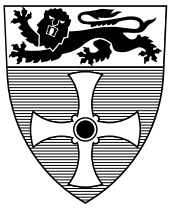
UNIVERSITY OF
NEWCASTLE



University of Newcastle upon Tyne

COMPUTING SCIENCE

Towards an Algebra of Abstractions for Communicating Processes

M. Koutny, Giuseppe Pappalardo and M. Pietkiewicz-Koutny.

TECHNICAL REPORT SERIES

No. CS-TR-949 Febuary, 2006

TECHNICAL REPORT SERIES

No. CS-TR-949 Febuary, 2006

Towards an Algebra of Abstractions for Communicating Processes

Maciej Koutny, Giuseppe Pappalardo, Marta Pietkiewicz-Koutny

Abstract

It is often desirable to describe the interface of an implementation system at a different (usually more detailed) level of abstraction to the interface of the relevant specification. This calls for a relation aimed at formalising the notion that a process is an acceptable implementation of another target process in the event that they possess different interfaces.

We conduct our investigation in the standard failures-divergences CSP process model, formulating a suitable implementation relation between the observable behaviours of the implementation and the target process. Interface difference and bridging is modelled by endowing the implementation relation with parameters, called extraction patterns, instrumental to interpreting implementation behaviour as target behaviour.

Reasonable notions of implementation and extraction patterns should result in a relation satisfying the realisability and compositionality properties. The former means that, if target and implementation in fact have the same interface, then the implementation relation between them collapses into the standard implementation pre-order of CSP. Compositionality allows a target composed of several connected systems to be implemented by connecting their respective implementations.

With respect to previous work, here we lift a restriction that prevented broadcast and other group communication to be modelled, and do without any constraint (other than divergence freedom) on admissible specification processes. Moreover, distributivity of the implementation relation over the network composition operator, i.e., compositionality, is extended to other useful process building operators, such as choice. Our novel approach is based on introducing operations over extraction patterns, mimicking (and being compatible with) operations over processes. We feel this constitutes a first step in the development of an algebra of abstraction for communicating processes in the CSP model.

Printed and published by the University of Newcastle upon Tyne,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

KOUTNY, M., PAPPALARDO, G., PIETKIEWICZ-KOUTNY, M..

Towards an Algebra of Abstractions for Communicating Processes
[By] M. Koutny and M. Pietkiewicz-Koutny.

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Science, 2006.

(University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. CS-TR-949)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE
Computing Science. Technical Report Series. CS-TR-949

Abstract

It is often desirable to describe the interface of an implementation system at a different (usually more detailed) level of abstraction to the interface of the relevant specification. This calls for a relation aimed at formalising the notion that a process is an acceptable implementation of another target process in the event that they possess different interfaces.

We conduct our investigation in the standard failures-divergences CSP process model, formulating a suitable implementation relation between the observable behaviours of the implementation and the target process. Interface difference and bridging is modelled by endowing the implementation relation with parameters, called extraction patterns, instrumental to interpreting implementation behaviour as target behaviour.

Reasonable notions of implementation and extraction patterns should result in a relation satisfying the realisability and compositionality properties. The former means that, if target and implementation in fact have the same interface, then the implementation relation between them collapses into the standard implementation pre-order of CSP. Compositionality allows a target composed of several connected systems to be implemented by connecting their respective implementations.

With respect to previous work, here we lift a restriction that prevented broadcast and other group communication to be modelled, and do without any constraint (other than divergence freedom) on admissible specification processes. Moreover, distributivity of the implementation relation over the network composition operator, i.e., compositionality, is extended to other useful process building operators, such as choice. Our novel approach is based on introducing operations over extraction patterns, mimicking (and being compatible with) operations over processes. We feel this constitutes a first step in the development of an algebra of abstraction for communicating processes in the CSP model.

About the author

Maciej Koutny obtained his MSc (1982) and PhD (1984) from the Warsaw University of Technology. In 1985 he joined the then Computing Laboratory of the University of Newcastle upon Tyne to work as a Research Associate. In 1986 he became a Lecturer in Computing Science at Newcastle, and in 1994 was promoted to an established Readership at Newcastle. In 2000 he became a Professor of Computing Science.

Marta Pietkiewicz-Koutny received her M.Sc. in Applied Mathematics from the Warsaw University of Technology in 1982. In 1984 she joined the Department of Operational Research, Institute of Econometrics in the Warsaw University of Economics where she worked as a junior lecturer until 1986. In 1987 she joined the Computing Laboratory of the University of Newcastle upon Tyne first as a research associate and then as a demonstrator (1988-1997). In the period 1997-2000 she was a Ph.D. student at the Department of Computing Science of the University of Newcastle upon Tyne, and in December 2000 she was awarded her Ph.D. degree.

Suggested keywords

BEHAVIOUR ABSTRACTION,
COMMUNICATING SEQUENTIAL PROCESSES,
COMPOSITIONALITY,
ALGEBRA OF ABSTRACTIONSSTRUCTURE AND BEHAVIOUR OF NETS,
THEORY OF REGIONS,
TRANSITION SYSTEMS

Towards an Algebra of Abstractions for Communicating Processes

Maciej Koutny
School of Computing Science, University of Newcastle, NE1 7RU, United Kingdom.
maciej.koutny@ncl.ac.uk

Giuseppe Pappalardo
Dipartimento di Matematica e Informatica, Università di Catania, I-95125 Catania, Italy,
pappalardo@dmi.unict.it

Marta Pietkiewicz-Koutny
School of Computing Science, University of Newcastle, NE1 7RU, United Kingdom.
marta.koutny@ncl.ac.uk

Abstract

It is often desirable to describe the interface of an implementation system at a different (usually more detailed) level of abstraction to the interface of the relevant specification. This calls for a relation aimed at formalising the notion that a process is an acceptable implementation of another target process in the event that they possess different interfaces.

