\mathrm{generic}_A(f'))}{\Rightarrow} & \psi = \mathrm{ap}^{!N}_A(\phi \uplus \psi)(\psi) \\ & \stackrel{(\mathrm{L.3.9})}{\Rightarrow} & \mathrm{ap}^{!N}_A(\phi \uplus \psi)(\psi) = \mathrm{ap}^{!N}_A(\phi)(\psi) \end{array}$$

Which proves that $\mathrm{ap}^{!N}_A(\phi)(f(\phi)) = \mathrm{ap}^{!N}_A(\phi \uplus \psi)(\psi) = f(\phi)$.

3.6.3 Genericity and Full Abstractness

Here and for the remaining operators, the genericity, contractivity, and the full abstractness of the result are clearly guaranteed by the form of the definition. The elements of the result are functions out of the respective GCB set, and they are pointwise defined, which ensures abstractness. We will show this for the general case: a definition by a predicate P:

$\mathrm{FAC}_A(\{f' \in \mathrm{GCB}_A | \forall \phi' \in \mathrm{NMS} : P(\phi', f'(\phi'))\})$

$= \{f \in \mathrm{GCPB}_A | \forall \phi \in \mathrm{PNMS} : \exists f' \in \mathrm{GCB}_A : \forall \phi' : P(\phi', f(\phi')) \wedge f(\phi) = f'(\phi)\}$

$\subseteq \{f \in \mathrm{GCPB}_A | \forall \phi \in \mathrm{PNMS} : P(\phi, f(\phi)) \wedge \phi \in \mathrm{NMS}\}$

'$\supseteq$' is clear from extensivity of the FAC closure (see T. 2.20).

In cases like this, we will omit the proof of genericity and fully abstractnes due to its triviality.

3.6.4 Renaming Congruence

Assume $f \in \tau(\mu(C_{I,P}))$ which is equivalent to $\tau^{-1}(f) \in \mu(C_{I,P})$. This means that for each $\phi \in \mathrm{NMS}$ there exist $f'_\phi \in C$ and so that $\tau^{-1}(f(\tau(\phi))) = \psi$ with $\psi = f'_\phi(\phi \uplus \psi)$.

We will now show that f is also element of $\mu(\tau(C_{I,P}))$.

Let $g \in \mu(\tau(C_{I,P}))$ be defined by $\forall \phi \in \mathrm{NMS} : g(\phi) = \psi$, $\psi = \tau^{-1}(f'_\phi(\tau(\phi \uplus \psi)))$, this equals, due to the congruence of renaming with respect to the multi union:

$\tau^{-1}(f'_\phi(\tau(\phi) \uplus \tau(\psi)))$. Now we substitute $\tau^{-1}(\psi')$ for ψ and $\tau^{-1}(\phi')$ for ϕ: $\forall \phi' \in \text{NMS} : \tau(g(\tau^{-1}(\phi'))) = \psi', \psi' = f'_\phi(\phi' \uplus \psi')$.

This proves that each function of $\tau(\mu(C_{I,P}))$ has an eqvivalent in $\mu(\tau(C_{I,P}))$.

Now assume $f \in \mu(\tau(C_{I,P}))$, which means there exist $f'_\phi \in C$ for every $\phi \in$ NMS, so that:

$\forall \phi \in \text{NMS} : f(\phi) = \psi, \psi = \tau(f'_\phi(\tau^{-1}(\phi \uplus \psi)))$

By substituting ψ' for $\tau^{-1}(\psi)$ and ϕ' for $\tau^{-1}(\phi)$ like above we get: $\tau^{-1}(f(\tau(\phi'))) = \psi'$ with $\psi' = f'_\phi(\phi' \uplus \psi')$

This means $\tau^{-1}(f) \in \mu(C_{I,P})$ or more explicitly: $f \in \tau(\mu(C_{I,P}))$.

With this we have proofed that $\tau(\mu(C_{I,P})) = \mu(\tau(C_{I,P}))$: The asynchronous feedback is renaming congruent.

3.6.5 Name Abstractness

Since $C_{I,P} \in \text{Comp}$ it is name abstract: $\text{NAC}_{I\cup?!P}(C) = C$

Assume $\tau \in \text{N} \xrightarrow{inj.} \text{N}$ and $\tau|_{I\cup?!P} = id|_{I\cup?!P}$:

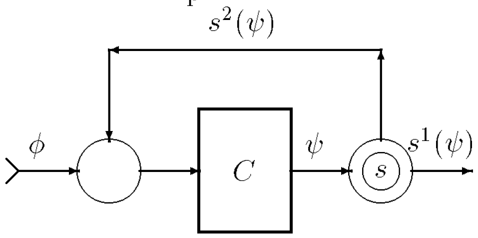
$\tau(\mu(C_{I,P})) \overset{(\mu\text{ is renaming congruent})}{=} \mu(\tau(C_{I,P})) \overset{(C\text{ is name abstract})}{=} \mu(C_{I,P})$

3.6.6 Congruence

Assume $C_{I,P} \equiv C'_{I,P'}$ via τ: $C_{I,P} = \tau(C'_{I,P'})$. Then also $\mu(C_{I,P}) \equiv \mu(C'_{I,P'})$ via τ: $\tau(\mu(C'_{I,P'})) = \mu(\tau(C'_{I,P'})) = \mu(C_{I,P})$.

3.7 Synchronous Feedback

The only, but important difference to the asynchronous feedback lays in the used split function. Every message which the feedbacked component outputs will either be output of the result or be input of the component. This choice will be made by the used split function, which determines the distribution for any output stream. But we will allow different split functions for every output stream which has to be splitted. By this, we will not be dependent on the filter congruence or the pointwise definedness of the split. These properties of splits only matter in our proofs of nonemptiness.



3.7.1 Definition

Definition 3.8 (Synchronous Feedback)
Comp :

$$\bar{\mu}(C_{I,P}) := \{f \in \mathrm{GCB}_{I\cup?!P} \mid \forall\phi : \exists g \in C, s \in \mathrm{Splits} : \\ f(\phi) = s^2(\psi), \psi = f'(\phi \uplus s^1(\psi))\}_{I,P}$$

3.7.2 Nonemptiness

Because we formulated the lemmas in the last section quite abstractly, they can also be used to prove the synchronous case.

Theorem 3.14 (Synchronous Feedbacks are nonempty)
$f' \in \mathrm{GCB}_A, f \in \mathrm{B}, s \in \mathrm{Splits}$:

$$\forall\phi \in \mathrm{NMS} : f(\phi) = s^2(\psi) \ \textit{where} \ \psi = f'(\phi \uplus s^1(\psi)) \Rightarrow f \in \mathrm{GCB}_A$$

Proof:
Compare proof of theorem 3.13.

Input Genericity

$$f(\mathrm{ap}^?_A(\phi)) = s^2(\psi) \text{ with } \psi = f'(\mathrm{ap}^?_A(\phi) \uplus s^1(\psi))$$

$$\overset{(\mathrm{generic}_A(f'))}{\Longrightarrow} \left\{ \begin{array}{l} \psi = \mathrm{ap}^{!N}_A(\mathrm{ap}^?_A(\phi) \uplus s^1(\psi))(\psi) \\ \psi = f'(\mathrm{ap}^?_A(\mathrm{ap}^?_A(\phi) \uplus s^1(\psi))) \end{array} \right\} \overset{(\mathrm{L.3.12})}{\Longrightarrow}$$

$$\psi = f'(\mathrm{ap}^?_A(\phi \uplus s^1(\psi))) \overset{(\mathrm{generic}_A(f'))}{=} f'(\phi \uplus s^1(\psi))$$

Output Genericity

$$\begin{array}{rcl} \mathrm{ap}^{!N}_A(\phi)(f(\phi)) & = & \mathrm{ap}^{!N}_A(\phi)(s^2(f'(\phi \uplus s^1(\psi)))) = \\ & \overset{(\mathrm{T.3.7})}{=} & s^2(\mathrm{ap}^{!N}_A(\phi)(f'(\phi \uplus s^1(\psi)))) = \\ & \overset{(\mathrm{L.3.9})}{=} & s^2(\mathrm{ap}^{!N}_A(\phi \uplus s^1(\psi))(f'(\phi \uplus s^1(\psi)))) = \\ & \overset{(\mathrm{generic}_A(f'))}{=} & s^2(f'(\phi \uplus s^1(\psi))) = f(\phi) \end{array}$$

□

3.7.3 Renaming Congruence

Let $f \in \tau(\bar{\mu}(C_{I,P}))$ be defined for each $\phi \in \mathrm{NMS}$ by $s_\phi \in \mathrm{Splits}, f' \in C$:

$$f(\phi) = \tau(s^2_\phi(\psi)), \ \psi = f_\phi(\tau^{-1}(\phi) \uplus s^1_\phi(\psi))$$

Let s'_ϕ be the splits for which: $\tau(s^i_\phi(\psi)) = s^i_\phi(\tau(\psi)), (i \in \{1, 2\})$:

$$f(\phi) = {s'}^2_\phi(\tau(\psi)), \ \psi = f_\phi(\tau^{-1}(\phi \uplus {s'}^1_\phi(\tau(\psi))))$$

By substituting ψ' for $\tau(\psi)$, we get:

$f(\phi) = s'^2_\phi(\psi')$, $\psi' = \tau(f_\phi(\tau^{-1}(\phi \uplus s'^1_\phi(\psi'))))$

That means f is element of $\bar{\mu}(\tau(C_{I,P}))$.

$\bar{\mu}(\tau(C_{I,P})) \subseteq \tau(\bar{\mu}(C_{I,P}))$ can be prove the reverse way. Compare proof of renaming congruence for the asynchronous feedback.