We conduct our investigation in the standard failures-divergences CSP process model, formulating a suitable implementation relation between the observable behaviours of the implementation and the target process. Interface difference and bridging is modelled by endowing the implementation relation with parameters, called extraction patterns, instrumental to interpreting implementation behaviour as target behaviour.

Reasonable notions of implementation and extraction patterns should result in a relation satisfying the realisability and compositionality properties. The former means that, if target and implementation in fact have the same interface, then the implementation relation between them collapses into the standard implementation pre-order of CSP. Compositionality allows a target composed of several connected systems to be implemented by connecting their respective implementations.

With respect to previous work, here we lift a restriction that prevented broadcast and other group communication to be modelled, and do without any constraint (other than divergence freedom) on admissible specification processes. Moreover, distributivity of the implementation relation over the network composition operator, i.e. compositionality, is extended to other useful process building operators, such as choice. Our novel approach is based on introducing operations over extraction patterns, mimicking (and being compatible with) operations over processes. We feel this constitutes a first step in the development of an algebra of abstraction for communicating processes in the CSP model.

Keywords: *Behaviour abstraction, communicating sequential processes, compositionality, algebra of abstractions.*

Introduction

To explain the basic ideas behind our approach, let us consider a *specification* network, P_{net}, composed of n communicating processes $P_1, \ldots, P_n$, and a corresponding *implementation* network, Q_{net}, also composed of n processes, $Q_1, \ldots, Q_n$. Intuitively, P_i is intended to be Q_i's specification; we shall also refer to P_i as a *target* or *base* system, and to Q_i as a *source* or *implementation* system. Note that P_i's and Q_i's interfaces need not coincide. However, we assume that the interface of Q_{net} at the boundary with the external environment is the same as that of P_{net}, and that we are not interested in the details of inter-process communication within the networks P_{net} and Q_{net}. In the CSP notation, this means that the specification network P_{net} is of the form $(P_1 \parallel P_2 \parallel \cdots \parallel P_n)\backslash A$, while the implementation network is of the form $Q_{net} \stackrel{\text{df}}{=} (Q_1 \parallel Q_2 \parallel \cdots \parallel Q_n)\backslash B$ (see Figure 1).

In the usual treatment of process algebras, such as [11, 13], the notion that Q_{net} *implements* P_{net} is based on the idea that Q_{net} is more deterministic than (or equivalent to) P_{net} in terms of the chosen semantics. In practice, to formally verify that such a property holds, one can proceed in either of the following two ways.

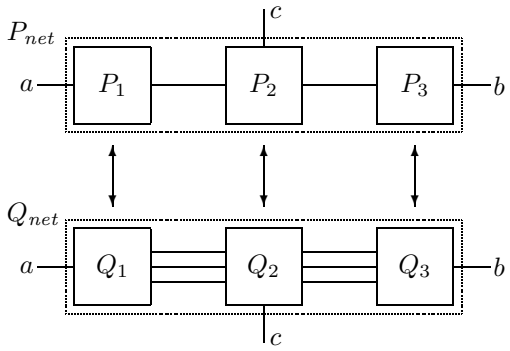


Figure 1. To establish that Q_{net} correctly implements P_{net}, it suffices to show that each of the processes Q_i correctly implements process P_i according to a well-chosen implementation relation, even when the actions in which they may respectively engage are different.

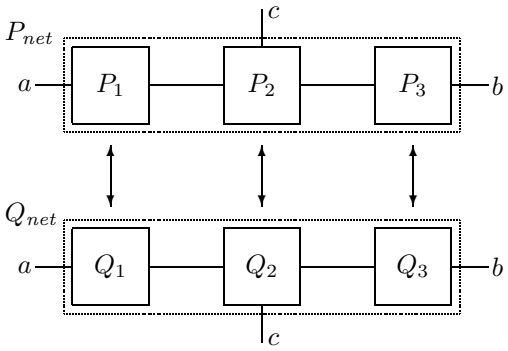


Figure 2. Network connectivity where each Q_i has the same observable actions as P_i.

- The first approach is to compare P_{net} and Q_{net} according to a chosen notion of refinement or equivalence. This is a straightforward approach, but one which potentially suffers from a severe state space explosion, since the compared processes are obtained by combining n components.

- The second, usually much better, approach attempts to (i) show an appropriate refinement or equivalence holds for each pair of processes P_i and Q_i, and (ii) resort to general theorems to infer the desired relation between the complete networks, P_{net} and Q_{net}.

Within the standard approaches, such as [11, 13], the latter *compositional* way of proving correctness of the implementation network is handled *only* in the case when for each i, P_i and Q_i have the same actions in their interfaces.[1] Yet in deriving an implementation from a specification we will often wish to implement abstract, high-level *interface* actions at a lower level of detail and in a more concrete manner. For example, the link available to connect base components P_i, P_j may be unreliable, and so may need to be implemented, in the sources Q_i, Q_j, by a pair of channels, one for data and one for acknowledgements. Or an intended implementation $\overline{Q_i}$ of P_i may be liable to fail itself, so that the final implementation Q_i is built by assembling redundant replicas of $\overline{Q_i}$, and thus has each channel of P_i replicated [9] (such a scenario was one of the original motivations behind the current work [10, 8]). Or it may simply be the case that a high-level action of P_i is rendered in a more concrete, and hence more implementable, form. As a result, the interface of an implementation process may exhibit a lower (and so different) level of abstraction than a specification process.

In the process algebraic context, dealing with *interface difference* necessitates the development of what Rensink and Gorrieri [12] have termed a *vertical implementation* relation. This should adequately capture the nature of the relationship between a specification and an implementation whose interfaces differ; and should collapse into the standard, *horizontal* one whenever the two interfaces happen to coincide. In works [10, 8, 4, 3], we independently identified this 'collapsing' requirement as *accessibility* or *realisability*, effectively pioneering it within the CSP process model [6, 13].

Technically, in our preceding works [4, 3], processes are formalised using Hoare's CSP language, with its standard failures-divergences semantics [6, 13]. The implementation relation is formulated in terms of failures and divergences of the implementation and target processes. Interface difference is modelled by endowing the implementation relation with parameters called extraction patterns. These are intended to interpret implementation behaviour as target behaviour (translating, in particular, traces), and suitably constrain the former in connection to acceptable refusals.

We developed the theory of [4, 3] under that crucial restriction of the *one-to-one* communication paradigm. I.e., we assumed no communication action is shared by more than two processes within a distributed network of processes (as is the case in Figure 1). This, in particular, excluded systems with broadcast, as well as disallowed, say, a potentially complex group protocol at the implementation level to be abstracted into a single action at the specification level. Another limitation of [4, 3] was that it only established the implementation relation to distribute over network composition (i.e. compositionality), not other useful operations employed to combine and construct processes, notably choice. We now set out to overcome such compromises.

The paper is organised as follows. In the next section,

[1] Hence the networks in Figure 1 cannot really be treated by this technique, as the constituent processes should be like in figure 2.

we introduce some basic notions used throughout the paper. The following section motivates our treatment on intuitive grounds, and is followed by sections introducing novel formalisations of extraction patterns, the implementation relation, and the associated new compositionality results.

1. The CSP model

In this section we recall the basic notions involved in the formal model of communicating systems used throughout the paper.

1.1. Actions and traces

Processes are represented in this paper using the failures-divergences model of Communicating Sequential Processes (CSP) [6, 13] — a formal model for the description of concurrent computing systems.

A CSP *process* can be regarded as a black box which may engage in interaction with its environment. Atomic instances of this interaction are called *actions* and must belong to the *alphabet* of the process. A *trace* of the process is a finite sequence of actions that a process can be observed to engage in.

The following notations are similar to that of [6] (below $t, u, t_1, t_2, \ldots$ are traces; a is an action; A is a set of actions; and T, T' are non-empty sets of traces):

- $t = \langle a_1, \ldots, a_n \rangle$ is the trace whose i-th element is a_i, and length, denoted $|t|$, is n.
- $t \circ u$ is the trace obtained by appending u to t.
- A^* is the set of all traces of actions from A, including the empty trace, $\langle\,\rangle$.
- $\leq$ denotes the prefix relation on traces.
- $Pref(T) \stackrel{\text{df}}{=} \{u \mid \exists t \in T :\ u \leq t\}$ is the *prefix-closure* of T.
- $t{\restriction}A$ is a trace obtained by deleting from t all the actions that do not belong to A.
- $t_1, t_2, \ldots$ is an *ω-sequence* of traces if $t_1 \leq t_2 \leq \ldots$ and $\lim_{i\to\infty} |t_i| = \infty$.
- A mapping $f : T \to T'$ is *monotonic* if $t, u \in T$ and $t \leq u$ implies $f(t) \leq f(u)$.
- A mapping $f : T \to T'$ is *strict* if $\langle\,\rangle \in T$ and $f(\langle\,\rangle) = \langle\,\rangle$.

To improve readability, we will sometimes use structured actions of the form $b.v$, where v is a *message* and b is a communication *channel*. We will then also talk about the set of channels of a process rather than the set of its actions.

1.2. Failures and divergences

We use the standard failures-divergences model of CSP [6, 13] in which a process P is a triple $(\alpha P, \phi P, \delta P)$ where αP (the *alphabet*) is a non-empty finite set of actions, ϕP (the *failures*) is a subset of $\alpha P^* \times \mathbb{P}(\alpha P)$,[2] and δP (the *divergences*) is a subset of αP^*. For a valid process P, its components should satisfy the conditions given below, where τP denotes the set of *traces* of P, $\tau P \stackrel{\text{df}}{=} \{t \mid \exists R \subseteq \alpha P :\ (t, R) \in \phi P\}$:

- τP is non-empty and prefix-closed.
- If $(t, R) \in \phi P$ and $S \subseteq R$ then $(t, S) \in \phi P$.
- If $(t, R) \in \phi P$ and $a \in \alpha P$ is such that $t \circ \langle a \rangle \notin \tau P$ then $(t, R \cup \{a\}) \in \phi P$.
- If $t \in \delta P$ then $(t \circ u, R) \in \phi P$, for all $u \in \alpha P^*$ and $R \subseteq \alpha P$.

If $(t, R) \in \phi P$ then P is said to *refuse* R after performing trace t. Intuitively, this means that P can deadlock, should the environment offer R as the set of possible actions to be executed after t. If $t \in \delta P$ then P is said to *diverge* after t. In the CSP model this means the process behaves in a totally uncontrollable way. Such a semantical treatment is based on what is often referred to as 'catastrophic' divergence, whereby the process in a diverging state is modelled as being able to engage in any action and generate any refusal (cf. last condition above).