3.7.4 Name Abstractness

$$\tau(\bar{\mu}(C_{I,P})) \overset{(\bar{\mu} \text{ is renaming congruent})}{=} \bar{\mu}(\tau(C_{I,P})) \overset{(C \text{ is name abstract})}{=} \bar{\mu}(C_{I,P})$$

3.7.5 Congruence

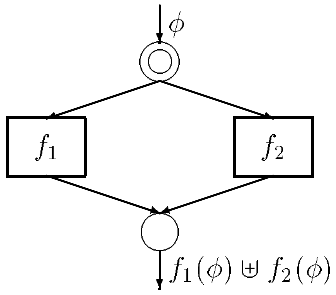
$$C_{I,P} = \tau(C'_{I,P'}) \Rightarrow \tau(\bar{\mu}(C'_{I,P'})) = \bar{\mu}(\tau(C'_{I,P'})) = \bar{\mu}(C_{I,P})$$

For further operators we will omit the proofs of name abstractness and congruence as long as they follow clearly from renaming congruence.

3.8 Asynchronous Parallel Composition

We use this composition, our easiest, for two components which receive a shared input and whose output is merged for the environment. But they do not interact in any way, that is why this combinator is fairly uncomplicated. Together with feedback it can be seen as a prestudy to the mutual feedback composition.

In the asynchronous case, the component reacts with a merging of the results of the operands' behaviors applied to the unchanged input.



3.8.1 Definition

Definition 3.9 (Asynchronous Parallel Composition)

$C_{1I_1,P_1}, C_{2I_2,P_2} \in \text{Comp}$; $P_1 \cap P_2 = \emptyset, I_1 \cap ?!P_2 = I_2 \cap ?!P_1 = \emptyset$:

$$\begin{aligned} C_{1I_1,P_1} \| C_{2I_2,P_2} := \{ & f \in GCB_{I_1 \cup I_2 \cup !?P_1 \cup !?P_2} | \\ & \forall \phi \in \text{NMS} : \exists f_1 \in C_1, f_2 \in C_2, m \in \text{Merges} : \\ & f(\phi) = f_1(\phi) \uplus f_2(\phi) \}_{I_1 \cup I_2, P_1 \cup P_2} \end{aligned}$$

3.8.2 Nonemptiness

Theorem 3.15 (Asynchronous Parallel Compositions are nonempty)

$f_1 \in \mathrm{GCB}_{A_1}, f_2 \in \mathrm{GCB}_{A_2}, f \in \mathrm{B}, m \in \mathrm{Merges}:$

$\forall \phi \in \mathrm{NMS}:\ f(\phi) = f_1(\phi) \uplus f_2(\phi) \Rightarrow f \in \mathrm{GCB}_{A_1 \cup A_2}$

Proof:

Input Genericity:

$$\begin{array}{rll}
 & f(\mathrm{ap}^?_{A_1 \cup A_2}(\phi)) & = \\
= & f_1(\mathrm{ap}^?_{A_1 \cup A_2}(\phi)) \uplus f_2(\mathrm{ap}^?_{A_1 \cup A_2}(\phi)) & = \\
\stackrel{(\mathrm{generic}_{A_i}(f_i))}{=} & f_1(\mathrm{ap}^?_{A_1}(\mathrm{ap}^?_{A_1 \cup A_2}(\phi))) \uplus f_2(\mathrm{ap}^?_{A_1}(\mathrm{ap}^?_{A_1 \cup A_2}(\phi))) & = \\
\stackrel{(\mathrm{T.2.11})}{=} & f_1(\mathrm{ap}^?_{A_1}(\phi)) \uplus f_2(\mathrm{ap}^?_{A_1}(\phi)) & = \\
\stackrel{(\mathrm{generic}_{A_i}(f_i))}{=} & f_1(\phi) \uplus f_2(\phi) \ = \ f(\phi) &
\end{array}$$

Output Genericity:

$$\begin{array}{rll}
 & \mathrm{ap}^{!N}_{A_1 \cup A_2}(\phi)(f(\phi)) & = \\
= & \mathrm{ap}^{!N}_{A_1 \cup A_2}(\phi)(f_1(\phi) \uplus f_2(\phi)) & = \\
\stackrel{(\mathrm{generic}_{A_i}(f_i))}{=} & \mathrm{ap}^{!N}_{A_1 \cup A_2}(\phi)(\mathrm{ap}^{!N}_{A_1}(\phi)(f_1(\phi)) \uplus \mathrm{ap}^{!N}_{A_2}(\phi)(f_2(\phi))) & = \\
\stackrel{(\mathrm{T.3.6})}{=} & \mathrm{ap}^{!N}_{A_1 \cup A_2}(\phi)(\mathrm{ap}^{!N}_{A_1}(\phi)(f_1(\phi))) \uplus \mathrm{ap}^{!N}_{A_2 \cup A_2}(\phi)(\mathrm{ap}^{!N}_{A_2}(\phi)(f_2(\phi))) & = \\
\stackrel{(\mathrm{T.2.13})}{=} & \mathrm{ap}^{!N}_{A_1}(\phi)(f_1(\phi)) \uplus \mathrm{ap}^{!N}_{A_2}(\phi)(f_2(\phi)) & = \\
\stackrel{(\mathrm{generic}_{A_i}(f_i))}{=} & f_1(\phi) \uplus f_2(\phi) \ = \ f(\phi) &
\end{array}$$

□

3.8.3 Renaming Congruence

Assume $f \in \tau(C_{1 I_1,P_1} \| C_{2 I_2,P_2})$:

$$\begin{array}{rl}
\forall \phi \in \mathrm{NMS}: f(\phi) & = \tau(f_{\phi_1}(\tau^{-1}(\phi)) \uplus f_{\phi_2}(\tau^{-1}(\phi))) \quad = \\
 & = \tau(f_{\phi_1}(\tau^{-1}(\phi))) \uplus \tau(f_{\phi_2}(\tau^{-1}(\phi))) = \\
 & = \tau(f_{\phi_1})(\phi) \uplus \tau(f_{\phi_2})(\phi)
\end{array}$$

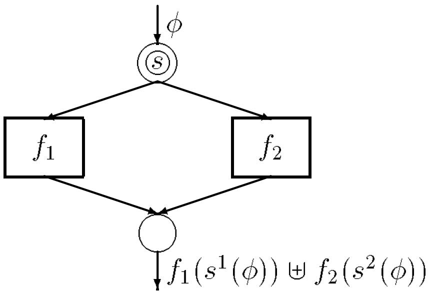
That means $f \in \tau(C_{1 I_1,P_1}) \| \tau(C_{2 I_2,P_2})$.

By this $\tau(C_{1 I_1,P_1} \| C_{2 I_2,P_2}) = \tau(C_{1 I_1,P_1}) \| \tau(C_{2 I_2,P_2})$ is proven.

Name abstractness and congruence follow straightforwardly from this.

3.9 Synchronous Parallel Composition

In the synchronous case we use a fair split to distribute the incoming messages between the two components.



3.9.1 Definition

Definition 3.10 (Synchronous Parallel Composition)

$C_{1\,I_1,P_1}, C_{2\,I_2,P_2} \in \mathrm{Comp};\ P_1 \cap P_2 = \emptyset, I_1 \cap ?!P_2 = I_2 \cap ?!P_1 = \emptyset$:

$$\begin{aligned} C_{1\,I_1,P_1} \bar{\|} C_{2\,I_2,P_2} := \{ & f \in GCB_{I_1 \cup I_2 \cup !?P_1 \cup !?P_2} | \\ & \forall \phi \in \mathrm{NMS} : \exists f_1 \in C_1, f_2 \in C_2, m \in \mathrm{Merges}, s \in \mathrm{Splits} : \\ & f(\phi) = f_1(s^1(\phi)) \uplus f_2(s^2(\phi)) \}_{I_1 \cup I_2, P_1 \cup P_2} \end{aligned}$$

3.9.2 Nonemptiness

We start by proving that filtering the input of a split is irrelevant to the set of active ports after the split.

Lemma 3.16

$A_2 \subseteq A_1 \subseteq \mathrm{N}, i \in \{1, 2\}$:

$$\mathrm{ap}_{A_2}(s^i(\mathrm{ap}^?_{A_1}(\phi))) = \mathrm{ap}_{A_2}(s^i(\phi))$$

Proof:
Inductive proof on n:

n=0:

$\mathrm{ap}_{A_2}(s^i(\mathrm{ap}^?_{A_1}(\phi)))(0) = A_2 = \mathrm{ap}_{A_2}(s^i(\phi))(0)$

n+1:

$$\begin{aligned} & \mathrm{ap}_{A_2}(s^i(\mathrm{ap}^?_{A_1}(\phi)))(n+1) && = \\ \overset{\text{(I.A.)}}{=} & \mathrm{ap}_{A_2}(s^i(\phi))(n) \cup \bigcup_{?c \in \mathrm{ap}_{A_2}(s^i(\phi))(n)} \{p \in ?!\mathrm{N} | p \in s^i(\mathrm{ap}^?_{A_1}(\phi))(n)(c)\} && = \\ \overset{(\mathrm{PWD}(s))}{=} & \mathrm{ap}_{A_2}(s^i(\phi))(n) \cup \\ & \bigcup_{?c \in \mathrm{ap}_{A_2}(s^i(\phi))(n)} \{p \in ?!\mathrm{N} | p \in s^i(\phi)(n)(c) \wedge ?c \in \mathrm{ap}_{A_1}(s^i(\phi))(n)\} && = \\ \overset{(A_2 \subseteq A_1)}{=} & \mathrm{ap}_{A_2}(s^i(\phi))(n+1) \end{aligned}$$

$\square$

As a consequence of the last lemma, input filters after the split dominate less restrictive input filters before the split.