A *base* process will be any divergence-free process P. Note that this is a minimal requirement which should always be made about a specification process as in CSP divergence signifies an unacceptable design fault.

1.3. Process operators

We shall now recall process operators used throughout the paper. These are the chief ones of CSP [6, 13].

Parallel composition $P \| Q$ models synchronous communication between processes in such a way that each of them is free to engage independently in any action that is not in the other's alphabet, but they have to engage simultaneously in any action that is in the intersection of their alphabets. Formally,

- $\alpha(P \| Q) \stackrel{\text{df}}{=} \alpha P \cup \alpha Q$.
- $\delta(P \| Q)$ comprises all traces $t \circ u$ such that the projections $(t \lceil \alpha P, t \lceil \alpha Q)$ belong to $\tau P \times \delta Q$ or $\delta P \times \tau Q$, and $u \in (\alpha P \cup \alpha Q)^*$.

[2] Note that $\mathbb{P}(X)$ denotes the *powerset* (set of subsets) of a set X.

- $\phi(P\|Q)$ comprises all failures $(t, R \cup S)$ such that $(t\lceil\alpha P, R) \in \phi P$ and $(t\lceil\alpha Q, S) \in \phi Q$, as well as all the elements of $\delta(P\|Q) \times \mathbb{P}(\alpha(P\|Q))$.

Parallel composition is commutative and associative. We shall use $P_1\| \cdots \|P_n$ to denote the parallel composition of processes $P_1, \ldots, P_n$.

Let P be a process and A be a set of actions of P. Then $P\backslash A$ is a process that behaves like P with the actions in A made invisible. Formally,

- $\alpha(P\backslash A) \stackrel{\text{df}}{=} \alpha P - A$.
- $\delta(P\backslash A)$ comprises all traces $t\lceil\alpha(P\backslash A) \circ u$ such that $u \in \alpha(P\backslash A)^*$, and $t \in \delta P$ or there exist $a_1, a_2, \ldots \in \alpha A$ such that for all $n \geq 1$, $t \circ \langle a_1, \ldots, a_n\rangle \in \tau P$).
- $\phi(P\backslash A)$ comprises all failures $(t\lceil\alpha(P\backslash A), R)$ such that $(t, R \cup A) \in \phi P$, as well as all the elements of $\delta(P\backslash A) \times \mathbb{P}(\alpha(P\backslash A))$.

Hiding is associative in that $(P\backslash A)\backslash A' = P\backslash(A \cup A')$.

In the next two operations on processes it is assumed that P and Q have the same alphabets. Then the deterministic choice ($P[]Q$) and non-deterministic choice ($P \sqcap Q$) are processes with the same alphabet as P and Q, the divergences being the union of those of P and Q, and the failures given respectively by:

- $\phi(P[]Q)$ comprises all failures $(\langle\,\rangle, R)$ belonging to $\phi P \cap \phi Q$, as well as all the failures (t, R) belonging to $\phi P \cup \phi Q$ such that $t \neq \langle\,\rangle$.
- $\phi(P \sqcap Q)$ is the union of ϕP and ϕQ.

The last operator is prefixing. Assuming that a is an action in the alphabet of a process P, $a \to P$ is the process with the same alphabet as P and

- $\delta(a \to P) \stackrel{\text{df}}{=} \{\langle a\rangle \circ t \mid t \in \delta P\}$.
- $\phi(a \to P)$ comprises all failures $(\langle a\rangle \circ t, R)$ such that $(t, R) \in \phi P$, as well as all the elements of $\{\langle\,\rangle\} \times \mathbb{P}(\alpha P - \{a\})$.

We also use STOP_A, or simply STOP if A is clear from the context, to denote a deadlocked process with the alphabet A.

In the examples, we use simple (mutually) recursive process definitions of the form $P \stackrel{\text{df}}{=} E$, where E is an expression built using the prefix, deterministic and non-deterministic choice, and STOP constructs. For example, $P \stackrel{\text{df}}{=} (a \to P)[](b \to \text{STOP})$ defines a process which can execute action a any number of times, and then perhaps execute b and terminate. It is beyond the scope of this paper to give a precise treatment of recursive processes, and the reader is referred to, e.g., [13] for a full account of this aspect of CSP.

2. Behaviour abstraction

Consider three base (or specification) processes: GEN, BUF_e and BUF_f, as shown in Figure 3(a). GEN generates an infinite sequence $010101\ldots$ of messages, or an infinite sequence $101010\ldots$ of messages on its output channel d, responding to the initial message (0 or 1) received on its input channel, c, at the beginning of its execution (note that, formally, the process executes actions in its alphabet $\alpha\text{GEN} = \{c.0, c.1, d.0, d.1\}$). BUF_e and BUF_f are buffer processes of capacity one, forwarding messages received on their shared input channel, d. Note that this means that the actions $d.0$ and $d.1$ are shared by all three processes, and so this example does not conform to the one-to-one inter-process communication paradigm (in a way, GEN 'broadcasts' messages on its output channel d). In terms of CSP, we have:

$$\begin{aligned}
\text{GEN} &\stackrel{\text{df}}{=} (c.0 \to \text{GEN}_0)[](c.1 \to \text{GEN}_1)\\
\text{GEN}_i &\stackrel{\text{df}}{=} d.i \to d.(1-i) \to \text{GEN}_i\\
\text{BUF}_x &\stackrel{\text{df}}{=} (d.0 \to x.0 \to \text{BUF}_x)[](d.1 \to x.1 \to \text{BUF}_x)
\end{aligned}$$

where $i = 0, 1$ and $x = e, f$.