Lemma 3.17

$A_2 \subseteq A_1 \subseteq \mathrm{N}, i \in \{1,2\}:$

$$\mathrm{ap}^?_{A_2}(s^i(\mathrm{ap}^?_{A_1}(\phi))) = \mathrm{ap}^?_{A_2}(s^i(\phi))$$

Proof:

$n \in \mathbb{N}, c \in \mathrm{N}:$

$$\begin{array}{cll}
 & \mathrm{ap}^?_{A_2}(s^i(\mathrm{ap}^?_{A_1}(\phi)))(n)(c) & = \\
\overset{(\text{I.A.})}{=} & \begin{cases} s^i(\mathrm{ap}^?_{A_1}(\phi))(n)(c) \text{ if } ?c \in \mathrm{ap}_{A_2}(s^i(\mathrm{ap}^?_{A_1}(\phi)))(n) \\ \epsilon \quad \text{else} \end{cases} & \\
\overset{(\text{L.3.16})}{=} & \begin{cases} s^i(\phi)(n)(c) \text{ if } ?c \in \mathrm{ap}_{A_2}(s^i(\phi))(n) \cap \mathrm{ap}_{A_1}(\phi))(n) \\ \epsilon \quad \text{else} \end{cases} & \\
\overset{(A_2 \subseteq A_1,\, s^i(\phi) \subseteq \phi)}{=} & \mathrm{ap}^?_{A_2}(s^i(\phi))(n) &
\end{array}$$

□

Theorem 3.18 (Synchronous Parallel Compositions are nonempty)

$f_1 \in \mathrm{GCB}_{A_1}, f_2 \in \mathrm{GCB}_{A_2}, f \in \mathrm{B}, m \in \mathrm{Merges}:$

$$\forall \phi \in \mathrm{NMS}: \ f(\phi) = f_1(s^1(\phi)) \uplus f_2(s^2(\phi)) \ \Rightarrow f \in \mathrm{GCB}_{A_1 \cup A_2}$$

Proof:

Input Genericity:

$$\begin{array}{cll}
 & f(\mathrm{ap}^?_{A_1 \cup A_2}(\phi)) & = \\
= & f_1(s^1(\mathrm{ap}^?_{A_1 \cup A_2}(\phi))) \uplus f_2(s^2(\mathrm{ap}^?_{A_1 \cup A_2}(\phi))) & = \\
\overset{(\mathrm{generic}_{A_i}(f_i))}{=} & f_1(\mathrm{ap}^?_{A_1}(s^1(\mathrm{ap}^?_{A_1 \cup A_2}(\phi)))) \uplus f_2(\mathrm{ap}^?_{A_2}(s^2(\mathrm{ap}^?_{A_1 \cup A_2}(\phi)))) & = \\
\overset{(\text{L.3.17})}{=} & f_1(\mathrm{ap}^?_{A_1}(s^1(\phi))) \uplus f_2(\mathrm{ap}^?_{A_2}(s^2(\phi))) & = \\
\overset{(\mathrm{generic}_{A_i}(f_i))}{=} & f_1(s^1(\phi)) \uplus f_2(s^2(\phi)) \ = \ f(\phi) &
\end{array}$$

Output Genericity:

$$
\begin{array}{rll}
 & \mathrm{ap}^{!N}_{A_1\cup A_2}(\phi)(f(\phi)) & = \\
= & \mathrm{ap}^{!N}_{A_1\cup A_2}(\phi)(f_1(s^1(\phi)) \uplus f_2(s^2(\phi))) & = \\
\overset{(\mathrm{generic}_{A_i}(f_i))}{=} & \mathrm{ap}^{!N}_{A_1\cup A_2}(\phi)(\mathrm{ap}^{!N}_{A_1}(s^1(\phi))(f_1(s^1(\phi)))\uplus & \\
 & \quad \mathrm{ap}^{!N}_{A_2}(s^2(\phi))(f_2(s^2(\phi)))) & = \\
\overset{(\mathrm{T.3.6})}{=} & \mathrm{ap}^{!N}_{A_1\cup A_2}(\phi)(\mathrm{ap}^{!N}_{A_1}(s^1(\phi))(f_1(s^1(\phi))))\uplus & \\
 & \quad \mathrm{ap}^{!N}_{A_2\cup A_2}(\phi)(\mathrm{ap}^{!N}_{A_2}(s^2(\phi))(f_2(s^2(\phi)))) & = \\
\overset{(\mathrm{T.2.13})}{=} & \mathrm{ap}^{!N}_{A_1}(s^1(\phi))(f_1(s^1(\phi))) \uplus \mathrm{ap}^{!N}_{A_2}(s^2(\phi))(f_2(s^2(\phi))) & = \\
\overset{(\mathrm{generic}_{A_i}(f_i))}{=} & f_1(s^1(\phi)) \uplus f_2(s^2(\phi)) \ = \ f(\phi) &
\end{array}
$$

□

3.9.3 Renaming Congruence

Assume $f \in \tau(C_{1I_1,P_1} \bar{\|} C_{2I_2,P_2})$:

$$
\begin{aligned}
\forall \phi \in \mathrm{NMS} : f(\phi) &= \tau(f_{\phi 1}(s^1_\phi(\tau^{-1}(\phi)))) \uplus f_{\phi 2}(s^2_\phi(\tau^{-1}(\phi)))) &= \\
&= \tau(f_{\phi 1}(\tau^{-1}({s'_\phi}^1(\phi)))) \uplus \tau(f_{\phi 2}(\tau^{-1}({s'_\phi}^2(\phi)))) &= \\
&= \tau(f_{\phi 1})({s'_\phi}^1(\phi)) \uplus \tau(f_{\phi 2})({s'_\phi}^1(\phi))
\end{aligned}
$$

That means $f \in \tau(C_{1I_1,P_1}) \| \tau(C_{2I_2,P_2})$.

We took s'_ϕ as that split which fullfills $\tau(s^i_\phi(\alpha)) = {s'_\phi}^i(\tau(\alpha))$ for each $\alpha \in \mathrm{NMS}$ (for existence refer to T.3.5). And also, seen reversely, such a s_ϕ exists for every s'_ϕ.

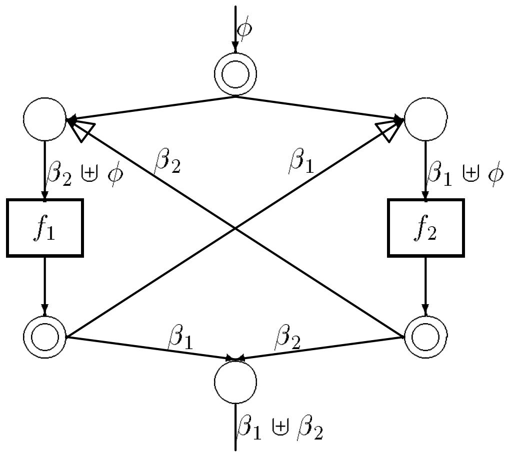
3.10 Asychronous Composition

We now come to the most important operator: composition with mutual feedback, also simply called: composition of components. It is important because it realizes a quite natural idea of network construction: If we have two components, then each communicates with the shared environment and they communicate with each other.

Seen from the stream processing view, there will be a stream from the outside, distributed to the two components, and the same reversely. So far this is parallel composition. But there will also be a stream from one component to the other and one back, which we call feedback streams.

The feedback problem can be solved as we did in the sections about the feedback operators, only that our equation is much more complicated now. But still the equation is, as a composition of nonexpansive and contractive functions, contractive and thereby uniquely solvable.

Although our notation will seem to propose two equations, the problem can be reduced to one by expanding the definition of one of the βs in that of the other.



3.10.1 Definition

Definition 3.11 (Asynchronous Composition)

$C_{1 I_1,P_1}, C_{2 I_2,P_2} \in \mathrm{Comp};\ P_1 \cap P_2 = \emptyset, I_1 \cap ?!P_2 = I_2 \cap ?!P_1 = \emptyset :$

$$\begin{aligned} C_{1 I_1,P_1} \otimes C_{2 I_2,P_2} := \{ & f \in GCB_{I_1 \cup I_2 \cup !?P_1 \cup !?P_2} | \\ & \forall \phi \in \mathrm{NMS} : \exists f_1 \in C_1, f_2 \in C_2 : \\ & f(\phi) = \beta_1 \uplus \beta_2 \ \textit{where}\ \beta_1 = f_1(\beta_2 \uplus \phi) \ \wedge \\ & \qquad\qquad\qquad\qquad \beta_2 = f_2(\beta_1 \uplus \phi) \ \}_{I_1 \cup I_2, P_1 \cup P_2} \end{aligned}$$

3.10.2 Nonemptiness

This proof needs a lot of preparatory work. Again, we often try to be very general in our lemmas, so that we can use them for the synchronous as th asynchronous case.

The first lemma shows that feedback channels hold no essential information determining new active channels. The output of a generic function cannot hold more access rights than a superstream of its input.

Lemma 3.19

$A_1, A_2, A_3 \subseteq \mathrm{N}; \phi_1, \phi_2, \phi_3, \beta_1, \beta_1', \beta_2, \beta_2' \in \mathrm{NMS} :$

$A_1, A_2 \subseteq A_3; \phi_1, \phi_2 \subseteq \phi_3; \beta_1' \subseteq \beta_1; \beta_2' \subseteq \beta_2 :$

$\beta_1 = \mathrm{ap}^{!N}_{A_1}(\beta_2' \uplus \phi_1)(\beta_1) \ \wedge \ \beta_2 = \mathrm{ap}^{!N}_{A_2}(\beta_1' \uplus \phi_2)(\beta_2) \ \Rightarrow$

$$\begin{aligned} &\mathrm{ap}_{A_3}(\beta_2' \uplus \phi_1) \subseteq \mathrm{ap}_{A_3}(\phi_3) \\ &\mathrm{ap}_{A_3}(\beta_1' \uplus \phi_2) \subseteq \mathrm{ap}_{A_3}(\phi_3) \end{aligned}$$

Proof:

Inductive Proof on n:

n=0:

$$\mathrm{ap}_{A_3}(\beta_2' \uplus \phi_1)(0) = A_3 = \mathrm{ap}_{A_3}(\phi_3)(0)$$
$$\mathrm{ap}_{A_3}(\beta_1' \uplus \phi_2)(0) = A_3 = \mathrm{ap}_{A_3}(\phi_3)(0)$$

n+1:

$$\begin{array}{rll}
 & \mathrm{ap}_{A_3}(\beta_2' \uplus \phi_1)(n+1) & \subseteq \\
\overset{(\text{I.A.})}{\subseteq} & \mathrm{ap}_{A_3}(\phi_3)(n) \cup \bigcup_{?c \in \mathrm{ap}_{A_3}(\phi_3)(n)} \{p \in ?!\mathrm{N} \mid p \in \phi_1(n)(c) \vee p \in \beta_2'(n)(c)\} & \subseteq \\
\subseteq & \mathrm{ap}_{A_3}(\phi_3)(n) \cup \bigcup_{?c \in \mathrm{ap}_{A_3}(\phi_3)(n)} \{p \in ?!\mathrm{N} \mid p \in \phi_1(n)(c) \wedge p \in \mathrm{ap}_{A_2}(\beta_1' \uplus \phi_2)(n) \copyright \ldots\} & \subseteq \\
\overset{(\text{I.A.})}{\subseteq} & \mathrm{ap}_{A_3}(\phi_3)(n) \cup \bigcup_{?c \in \mathrm{ap}_{A_3}(\phi_3)(n)} \{p \in ?!\mathrm{N} \mid p \in \phi_1(n)(c) \wedge p \in \mathrm{ap}_{A_3}(\phi_3)(n) \copyright \ldots\} & \subseteq \\
\subseteq & \mathrm{ap}_{A_3}(\phi_3)(n+1) &
\end{array}$$

The proof for $\mathrm{ap}_{A_3}(\beta_1' \uplus \phi_2)(n+1) \subseteq \mathrm{ap}_{A_3}(\phi_3)(n+1)$ is symmetric: Only the indexes have to be exchanged in the proof above.

□

A less restrictive input filter does not matter for our composition applied to components which are more restrictive themselves.

Lemma 3.20

$A_1, A_2, A_3 \subseteq \mathrm{N}; \phi, \beta_1, \beta_1', \beta_2, \beta_2' \in \mathrm{NMS}:$
$A_1, A_2 \subseteq A_3; \beta_1' \subseteq \beta_1; \beta_2' \subseteq \beta_2:$

$$\beta_1 = \mathrm{ap}^{!N}_{A_1}(\beta_2' \uplus \mathrm{ap}^{?}_{A_3}(\phi))(\beta_1) \wedge \beta_2 = \mathrm{ap}^{!N}_{A_2}(\beta_1' \uplus \mathrm{ap}^{?}_{A_3}(\phi))(\beta_2) \Rightarrow$$

$$\mathrm{ap}_{A_1}(\beta_2' \uplus \mathrm{ap}^{?}_{A_3}(\phi)) = \mathrm{ap}_{A_1}(\beta_2' \uplus \phi)$$
$$\mathrm{ap}_{A_2}(\beta_1' \uplus \mathrm{ap}^{?}_{A_3}(\phi)) = \mathrm{ap}_{A_2}(\beta_1' \uplus \phi)$$

Proof:

Inductive Proof on n:

n=0:

$$\mathrm{ap}_{A_1}(\beta_2' \uplus \mathrm{ap}^{?}_{A_3}(\phi))(0) = A_1 = \mathrm{ap}_{A_1}(\beta_2' \uplus \phi)(0)$$
$$\mathrm{ap}_{A_2}(\beta_1' \uplus \mathrm{ap}^{?}_{A_3}(\phi))(0) = A_2 = \mathrm{ap}_{A_2}(\beta_1' \uplus \phi)(0)$$

n+1:

$$\begin{array}{rl}
 & \mathrm{ap}_{A_1}(\beta_2' \uplus \mathrm{ap}^{?}_{A_3}(\phi))(n+1) \qquad = \\
= & \mathrm{ap}_{A_1}(\beta_2' \uplus \mathrm{ap}^{?}_{A_3}(\phi))(n) \cup \\
 & \bigcup_{?c \in \mathrm{ap}_{A_1}(\beta_2' \uplus \mathrm{ap}^{?}_{A_3}(\phi))(n)} \begin{array}{l}\{p \in ?!\mathrm{N} \mid p \in \beta_2')(n)(c) \vee \\ (p \in \phi(n)(c) \wedge ?c \in \mathrm{ap}_{A_3}(\phi)(n))\}\end{array}
\end{array}$$

We can drop the $?c \in \mathrm{ap}_{A_3}(\phi)(n)$ condition because of:

$$\mathrm{ap}_{A_1}(\beta_2' \uplus \mathrm{ap}^?_{A_3}(\phi)) \subseteq \mathrm{ap}_{A_3}(\beta_2' \uplus \mathrm{ap}^?_{A_3}(\phi)) \overset{(\mathrm{L.3.19})}{\subseteq} \mathrm{ap}_{A_3}(\mathrm{ap}^?_{A_3}(\phi)) \overset{(\mathrm{L.2.10})}{=} \mathrm{ap}_{A_3}(\phi)$$

With that and the inductive assumption, the above equals: $\mathrm{ap}_{A_1}(\beta_2' \uplus \phi)(n+1)$

The proof for $\mathrm{ap}_{A_2}(\beta_1' \uplus \mathrm{ap}^?_{A_3}(\phi))(n+1) = \mathrm{ap}_{A_2}(\beta_1' \uplus \phi)(n+1)$ is similar, just the indexes have to be exchanged.

□

The content of the feedback channels is independent of a less restrictive filter on the input.

Corollary 3.21

$A_1, A_2, A_3 \subseteq \mathrm{N}; \phi, \beta_1, \beta_1', \beta_2, \beta_2' :$

$A_1, A_2 \subseteq A_3; \beta_1' \subseteq \beta_1; \beta_2' \subseteq \beta_2 :$

$$\beta_1 = \mathrm{ap}^{!N}_{A_1}(\beta_2' \uplus \mathrm{ap}^?_{A_3}(\phi))(\beta_1) \wedge \beta_2 = \mathrm{ap}^{!N}_{A_2}(\beta_1' \uplus \mathrm{ap}^?_{A_3}(\phi))(\beta_2) \Rightarrow$$

$$\beta_1 = \mathrm{ap}^{!N}_{A_1}(\beta_2' \uplus \phi)(\beta_1) \wedge \quad \beta_2 = \mathrm{ap}^{!N}_{A_2}(\beta_1' \uplus \phi)(\beta_2)$$

Proof:

From lemma 3.20 we know that $\mathrm{ap}_{A_1}(\beta_2' \uplus \mathrm{ap}^?_{A_3}(\phi_1)) = \mathrm{ap}_{A_1}(\beta_2' \uplus \phi_1)$ which establishes the identical effect of the output filters on β_1. The same argument with different indexes works for β_2.

□

Finally we can see the sense of our preparations.

Theorem 3.22 (Asynchronous Compositions are nonempty)

$f_1 \in \mathrm{GCB}_{A_1}, f_2 \in \mathrm{GCB}_{A_2}, f \in \mathrm{B} :$

$$\begin{aligned} \forall \phi \in \mathrm{NMS} : f(\phi) = \beta_1 \uplus \beta_2 & \\ \textit{where } \beta_1 = f_1(\beta_2 \uplus \phi) \wedge & \\ \beta_2 = f_2(\beta_1 \uplus \phi) & \end{aligned}$$

$$\Rightarrow f \in \mathrm{GCB}_{A_1 \cup A_2}$$

Proof:

Input Genericity:

$$\begin{aligned} & f(\mathrm{ap}^?_{A_1 \cup A_2}(\phi)) && = \\ = \quad & f_1(\beta_2 \uplus \mathrm{ap}^?_{A_1 \cup A_2}(\phi)) \uplus (\beta_1 \uplus \mathrm{ap}^?_{A_1 \cup A_2}(\phi)) && = \\ \overset{(\mathrm{generic}_{A_1}(f_1))}{=} \quad & f_1(\mathrm{ap}^?_{A_1}(\beta_2 \uplus \mathrm{ap}^?_{A_1 \cup A_2}(\phi))) \uplus \ldots && \end{aligned}$$

We will only consider the left side of the outer multi union. The right side can be transformed the same, just with renamed indexes. Let $n \in \mathbb{N}, c \in \mathrm{N}$:

$$\begin{array}{rll}
& \mathrm{ap}^?_{A_1}(\beta_2 \uplus \mathrm{ap}^?_{A_1 \cup A_2}(\phi))(n)(c) & = \\
= & \begin{cases} (\beta_2 \uplus \phi)(n)(c) & \text{if } ?c \in \mathrm{ap}_{A_1}(\beta_2 \uplus \mathrm{ap}^?_{A_1\cup A_2}(\phi))(n) \cap \mathrm{ap}_{A_1 \cup A_2}(\phi)(n) \\ \beta_2(n)(c) & \text{if } ?c \in \mathrm{ap}_{A_1}(\beta_2 \uplus \mathrm{ap}^?_{A_1\cup A_2}(\phi))(n) \setminus \mathrm{ap}_{A_1 \cup A_2}(\phi)(n) \\ \epsilon & \text{else} \end{cases} & = \\
\overset{(\text{L.3.19,3.20})}{=} & \begin{cases} (\beta_2 \uplus \phi)(n)(c) & \text{if } ?c \in \mathrm{ap}_{A_1}(\beta_2 \uplus \phi)(n) \\ \epsilon & \text{else} \end{cases} & = \\
= & \mathrm{ap}^?_{A_1}(\beta_2 \uplus \phi) &
\end{array}$$

We can eliminate the remaining filter as easy as we introduced it, since a generic function is applied to it.