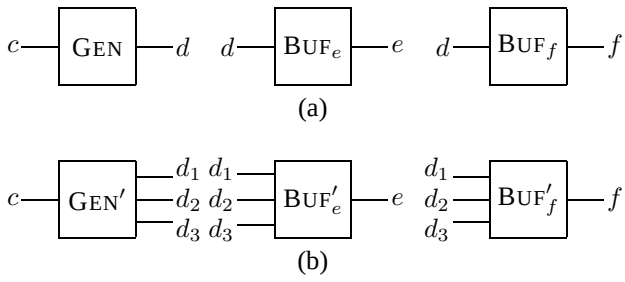


Figure 3. Three base processes (a); and their implementations (b). Note that c, d, e, f are channels and the actual executed actions are of the form $c.0, c.1, d.1$, etc.

Suppose that communication between the three processes at the shared channel d has been implemented using three channels, d_1, d_2 and d_3, as shown in Figure 3(b). The original transmissions on d became triplicated and the different copies are sent on the channels d_i ($i = 1, 2, 3$). That is, GEN' sends three copies of a message, while each of BUF'_e and BUF'_f forwards the first copy of the message ignoring the other two. This simple replication scheme works as it can be shown:

$$(\text{GEN}\|\text{BUF}_e\|\text{BUF}_f)\backslash\{d.0, d.1\} =$$
$$(\text{GEN}'\|\text{BUF}'_e\|\text{BUF}'_f)\backslash D\ ,$$

where $D \stackrel{\text{df}}{=} \bigcup_{i=1}^{3}\{d_i.0, d_i.1\}$. Suppose now that the transmission of messages is imperfect and two types of faulty

behaviour can occur: $\text{BUF}''_x \stackrel{\text{df}}{=} \text{BUF}'_x \sqcap \text{STOP}$ and $\text{BUF}'''_x \stackrel{\text{df}}{=} \text{BUF}'_x \sqcap \overline{\text{BUF}'_x}$ (for $x = e, f$), where $\overline{\text{BUF}'_x}$ is BUF'_x with all the communication on channel d_1 (for $x = e$) and d_2 (for $x = f$) being blocked. In other words, each BUF''_x can break down completely, refusing to input any message, while each BUF'''_x can only fail in such a way that although one of the channels d_i is dead, the other two can still be used to accept messages.[3] Since it can be shown that:

$$(\text{GEN} \| \text{BUF}_e \| \text{BUF}_f) \backslash \{d.0, d.1\} \neq (\text{GEN}' \| \text{BUF}''_e \| \text{BUF}''_f) \backslash D$$

and

$$(\text{GEN} \| \text{BUF}_e \| \text{BUF}_f) \backslash \{d.0, d.1\} = (\text{GEN}' \| \text{BUF}'''_e \| \text{BUF}'''_f) \backslash D \ , \quad (1)$$

it follows that each BUF'''_x is much 'better' an implementation of the BUF_x process than BUF''_x.

We will now analyse the differences in the behaviours of the processes BUF'''_x and BUF''_x, and at the same time introduce informally some basic concepts used subsequently.

We start by observing that the communication between the processes GEN', BUF'''_e and BUF'''_f can be thought of as adhering to the following two rules:

R1 The transmissions over d_1, d_2 and d_3 are consistent w.r.t. message content and can thus be directly related to transmissions over d.[4] The set of all traces over D satisfying such a property will be denoted by Dom.

R2 Transmission over d_3 is reliable, but there is no such guarantee for d_1 and d_2.

On the other hand, communication between the processes GEN', BUF''_e and BUF''_f satisfies the first rule, but fails to satisfy the second one, as the behaviour of BUF''_e and BUF''_f allows all channels d_i to be blocked. To express this difference formally, we need to render R1 and R2 in some form of precise notation.

To capture the relationship between traces of target and base processes on the corresponding channels, we will employ an (extraction) mapping $extr$ which for a trace in Dom returns its interpretation as a trace over $\{d.0, d.1\}$. For example, interpreting to abstract from replication yields

$$\begin{array}{rcl} \langle\,\rangle & \mapsto & \langle\,\rangle \\ \langle d_1.0\rangle & \mapsto & \langle d.0\rangle \\ \langle d_3.0\rangle & \mapsto & \langle d.0\rangle \\ \langle d_2.1, d_3.1\rangle & \mapsto & \langle d.1\rangle \\ \langle d_3.1, d_1.1, d_1.0\rangle & \mapsto & \langle d.1, d.0\rangle \end{array}$$

[3] One could think about this scenario as modelling the situation that in order to improve performance, a 'slow' channel d is replaced by three channels: two high-speed yet unreliable channels d_1 and d_2, and a slow but reliable backup channel d_3.

[4] This follows from the way GEN' produces its output.

Notice that the extraction mapping, denoted by $extr$, need only be defined for traces well-formed according to R1, i.e., those in Dom (for no other traces over D will be realised by the parallel composition of the processes GEN', BUF'''_e and BUF'''_f).

We further observe that, in view of R2, some of the traces in Dom may be regarded as *incomplete*. For example, $\langle d_1.1, d_3.1, d_1.0\rangle$ is such a trace since channel d_3 is reliable and so another copy of $d_1.0$ (i.e., $d_3.0$) is bound to be eventually transmitted. The set of all other traces in Dom — i.e., those which can be regarded as *complete* — will be denoted by dom.[5] For our example, dom will contain all traces in Dom where transmissions on d_1 and d_2 have not overtaken that on d_3.[6]

Although it will play a central role, the extraction mapping alone is not sufficient to identify the 'correct' implementation of BUF_x in the presence of faults, for both the good and the bad implementation can generate all the target's traces, up to extraction. I.e.:

$$\tau\text{BUF}_x = extr(T_x \cap \tau\text{BUF}''_x) = extr(T_x \cap \tau\text{BUF}'''_x)$$

where $x = e, f$ and T_x is the set of all $t \in (D \cup \{x.0, x.1\})^*$ such that $t \restriction D$ belongs to Dom.

What one also needs is an ability to relate the refusals of BUF''_x and BUF'''_x with the possible refusals of the base process BUF_x, so as to discriminate them on this basis. This, however, is much harder than relating traces. For suppose we attempted to 'extract' the refusals of BUF'''_e applying $extr$ to them too. Then, we would have had

$$\begin{array}{rcl} (\langle\,\rangle, \{d_1.0\}) & \in & \phi\text{BUF}'''_e \\ extr(\langle\,\rangle, \{d_1.0\}) = (\langle\,\rangle, \{d.0\}) & \notin & \phi\text{BUF}_e \ . \end{array}$$

Hence, BUF''''s behaviour (that of a buffer) would be interpreted as liable to refuse the initial input; yet we know from equation (1) that BUF''' is actually a "good" implementation.

Thus, we abandon the ambition of interpreting refusals made by implementations, and simply introduce a device intended to constrain good implementations not to deadlock when they shouldn't, i.e., essentially, when their interaction is not complete yet. In our example, this comes in the form of mappings ref_i ($i = 1, 2, 3$) thus defined, for each $t \in Dom$:

- $ref_1(t)$ — used for BUF'''_e — will contain all subsets R of D such that if $a \in R - \{d_1.0, d_1.1\}$ then $t \circ \langle a\rangle \notin Dom$.

[5] In general, $Dom = Pref(dom)$, meaning that each interpretable trace has, at least in theory, a chance of being completed.

[6] Another example which can commonly occur in practice is that if the whole sequence of actions $a_1, \ldots, a_k$ is extracted to a single action a, i.e., $\langle a_1, \ldots, a_i\rangle \mapsto \langle\,\rangle$ for $i < k$ and $\langle a_1, \ldots, a_k\rangle \mapsto \langle a\rangle$, then we do not consider a transmission complete unless the whole sequence $a_1, \ldots, a_k$ has been transmitted.

- $ref_2(t)$ — used for BUF'''_f — will contain all subsets R of D such that if $a \in R - \{d_2.0, d_2.1\}$ then $t \circ \langle a \rangle \notin Dom$.

- $ref_3(t)$ — used for GEN' — will contain all subsets R of D such that if $a \in R$ then $t \circ \langle a \rangle \notin Dom$.

The idea is that whenever these good implementations engage in a trace t' such that $t' \upharpoonright D = t$ belongs to Dom and is not complete, each of them should only refuse members of the relevant $ref_i(t)$ (and cannot exceed its maximals). As a result, it is easy to check that t' will not drive the implementations to a deadlock (i.e. together they cannot refuse the entire D).

More rigorously, within the treatment to come, based on the ref as well as the $extr$ mappings, one can define three *extraction patterns* $\mathfrak{ep}_i$ $(i = 1, 2, 3)$, such that GEN', BUF'''_e and BUF'''_f are implementations of GEN, BUF_e and BUF_f w.r.t. extraction patterns $\mathfrak{ep}_1$, $\mathfrak{ep}_2$ and $\mathfrak{ep}_3$, respectively. This, exploiting the results presented in the rest of this paper, can be used to infer that the equality

$$(\text{GEN} \| \text{BUF}_e \| \text{BUF}_f) \backslash \{d.0, d.1\} = (\text{GEN}' \| \text{BUF}'''_e \| \text{BUF}'''_f) \backslash D$$

holds and so prove the correctness of the three process implementation of the original network in Figure 3(a).

We will present another development as well. This is motivated by an observation that the parallel composition $\text{BUF}'''_e \| \text{BUF}'''_f$ can intuitively be seen as a valid implementation of the parallel composition process $\text{BUF}_e \| \text{BUF}_f$. The key question here is what would be the 'right' extraction pattern against which the validity of such an implementation can be measured. Clearly, this could not be $\mathfrak{ep}_1$ or $\mathfrak{ep}_2$ as they are based on the refusal mappings ref_1 and ref_2 which are each good for one of the processes in $\text{BUF}'''_e \| \text{BUF}'''_f$, but not both. The solution we will propose in this paper is to compose the two extraction patterns $\mathfrak{ep}_1$ and $\mathfrak{ep}_2$[7] into an extraction pattern $\mathfrak{ep}_1 \| \mathfrak{ep}_2$, and then prove a general theorem that $\text{BUF}'''_e \| \text{BUF}'''_f$ is a valid implementation of $\text{BUF}_e \| \text{BUF}_f$ w.r.t. $\mathfrak{ep}_1 \| \mathfrak{ep}_2$.

In the rest of this paper we will present a formalisation of the ideas outlined in this section. We will first define formally extraction patterns and the implementation relation parameterised by a set of extraction patterns together with a realisability result. After that, two composition operators on extraction patterns will be given together with a number of compositionality results.

[7] Basically, by combining together the refusal mappings ref_1 and ref_2 into a mapping ref so that $ref(t)$ will contain all subsets R of D such that if $a \in R - \{d_1.0, d_1.1, d_2.0, d_2.1\}$ then $t \circ \langle a \rangle \notin Dom$.