Lemmas 3.19 and 3.20 were used in the proof to show that: $\mathrm{ap}_{A_1}(\beta_2 \uplus \mathrm{ap}^?_{A_1 \cup A_2}(\phi))(n) =$
$= \mathrm{ap}_{A_1}(\beta_2 \uplus \phi)(n)$ and that this is a subset of $\mathrm{ap}_{A_1 \cup A_2}(\phi)(n)$.

Output Genericity:

Again we only consider one of the two symmetric sides of the outer union. The other one will simply be omited in our presentation of the proof.

$$\begin{array}{rll}
& \mathrm{ap}^{!N}_{A_1 \cup A_2}(\phi)(f(\phi)) & = \\
= & \mathrm{ap}^{!N}_{A_1 \cup A_2}(\phi)(\mathrm{ap}^{!N}_{A_1}(\beta_2 \uplus \phi)(\beta_1) \uplus \ldots) & = \\
\overset{(\text{T.3.6})}{=} & \mathrm{ap}^{!N}_{A_1 \cup A_2}(\phi)(\mathrm{ap}^{!N}_{A_1}(\beta_2 \uplus \phi)(\beta_1)) \uplus \ldots &
\end{array}$$

Let $n \in \mathbb{N}, c \in \mathrm{N}$:

$$\begin{array}{rll}
& \mathrm{ap}^{!N}_{A_1 \cup A_2}(\phi)(\mathrm{ap}^{!N}_{A_1}(\beta_2 \uplus \phi)(\beta_1)(n)(c) & = \\
= & \begin{cases} \mathrm{ap}_{A_1 \cup A_2}(\phi) \copyright \mathrm{ap}_{A_1}(\beta_2 \uplus \phi) \copyright \beta_1(n)(c) & \\ \qquad \text{if } !c \in \mathrm{ap}_{A_1 \cup A_2}(\phi) \cap \mathrm{ap}_{A_1}(\beta_2 \uplus \phi) & \\ \epsilon \ \text{else} & \end{cases} & = \\
\overset{(\text{L.3.19})}{=} & \begin{cases} \mathrm{ap}_{A_1}(\beta_2 \uplus \phi) \copyright \beta_1(n)(c) & \text{if } !c \in \mathrm{ap}_{A_1}(\beta_2 \uplus \phi) \\ \epsilon & \text{else} \end{cases} & = \\
= & \mathrm{ap}^{!N}_{A_1}(\beta_2 \uplus \phi)(\beta_1)(n)(c) &
\end{array}$$

□

3.10.3 Renaming Congruence

We will show that if and only if $\tau(f) \in C1_{I_1,P_1} \otimes C2_{I_2,P_2}$ then $f \in \tau(C1_{I_1,P_1}) \otimes \tau(C2_{I_2,P_2})$.

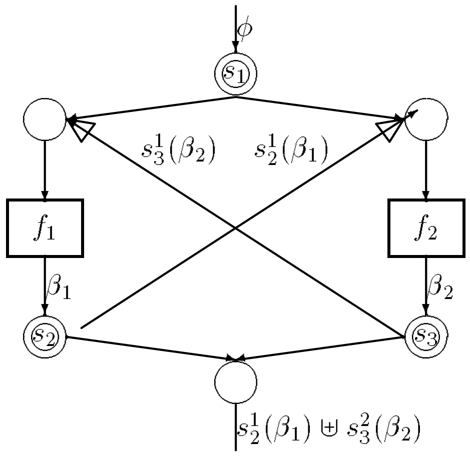
$\phi \in \mathrm{NMS}$:

$$\begin{aligned}
\tau(f(\tau^{-1}(\phi))) &= \tau(\beta_1 \uplus \beta_2) \\
&= \tau(\beta_1) \uplus \tau(\beta_2) ;
\end{aligned}$$

$$\tau(\beta_1) = \tau(f_1(\tau^{-1}(\phi) \uplus \beta_2)) = \tau(f_1(\tau^{-1}(\phi \uplus \tau(\beta_2))))$$
$$\tau(\beta_2) = \tau(f_2(\tau^{-1}(\phi) \uplus \beta_1)) = \tau(f_2(\tau^{-1}(\phi \uplus \tau(\beta_1))))$$

3.11 Synchronous Composition

There are three positions where synchronicity makes a difference: the distribution of inputs to the components and the two splits between feedback and output. Therefore we need to choose three different split functions, but again: we allow different splits for each input stream, that means pointwise definedness and filter congruence do not matter for the actually defined results. There sense lies in the nonemptiness proofs.



3.11.1 Definition

Definition 3.12 (Synchronous Composition)

$C1_{I_1,P_1}, C2_{I_2,P_2} \in \text{Comp};\ P_1 \cap P_2 = \emptyset, I_1 \cap ?!P_2 = I_2 \cap ?!P_1 = \emptyset$:

$$C1_{I_1,P_1} \bar{\otimes} C2_{I_2,P_2} :=$$
$$\{f \in GCB_{I_1 \cup I_2 \cup !?P_1 \cup !?P_2} |$$
$$\forall \phi \in \text{NMS} : \exists f_1 \in C_1, f_2 \in C_2, s_1, s_2, s_3 \in \text{Splits} :$$
$$f(\phi) = s_2^1(\beta_1) \uplus s_3^2(\beta_2) \ \textit{where}\ \beta_1 = f_1(s_3^1(\beta_2) \uplus s_1^1(\phi)) \wedge$$
$$\beta_2 = f_2(s_2^2(\beta_1) \uplus s_1^2(\phi))\ \}_{I_1 \cup I_2, P_1 \cup P_2}$$

3.11.2 Nonemptiness

Additionally to the lemmas established before the asynchronous case, we need a specific version of L.3.20. The input filter before the split is unnecessary.

Lemma 3.23

$A_1, A_2, A_3 \subseteq \text{N};\ \phi, \beta_1, \beta_1', \beta_2, \beta_2' \in \text{NMS};\ s \in \text{Splits}$:

$A_1, A_2 \subseteq A_3;\ \beta_1' \subseteq \beta_1;\ \beta_2' \subseteq \beta_2$:

$$\beta_1 = \mathrm{ap}^{!N}_{A_1}(\beta_2' \uplus s^1(\mathrm{ap}^?_{A_3}(\phi)))(\beta_1) \wedge \beta_2 = \mathrm{ap}^{!N}_{A_2}(\beta_1' \uplus s^2(\mathrm{ap}^?_{A_3}(\phi)))(\beta_2) \Rightarrow$$

$$\begin{aligned} \mathrm{ap}_{A_1}(\beta_2' \uplus s^1(\mathrm{ap}^?_{A_3}(\phi))) &= \mathrm{ap}_{A_1}(\beta_2' \uplus s^1(\phi)) \\ \mathrm{ap}_{A_2}(\beta_1' \uplus s^2(\mathrm{ap}^?_{A_3}(\phi))) &= \mathrm{ap}_{A_2}(\beta_1' \uplus s^2(\phi)) \end{aligned}$$

Proof:

Inductive Proof on n:

n=0:

$$\begin{aligned} \mathrm{ap}_{A_1}(\beta_2' \uplus s^1(\mathrm{ap}^?_{A_3}(\phi)))(0) &= A_1 = \mathrm{ap}_{A_1}(\beta_2' \uplus s^1(\phi))(0) \\ \mathrm{ap}_{A_2}(\beta_1' \uplus s^2(\mathrm{ap}^?_{A_3}(\phi)))(0) &= A_2 = \mathrm{ap}_{A_2}(\beta_1' \uplus s^2(\phi))(0) \end{aligned}$$

n+1:

$$\begin{aligned} &\mathrm{ap}_{A_1}(\beta_2' \uplus s^1(\mathrm{ap}^?_{A_3}(\phi)))(n+1) = \\ =\ &\mathrm{ap}_{A_1}(\beta_2' \uplus s^1(\mathrm{ap}^?_{A_3}(\phi)))(n) \cup \\ &\bigcup_{?c \in \mathrm{ap}_{A_1}(\beta_2' \uplus s^1(\mathrm{ap}^?_{A_3}(\phi)))(n)} \{p \in ?!\mathrm{N} \mid p \in \beta_2'(n)(c) \vee \\ &\quad (p \in s^1(\phi)(n)(c) \wedge ?c \in \mathrm{ap}_{A_3}(\phi)(n))\} \end{aligned}$$

We can drop the $?c \in \mathrm{ap}_{A_3}(\phi)(n)$ condition because of:

$$\mathrm{ap}_{A_1}(\beta_2' \uplus s^1(\mathrm{ap}^?_{A_3}(\phi)))(n) \overset{(\mathrm{L.3.19})}{\subseteq} \mathrm{ap}_{A_3}(\phi)(n)$$

With that and the inductive assumption, the above equals: $\mathrm{ap}_{A_1}(\beta_2' \uplus s^1(\phi))(n+1)$

The proof for $\mathrm{ap}_{A_2}(\beta_1' \uplus s^2(\mathrm{ap}^?_{A_3}(\phi))(n+1) = \mathrm{ap}_{A_2}(\beta_1' \uplus s^2(\phi))(n+1)$ is similar, just the indexes have to be exchanged.