3. Extraction patterns

Extraction patterns (used in various forms in, e.g., [4, 3]) relate behaviour on a set of external channels in an implementation process to that on an external channel in the target process. They serve two main purposes: to interpret the behaviour and to encode some correctness requirements.

Definition 3.1 *An* extraction pattern *is a tuple* $\mathfrak{ep} \stackrel{\text{df}}{=} (Src, Trg, dom, extr, ref)$, *where:*

- Src *and* Trg *are non-empty sets of* source *and* target *actions, respectively.*
- $dom \subseteq Src^*$ *is a non-empty set of traces.*
- $extr$ *is a strict monotonic mapping from the traces in* $Dom \stackrel{\text{df}}{=} Pref(dom)$ *to the traces in* Trg^*.
- ref *is a mapping defined for traces* $t \in Dom$ *such that* $ref(t)$ *is a non-empty family of proper subsets of* Src. *It is assumed that:*
  - $R' \subset R \in ref(t)$ *implies* $R' \in ref(t)$.
  - *if* $a \in Src$ *and* $t \circ \langle a \rangle \notin Dom$ *then* $R \cup \{a\} \in ref(t)$, *for all* $R \in ref(t)$.

In the above, the mapping $extr$ interprets a trace over the actions from Src (in the implementation or source process) in terms of a trace over the actions from Trg (in the target process). The extraction mapping $extr$ is monotonic as receiving more information cannot decrease previous knowledge about the transmission.

The mapping ref is used to define correct behaviour in terms of failures as it gives bounds on refusals after execution of a particular trace sequence over the source actions. dom contains those traces in Dom for which the communication over Src may be regarded as complete; the constraint on refusals given by ref is only allowed to be violated for such traces (see also definition 4.1(4)). The intuition behind this requirement is that we cannot regard as correct a situation where deadlock occurs in the implementation process when behaviour is incomplete, for regarding this as correct behaviour might imply that the specification process could in some sense deadlock while in the middle of executing a single (atomic) action. $Src \notin ref(t)$ forbids a process to refuse all possible transmissions after an unfinished communication t. The last two properties required of ref reflect similar conditions imposed on the failures of a CSP process.

We lift some of the notions introduced above to a finite set of extraction patterns $\mathfrak{Ep} = \{\mathfrak{ep}_1, \ldots, \mathfrak{ep}_n\}$ $(n \geq 0)$, where

$$\mathfrak{ep}_i = (Src_i, Trg_i, dom_i, extr_i, ref_i)$$

for $i = 1, \ldots, n$, are such that $Src_i \cap Src_j = \varnothing$ and $Trg_i \cap Trg_j = \varnothing$ for $i \neq j$, in the following way:

- $Src_{\mathfrak{Ep}} \stackrel{\text{df}}{=} \bigcup_{i=1}^{n} Src_i$ and $Trg_{\mathfrak{Ep}} \stackrel{\text{df}}{=} \bigcup_{i=1}^{n} Trg_i$.
- $dom_{\mathfrak{Ep}}$ and $Dom_{\mathfrak{Ep}}$ are the sets of all traces t over $Src_{\mathfrak{Ep}}$ such that respectively $t\lceil Src_i \in dom_i$ and $t\lceil Src_i \in Dom_i$, for every $i \leq n$.

Moreover, we define a mapping $extr_{\mathfrak{Ep}}$ for any trace t such that $t\lceil Src_{\mathfrak{Ep}} \in Dom_{\mathfrak{Ep}}$, in the following way:

- $extr_{\mathfrak{Ep}}(\langle\,\rangle) \stackrel{\text{df}}{=} \langle\,\rangle$.
- for every $t \circ \langle a \rangle$ for which $extr_{\mathfrak{Ep}}$ is defined, we have $extr_{\mathfrak{Ep}}(t \circ \langle a \rangle) \stackrel{\text{df}}{=} extr_{\mathfrak{Ep}}(t) \circ u$, where the (possibly empty) u is such that if $a \in Src_i$, then

$$extr_i(t\lceil Src_i \circ \langle a \rangle) = extr_i(t\lceil Src_i) \circ u\ ,$$

and if $a \notin Src_{\mathfrak{Ep}}$ then $u = \langle a \rangle$.

Intuitively, $extr_{\mathfrak{Ep}}$ proceeds along a trace t from the beginning, leaving all the actions outside $Src_{\mathfrak{Ep}}$ unchanged, and extracting 'as soon as possible' all the actions from $Trg_{\mathfrak{Ep}}$ on the basis of the encountered actions from $Src_{\mathfrak{Ep}}$.

4. The implementation relation

Suppose that we intend to implement a base process P using another process Q with a possibly different communication interface. The correctness of the implementation will be expressed in terms of a set of extraction patterns, $\mathfrak{Ep}$ with $Src_{\mathfrak{Ep}} \subseteq \alpha Q$ and $Trg_{\mathfrak{Ep}} \subseteq \alpha P$. The actions which are to be related by $\mathfrak{Ep}$ will not necessarily be all the external actions of the processes P and Q. However, the actions not related by the extraction patterns will be the same in both P and Q, i.e., we will assume that $\alpha Q - Src_{\mathfrak{Ep}} = \alpha P - Trg_{\mathfrak{Ep}}$.