$\square$

Theorem 3.24 (Synchronous Compositions are nonempty)

$f_1 \in \mathrm{GCB}_{A_1}, f_2 \in \mathrm{GCB}_{A_2}, f \in \mathrm{B}; s_1, s_2, s_3 \in \mathrm{Splits}:$

$$\begin{aligned} \forall \phi \in \mathrm{NMS}: f(\phi) = s_2^1(\beta_1) \uplus s_3^2(\beta_2) \\ \textit{where } \beta_1 = f_1(s_3^1(\beta_2) \uplus s_1^1(\phi)) \wedge \\ \beta_2 = f_2(s_2^2(\beta_1) \uplus s_1^2(\phi)) \end{aligned}$$

$$\Rightarrow f \in \mathrm{GCB}_{A_1 \cup A_2}$$

Proof:

Input Genericity:

$$\begin{aligned} & f(\mathrm{ap}^?_{A_1 \cup A_2}(\phi)) && = \\ = \quad & s_2^1(f_1(s_3^1(\beta_2) \uplus s_1^1(\mathrm{ap}^?_{A_1 \cup A_2}(\phi)))) \uplus \ldots && = \\ \overset{(\mathrm{generic}_{A_1}(f_1))}{=} \quad & s_2^1(f_1(\mathrm{ap}^?_{A_1}(s_3^1(\beta_2) \uplus s_1^1(\mathrm{ap}^?_{A_1 \cup A_2}(\phi))))) \uplus \ldots \end{aligned}$$

We will only consider the left side of the outer merge. The right side is the same with changed indexes. Let $n \in \mathbb{N}, c \in \mathrm{N}$:

$$
\begin{aligned}
& \mathrm{ap}^?_{A_1}(s^1_3(\beta_2) \uplus s^1_1(\mathrm{ap}^?_{A_1 \cup A_2}(\phi)))(n)(c) & = \\
= & \begin{cases} s^1_3(\beta_2) \uplus s^1_1(\phi)(n)(c) \\ \quad \text{if } ?c \in \mathrm{ap}_{A_1}(s^1_3(\beta_2) \uplus s^1_1(\mathrm{ap}^?_{A_1\cup A_2}(\phi)))(n) \cap \mathrm{ap}_{A_1 \cup A_2}(\phi)(n) \\ s^1_3(\beta_2)(n)(c) \\ \quad \text{if } ?c \in \mathrm{ap}_{A_1}(s^1_3(\beta_2) \uplus s^1_1(\mathrm{ap}^?_{A_1\cup A_2}(\phi))))(n) \setminus \mathrm{ap}_{A_1 \cup A_2}(\phi)(n) \\ \epsilon \ \text{else} \end{cases} & = \\
\overset{(\text{L.3.19,3.23})}{=} & \begin{cases} (s^1_3(\beta_2) \uplus s^1_1(\phi))(n)(c) \text{ if } ?c \in \mathrm{ap}_{A_1}(s^1_3(\beta_2) \uplus s^1_1(\phi))(n) \\ \epsilon \qquad \text{else} \end{cases} & = \\
= & \ \mathrm{ap}^?_{A_1}(s^1_3(\beta_2) \uplus s^1_1(\phi))
\end{aligned}
$$

We can eliminate the remaining filter as easy as we introduced it, since a generic function is applied to it.

Lemmas 3.19 and 3.23 were used in the proof to show that: $\mathrm{ap}_{A_1}(s^1_3(\beta_2) \uplus s^1_1(\mathrm{ap}^?_{A_1\cup A_2}(\phi))(n) = \mathrm{ap}_{A_1}(s^1_3(\beta_2) \uplus s^1_1(\phi))(n)$ and that this is a subset of $\mathrm{ap}_{A_1 \cup A_2}(\phi)(n)$.

Output Genericity:

Again we only consider one of the two symmetric sides of the outer merge. The other one will simply be omited in our presentation of the proof.

$$
\begin{aligned}
& \mathrm{ap}^{!N}_{A_1\cup A_2}(\phi)(f(\phi)) & = \\
= & \ \mathrm{ap}^{!N}_{A_1\cup A_2}(\phi)(s^1_2(\mathrm{ap}^{!N}_{A_1}(s^1_3(\beta_2) \uplus \phi)(\beta_1)) \uplus \ldots)) & = \\
\overset{(\text{T.3.6,3.7})}{=} & \ s^1_2(\mathrm{ap}^{!N}_{A_1\cup A_2}(\phi)(\mathrm{ap}^{!N}_{A_1}(s^1_3(\beta_2) \uplus s^1_1(\phi))(\beta_1)) \uplus \ldots
\end{aligned}
$$

Let $n \in \mathbb{N}, c \in \mathrm{N}$:

$$
\begin{aligned}
& \mathrm{ap}^{!N}_{A_1\cup A_2}(\phi)(\mathrm{ap}^{!N}_{A_1}(s^1_3(\beta_2) \uplus s^1_1(\phi))(\beta_1))(n)(c) & = \\
= & \begin{cases} \mathrm{ap}_{A_1\cup A_2}(\phi) \copyright \mathrm{ap}_{A_1}(s^1_3(\beta_2) \uplus s^1_1(\phi)) \copyright \beta_1(n)(c) \\ \qquad \text{if } !c \in \mathrm{ap}_{A_1\cup A_2}(\phi) \cap \mathrm{ap}_{A_1}(s^1_3(\beta_2) \uplus s^1_1(\phi)) \\ \epsilon \ \text{else} \end{cases} & = \\
\overset{(\text{L.3.19})}{=} & \begin{cases} \mathrm{ap}_{A_1}(s^1_3(\beta_2) \uplus s^1_1(\phi)) \copyright \beta_1(n)(c) \text{ if } !c \in \mathrm{ap}_{A_1}(s^1_3(\beta_2) \uplus s^1_1(\phi)) \\ \epsilon \qquad \text{else} \end{cases} & = \\
= & \ \mathrm{ap}^{!N}_{A_1}(s^1_3(\beta_2) \uplus s^1_1(\phi))(\beta_1)(n)(c)
\end{aligned}
$$

□

3.11.3 Renaming Congruence

We will show that if and only if $\tau(f) \in C_{1I_1,P_1} \bar{\otimes} C_{2I_2,P_2}$ then $f \in \tau(C_{1I_1,P_1}) \bar{\otimes} \tau(C_{2I_2,P_2})$.

As usual, for all splits s let s' be the 'renamed split', such that: $\tau(s^i(\phi)) = s'^i(\tau(\phi))$, $\tau(\phi \uplus \psi) = \tau(\phi) \uplus \tau(\psi)$. Or, reading the proof backwards, s is the renamed split to s' with respect to τ^{-1}.

$\phi \in$ NMS:

$$
\begin{aligned}
\tau(f(\tau^{-1}(\phi))) &= \tau(s_2^1(\beta_1) \uplus s_3^2(\beta_2)) \\
&= {s'}_2^1(\tau(\beta_1)) \uplus {s'}_3^2(\tau(\beta_2))\,; \\
\tau(\beta_1) &= \tau(f_1(s_1^1(\tau^{-1}(\phi)) \uplus s_3^1(\beta_2))) \;=\; \tau(f_1(\tau^{-1}({s'}_1^1(\phi) \uplus {s'}_3^1(\tau(\beta_2))))) \\
\tau(\beta_2) &= \tau(f_2(s_1^2(\tau^{-1}(\phi)) \uplus s_2^2(\beta_2))) \;=\; \tau(f_2(\tau^{-1}({s'}_1^2(\phi) \uplus {s'}_2^2(\tau(\beta_1)))))
\end{aligned}
$$

Appendix A

Basic Definitions

In this section we gathered all definitions which are not central to the presented theories, but have to be object of a necessary agreement.

A.1 General

A.1.1 Partial Functions

$A \rightharpoonup B$ denotes the set of partial functions from A to B.

$f \in A \rightharpoonup B$: $dom(f)$ denotes the domain of f.

$f \in A \rightharpoonup B$: $codom(f)$ denotes the codomain of f (its image).

Remark: $x \notin dom(f)$: $f(x)$ is undefined and should not influence the value of any expression in that case.

A.1.2 Natural Numbers

$\mathbb{N}$ denotes the set of natural numbers *including zero*.
$\mathbb{N}^\infty := \mathbb{N} \cup \{\infty\}$
where $\forall n \in \mathbb{N} : n < \infty$ and $\infty + n = \infty - n = \infty + \infty = \infty$ is assumed.

A.2 Multisets

We can enrich the concept of sets with an explicit representation of quantities. A set is a mappping from the respective objects to a boolean set: 2^M, each object is either present or not. A *multiset* maps each object onto its quantity: $\mathbb{N}^M$. We denote the set of multisets with M^+.

We have to define some of the usual set operations in their application on multisets (e.g. $\hat{\subseteq}$) and in their application on mixed arguments (e.g. $\subseteq$ applied on a set and a multiset).

A.2.1 The Empty Set: $\hat{\emptyset}$

The empty set, as usual, contains no element. That means its quantity in every element is zero: $x \in M : \hat{\emptyset}(x) := 0$.

A.2.2 Elements

Certainly, a element is *in* a set if its quantity is not zero:

$A \in M^+, x \in M : \quad x \in A :\Leftrightarrow A(x)\, 0$

A.2.3 The Submultiset Relation: $\hat{\subseteq}$

We demand every element of the subset to be in the superset in a bigger quantity:

$A, B \in M^+ : \quad A \hat{\subseteq} B :\Leftrightarrow \forall x : A(x) \leq B(x)$

A.2.4 The Multiset Union: $\uplus$

This union, in contrast to its classical counterpart, also takes quantities into account:

$A, B \in M^+ : \forall x \in M : (A \uplus B)(x) := A(x) + B(x)$

A.2.5 The Subset Relation: $\subseteq$

Sometimes we want to discuss subset relation without taking regards of quantities:

$A, B \in M^+ \cup M^* : \quad A \subseteq B :\Leftrightarrow \forall x : x \in A \Rightarrow x \in B$

A.2.6 Filters: ©

A set can be used as a filter on multisets: Each multiset element which is not also in the set is thereby removed:

$$A \in M^*, B \in M^+ : \quad \forall x \in M : (A © B)(x) := \begin{cases} B(x) & \text{if } x \in A \\ 0 & \text{else} \end{cases}$$

A.3 Multisets, Lists and Streams

Our view of lists is based on that of streams as functions from $\mathbb{N}$ to the respecive set. We consider lists as functions from subsets of $\mathbb{N}$ to the set to obtain a uniform notation for streams and lists.

A.3.1 Lists

$S^* := \bigcup_{n \in \mathbb{N}} \{0, \ldots, n-1\} \to S$
where $\emptyset \to S = \{\epsilon\}$ and ϵ is treated as the nowhere defined function.

A.3.2 Streams

$S^\infty := \mathbb{N} \to S$ are the *streams* (infinite lists) over S.

A.3.3 Lists and Streams

Let N be a set of names:
$(N \to S)^*$ is the set of named Lists.
$(N \to S)^\infty$ is the set of named Streams.

A.3.4 Operations on Lists and Streams

Length

$s \in S^* \cup S^\infty$:

$$|s| := \begin{cases} n & \text{if } s \in \{0, \ldots, n-1\} \to S \\ \infty & \text{if } s \in \mathbb{N} \to S \end{cases}$$

Appending

For lists or streams with head h and tail t we will, as usual, write $h.t$. More precisely: A list is either empty: ϵ or if l is a list and e is an element, then $h.l$ is also a list.

Prefixes

$s \in S^* \cup S^\infty, n \leq |s|$:

$|s\downarrow_n| = n \ \wedge \forall m \leq n : s\downarrow_n(m) = s(m)$ especially: $s\downarrow_0 = \epsilon,\ |s| = \infty : s\downarrow\infty = s$

Suffixes

$s \in S^* \cup S^\infty, n \leq |s|$:

$|s\uparrow_n| = |s| - n \ \wedge \forall m \leq |s| - n : s\uparrow_n(m) = s(m+n)$ especially: $s\uparrow_0 = s,\ |s| = \infty : s\uparrow\infty = \epsilon$

Concatenation

$s_1, s_2 \in S^* \cup S^\infty, n < |s_1| + |s_2|$:

$$(s_1 \circ s_2)(n) \;:=\; \begin{cases} s_1(n) & if n < |s_1| \\ s_2(n - |s_1|) & if n \geq |s_1| \end{cases}$$

Prefix Relation

$s_1, s_2 \in S^* \cup S^\infty$:

$s_1 \sqsubseteq s_2 \;:\Leftrightarrow\; \exists s \in S^* \cup S^\infty : s_1 \circ s \;=\; s_2$

A.3.5 Functions on Streams of Named Multisets

Multiset Union

$s_1, s_2 \in (N \to S^+)$:

$x \in S$: $(s_1 \uplus s_2)(x) := \; s_1(x) \uplus s_2(x)$

$s_1, s_2 \in (N \to S^+)^* \cup (N \to S^+)^\infty, x \in S, n < |s_1| = |s_2|$:

$(s_1 \uplus s_2)(n)(x) \;:=\; s_1(n)(x) \uplus s_2(n)(x)$

Filters

The multiset filter can be canonically extended on streams of multisets:

$F \subseteq S, s \in (N \to S^+) \cup (N \to S^\infty), n \in N$:

$(F \copyright s)(n) \;:=\; F \copyright (s(n))$

$F \subseteq S, s \in (S^+)^* \cup (S^*)^\infty, n \leq |s|$:

$(F \copyright s)(n) \;:=\; F \copyright (s(n))$

Set Functions

$s_1, s_2 \in (N \to S^+)$:

$s_1 \subseteq s_2 \;:\Leftrightarrow\; \forall n \in N : s_1(n) \subseteq s_2(n)$

$s_1, s_2 \in (N \to S^+)^* \cup (N \to S^+)^\infty, |s_1| \;=\; |s_2|$:

$s_1 \subseteq s_2 \;:\Leftrightarrow\; \forall n \leq |s_1| : s_1(n) \subseteq s_2(n)$

A.3.6 Functions on Lists and Streams of Lists

Filters

$F \subseteq S, h.t \in S^*$:

$F \copyright \epsilon \;:=\; \epsilon$

$$F \copyright h.t \ := \ \begin{cases} F \copyright t & \text{if } h \in F \\ h.\copyright t & \text{if } h \notin F \end{cases}$$

$F \subseteq S, s \in (N \to S^*) \cup (N \to S^\infty), n \in N :$

$$(F \copyright s)(n) \ := \ F \copyright (s(n))$$

$F \subseteq S, s \in (S^*)^* \cup (S^*)^\infty, n \leq |s| :$

$$(F \copyright s)(n) \ := \ F \copyright (s(n))$$

Set Functions

$h.t, l_1, l_2 \in S^*, x \in S :$

$$x \notin \epsilon$$

$$x \in h.t \ :\Leftrightarrow \ x = h \ \vee \ x \in t$$

$$l_1 \subseteq l_2 :\Leftrightarrow \ \forall x \in l_1 : x \in l_2$$

$s_1, s_2 \in (N \to S^*), |s_1| = |s_2| :$

$$s_1 \subseteq s_2 \ :\Leftrightarrow \ \forall n \in N : s_1(n) \subseteq s_2(n)$$

$s_1, s_2 \in (N \to S^*)^* \cup (N \to S^*)^\infty, |s_1| = |s_2| :$

$$s_1 \subseteq s_2 \ :\Leftrightarrow \ \forall n \leq |s_1| : s_1(n) \subseteq s_2(n)$$

Appendix B

Metric Space Basics

B.1 Metric Spaces

This section was, because of its generality, simply taken from [GS96].

The fundamental concept in metric spaces is the concept of distance.

Definition B.1 *Metric Spaces*
A metric space *is a pair (D, d) consisting of a nonempty set D and a mapping $d \in D \times D \to R$, called a* metric *or* distance, *which has the following properties:*

(1) $\forall x, y \in D : \quad d(x, y) = 0 \quad \Leftrightarrow \quad x = y$

(2) $\forall x, y \in D : \quad d(x, y) = d(y, x)$

(3) $\forall x, y, z \in D : d(x, y) \leq d(x, z) + d(z, y)$

A very simple example of a metric is the discrete metric.

Definition B.2 *The discrete metric*
The discrete metric (D, d) *over a set D is defined as follows:*

$$d(x, y) = \begin{cases} 0 \ \textit{if}\ x = y \\ 1 \ \textit{if}\ x \neq y \end{cases}$$

B.2 Convergence

Measuring the distance between the elements of a sequence $(x_i)_{i \in \mathbb{N}}$ in D we obtain the familiar definitions for convergence and limits.

Definition B.3 *Convergence and limits*
Let (D, d) be a metric space and let $(x_i)_{i \in \mathbb{N}}$ be a sequence in D.

(1) *We say that $(x_i)_{i \in \mathbb{N}}$ is a* Cauchy sequence *whenever we have:*
$\forall \epsilon > 0 : \exists N \in \mathbb{N} : \forall n, m > N : d(x_n, x_m) < \epsilon.$

(2) *We say that* $(x_i)_{i\in I\!N}$ converges to $x \in D$ *denoted by* $x = lim_{n\to\infty} x_i$ *and call* x *the* limit *of* $(x_i)_{i\in I\!N}$ *whenever we have:*

$\forall \epsilon > 0 : \exists N \in I\!N : \forall n > N : d(x_n, x) < \epsilon.$

(3) *The metric space* (D, d) *is called* complete *whenever each Cauchy sequence converges to an element of* D*.*

Theorem B.1
The discrete metric is complete.

Proof:

Each Cauchy sequence is constant from a given N.

B.3 Contractive Functions

A very important class of functions over metric spaces is the class of *Lipschitz functions*.

Definition B.4 *Lipschitz functions*
Let (D_1, d_1) *and* (D_2, d_2) *be metric spaces and let* $f \in D_1 \to D_2$ *be a function. We call* f Lipschitz function *with constant* c *if there is a constant* $c \geq 0$ *such that the following condition is satisfied:*

$d(f(x), f(y)) \leq c \cdot d(x, y)$

For a function f *with arity* n *the above condition generalizes to:*

$d(f(x_1, \ldots, x_n), f(y_1, \ldots, y_n)) \leq c \cdot max\{d(x_i, y_i) \mid i \in [1..n]\}$

If $c = 1$ *we call* f non-expansive. *If* $c < 1$ *we call* f contractive.

Theorem B.2
The composition of two Lipschitz functions $f \in D_1 \to D_2$ *and* $g \in D_2 \to D_3$ *is a Lipschitz function with constant* $c_1 \cdot c_2$*.*

Proof:

$d(g(f(x_1)), g(f(x_2)) \leq c_2 \cdot d(f(x_1), f(x_2)) \leq c_2 \cdot c_1 \cdot d(x_1, x_2)$

Corollary B.3
The composition of a contractive and a non-expansive function is contractive. The composition of two non-expansive functions is non-expansive. Identity is non-expansive. □

The main tool for handling recursion in metric spaces is the Banach's fixed point theorem. It guarantees the existence of a unique fixed point for every contractive function.

Theorem B.4 *Banach's fixed point theorem* . *Let* (D, d) *be a complete metric space and* $f \in D \to D$ *a contractive function. Then there exists an* $x \in D$*, such that the following holds:*

(1) $x = f(x)$ *(x is a fixed point of f)*

(2) $\forall y \in D: \ y = f(y) \Rightarrow y = x$ *(x is unique)*

(3) $\forall z \in D: \ x = lim_{n \to \infty} f^n(z)$ *where*

$$f^0(z) \quad = z$$
$$f^{n+1}(z) = f(f^n(z))$$

Proof:

See [Eng77] or [Sut75].

Usually we want to use a parameterized version of this theorem.

Definition B.5 *Parameterized fixed point*
Let $f \in D \times D_1 \times \ldots \times D_n \to D$ be a function of non-empty complete metric spaces that is contractive in its first argument. We define the parameterized fixed point *function μf as follows:*

$$(\mu f) \in D_1 \times \ldots \times D_n \to D$$
$$(\mu f)(y_1, \ldots, y_n) = x$$

where x is the unique element of D such that $x = f(x, y_1, \ldots, y_n)$ as guaranteed by Banach's fixed point theorem.

Theorem B.5
If f is contractive (non-expansive) so is μf.

Proof:

See for example [MPS86] pages 114–115.

B.4 The Metric of Streams

Definition B.6 *The metric of streams*
The metric of streams (E^∞, d) is defined as follows:

$$d(s,t) = inf\{2^{-j} \mid s \downarrow_j = t \downarrow_j\}$$

This metric is also known as the Baire metric [Eng77].

Theorem B.6
The metric space of streams (E^∞, d) is complete.

Proof:

See for example [Eng77].

Appendix C

Proposal for a Network Semantic for π–Terms

C.1 Preparatory Definitions

C.1.1 Basic Partial Components

To simplify the definition of our interpretation, we do some peparatory by defining basic partial components, on which our interpretation of sender receiver prefixes will be based.

Basic Channel Sender This components sends exactly one channel name, i.e., the two port names, on a certain channel. The time interval in which it sends is not determined, it can also be silent forever.

$x, y \in \mathrm{N}$:

$$\begin{aligned} !^c x(y) := \{f \in \mathrm{GCPB}_{\{!x,!y,?y\}} | \, & \forall \phi \in dom(f) : \exists n < |\phi| : \\ & (f(\phi)(n)(x) = \{!y, ?y\} \vee f(\phi)(n)(x) = \hat{\emptyset}) \wedge \\ & \forall m \neq n, m < |\phi| : \forall c \neq x : f(\phi)(m)(c) = \hat{\emptyset}\}_{\{!x,!y,?y\},\emptyset,\emptyset} \end{aligned}$$

Basic Message Sender Same as before, only that now a proper message is sent.

$x \in \mathrm{N}, y \in \mathrm{M}$:

$$\begin{aligned} !x(y) := \{f \in \mathrm{GCPB}_{\{!x\}} | \, & \forall \phi \in dom(f) : \exists n < |\phi| : \\ & (f(\phi)(n)(x) = \{y\} \vee f(\phi)(n)(x) = \hat{\emptyset}) \wedge \\ & \forall m \neq n, m < |\phi| : \forall c \neq x : f(\phi)(m)(c) = \hat{\emptyset}\}_{\{!x\},\emptyset,\emptyset} \end{aligned}$$

Basic Channel Receiver This receiver waits until at least one time the specified channel name was received on the specified channel.

$x, y \in \mathrm{N}:$

$$
\begin{aligned}
?^c x(y) := \{ & f \in \mathrm{GCPB}_{\{?x\}} | \forall \phi \in dom(f), n < |\phi|, c \in \mathrm{N}: \\
& f(\phi)(n)(c) = \hat{\emptyset} \wedge \\
& ((\exists n < |\phi| : ?y, !y \in \phi(n)(x)) \vee |\phi| = \infty\}_{?x,\emptyset,\{?y,!y\}}
\end{aligned}
$$

Basic Message Receiver Same as before, just for proper messages.

$x \in \mathrm{N}, y \in \mathrm{M}:$

$$
\begin{aligned}
?x(y) := \{ & f \in \mathrm{GCPB}_{\{?x\}} | \forall \phi \in dom(f), n < |\phi|, c \in \mathrm{N}: \\
& f(\phi)(n)(c) = \hat{\emptyset} \wedge \\
& ((\exists n < |\phi| : ?y, !y \in \phi(n)(x)) \vee |\phi| = \infty\}_{?x,\emptyset,\emptyset}
\end{aligned}
$$

C.1.2 Acknowledgements

A receiving component will have to acknowledge the 'channel name' (two port names) it receives. Only when the sender received the corresponding acknowledgement on the right channel, it will continue.

As acknowledgements we define a subset of M, consisting of one acknowledgement signal for each channel name. Therefore we use the constructor *ack*: $ack(\mathrm{N}) \subseteq \mathrm{M}$.

C.2 Interpretation for the π-Terms

We will write $[\![.]\!]$ for our interpreting function. $[\![.]\!]$ should be defined on all π-terms and result in generic components. $[\![.]\!]$ should also have many more properties, but we do not yet have any proofs of adequacy properties.

For the following π operators, confer to [Mil91], [MPW92a], and [MPW92b].

C.2.1 Sender Prefix

$$[\![\overline{x}y.t]\!] := !^c x(y).?x(ack(y)) \circ \tau([\![t]\!])$$

where τ ensures disjointness of the respective initial sets.

C.2.2 Receiver Prefix

$$[\![x(y).t]\!] := ?^c x(y).!x(ack(y)) \circ \tau([\![t]\!])$$

where τ ensures disjointness of the respective initial sets.

C.2.3 Sum

$$[\![\sum_{i\in I} t_i]\!] \;:=\; \begin{cases} \biguplus_{i\in I} \tau_i([\![t_i]\!]) & for\ I \neq \emptyset \\ \{NB\} & for\ I = \emptyset \end{cases}$$

where $\forall n \in \mathbb{N}, c \in \mathrm{N} : NB(\phi)(n)(c) := \epsilon$

and the τ_i ensure disjointness.

C.2.4 Parallel Composition

$$[\![t_1 \mid t_2]\!] \;:=\; [\![t_1]\!]\bar{\otimes}\tau([\![t_2]\!])$$

where τ ensures disjointness.

C.2.5 Hiding

$$[\![(\nu x)t]\!] \;:=\; \nu x : \tau([\![t]\!])$$

where τ ensures that x is not in the private set of $[\![t]\!]$.

C.2.6 Bang

We have to fulfill the equation $!P = P \mid !P$.

$$[\![!P]\!] := \biguplus_{n\in\mathbb{N}} \underbrace{[\![\tau_{n,i}(P) \mid \ldots \mid \tau_{n,n}(P)]\!]}_{\text{n times}}$$

where the $\tau_{i,j}$ ensure disjointness.

Bibliography

[BDD+95] Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, and Rainer Weber. The design of distributed systems — an introduction to FOCUS. Technical report, Technische Universität München, Institut für Informatik, 1995.

[Bro87] M. Broy. Semantics of finite and infinite networks of concurrent communicating agents. *Distributed Computing*, 2:13–31, 1987.

[BŞ95] Manfred Broy and Gheorghe Ştefănescu. The algebra of stream processing functions. Technical report, Institute of Mathematics, Romanian Academy, 1995.

[Eng77] R. Engelking. *General Topology.* PWN – Polish Scientific Publishers, 1977.

[GS96] Radu Grosu and Ketil Stoelen. A Model for Mobile Point-to-Point Data-flow Networks without Channel Sharing . In Martin Wirsing, editor, *AMAST'96.* LNCS, 1996.

[Kah74] G. Kahn. The semantics of a simple language for parallel programming. In *Proc. Information Processing 74*, pages 471–475. North-Holland, 1974.

[Mil91] Robin Milner. The polyadic π-calculus: A tutorial. Technical Report ECS-LFCS-91-180, LFCS, Department of Computer Science, University of Edinburgh, 1991.

[MPS86] D. MacQueen, G. Plotkin, and R. Sethi. An Ideal Model for Recursive Polymorphic Types. *Information and Control*, 71:95–130, 1986.

[MPW92a] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I. *Information and Computation*, 100:1–40, 1992.

[MPW92b] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part II. *Information and Computation*, 100:41–77, 1992.

[Par83] D. Park. The "fairness" problem and nondeterministic computing networks. In *Proc. 4th Foundations of Computer Science, Mathematical Centre Tracts 159*, pages 133–161. Mathematisch Centrum Amsterdam, 1983.

[Pau94a] Lawrence C. Paulson. Introduction to isabelle. at the TUM under: `/usr/proj/isabelle/doc/intro.dvi`, 1994.

[Pau94b] Lawrence C. Paulson. The isabelle reference manual. at the TUM under: `/usr/proj/isabelle/doc/ref.dvi`, 1994.

[Pau94c] Lawrence C. Paulson. Isabelle's object logics. at the TUM under: `/usr/proj/isabelle/doc/logic.dvi`, 1994.

[Rus90] J. R. Russell. On oraclizable networks and Kahn's principle. In *Proc. POPL'90*, pages 320–328, 1990.

[Sut75] W.A. Sutherland. *Introduction to metric and topological spaces.* Claredon Press - Oxford, 1975.