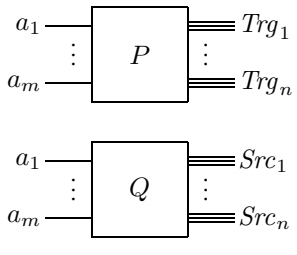


Figure 4. Base process P and its implementation Q. Note that $a_1, \ldots, a_m$ are actions, and Src_i, Trg_i (for $i = 1, \ldots, n$) are sets of actions.

Let P be a base process as in figure 4 and, for $i = 1, \ldots, n$, let

$$\mathfrak{ep}_i \stackrel{\text{df}}{=} (Src_i, Trg_i, dom_i, extr_i, ref_i)$$

be extraction patterns such that $Src_i \cap Src_j = \varnothing$ and $Trg_i \cap Trg_j = \varnothing$ for all $i \neq j$, and $Trg_{\mathfrak{Ep}} \subseteq \alpha P$, where $\mathfrak{Ep} = \{\mathfrak{ep}_1, \ldots, \mathfrak{ep}_n\}$ $(n \geq 0)$. We then take a process Q satisfying $\alpha Q = Src_{\mathfrak{Ep}} \uplus (\alpha P - Trg_{\mathfrak{Ep}})$, as shown in Figure 4. We then introduce some auxiliary notions:

- $\tau_{\mathfrak{Ep}} Q$ is defined as the set of all traces $t \in \tau Q$ such that $t\restriction Src_{\mathfrak{Ep}} \in Dom_{\mathfrak{Ep}}$ if the set of extraction patterns $\mathfrak{Ep}$ is non-empty, and $\tau_{\mathfrak{Ep}} Q = \tau Q$ if $\mathfrak{Ep} = \varnothing$.
- $\phi_{\mathfrak{Ep}} Q$ is the set of all $(t, R) \in \phi Q$ such that $t \in \tau_{\mathfrak{Ep}} Q$.
- $\delta_{\mathfrak{Ep}} Q$ is the set of all $t \in \delta Q$ such that $t \in \tau_{\mathfrak{Ep}} Q$.
- We will say that actions Src_i are *blocked* at a failure $(t, R) \in \phi_{\mathfrak{Ep}} Q$ if $Src_i \cap R \notin ref_i(t\lceil Src_i)$. We denote this by $i \in Blocked(t, R)$.

Note that $i \in Blocked(t, R)$ signifies that the refusal bound imposed by ref_i has been breached. Moreover, $\tau_{\mathfrak{Ep}} Q$, $\phi_{\mathfrak{Ep}} Q$ and $\delta_{\mathfrak{Ep}} Q$ are the only part of the behaviour of Q that is of interest.

Definition 4.1 *Under the above assumptions, Q is an* implementation *of P w.r.t. the set of extraction patterns $\mathfrak{Ep}$ if the following hold:*

1. $\delta_{\mathfrak{Ep}} Q = \varnothing$.
2. $extr_{\mathfrak{Ep}}(\tau_{\mathfrak{Ep}} Q) \subseteq \tau P$.
3. *If $t_1, t_2, t_3, \ldots$ is an ω-sequence of traces in $\tau_{\mathfrak{Ep}} Q$, then $extr_{\mathfrak{Ep}}(t_1), extr_{\mathfrak{Ep}}(t_2), extr_{\mathfrak{Ep}}(t_3), \ldots$ is an ω-sequence of traces in τP.*
4. *If Src_i is blocked at a failure $(t, R) \in \phi_{\mathfrak{Ep}} Q$ (i.e., $i \in Blocked(t, R)$), then $t\restriction Src_i \in dom_i$.*
5. *If $(t, R) \in \phi_{\mathfrak{Ep}} Q$ is such that $t\restriction Src_i \in dom_i$ for all $i \leq n$, then*

$$\Big(extr_{\mathfrak{Ep}}(t)\ , (R - Src_{\mathfrak{Ep}}) \cup \bigcup_{i \in Blocked(t,R)} Trg_i \Big) \in \phi P\ .$$

We denote this by $Q \preceq_{\mathfrak{Ep}} P$.

According to the above definition, within that part of the behaviour of Q that is of interest to us we have the following: (1) Q is divergence-free; (2) all its traces can be interpreted through $\mathfrak{Ep}$ as traces of P; (3) it is not possible to execute Q indefinitely without extracting any actions of P; (4) if refusals grow in excess of their bounds on a source action set Src_i, then communication on Src_i may be interpreted as locally completed for the set of actions Src_i; and (5) if a trace is locally completed for all sets of actions Src_i, then any blocking on a source action set Src_i in Q corresponds to the refusal of the whole Trg_i in P. The last condition can be regarded as a kind of refusal extraction.

A direct semantical comparison of an implementation process Q with the base process P is only possible if there is no difference in their communication interfaces, i.e., in the above definition $\mathfrak{Ep} = \varnothing$. Then we simply denote $Q \preceq P$. A comparison can then be made using the standard *refinement* ordering of CSP, denoted by $\sqsupseteq$ (formally, $Q \sqsupseteq P$ if $\alpha Q = \alpha P$, $\phi Q \subseteq \phi P$ and $\delta Q \subseteq \delta P$).

Theorem 4.2 *$Q \preceq P$ if and only if $Q \sqsupseteq P$.*

Proof: It suffices to observe that if $\mathfrak{Ep} = \varnothing$ then $\alpha P = \alpha Q$ and definition 4.1 reduces to the following: $\delta Q = \varnothing$, $\tau Q \subseteq \tau P$ and $\phi Q \subseteq \phi P$. □

In other words, if $\mathfrak{Ep} = \varnothing$ then the implementation relation collapses to the standard CSP refinement pre-order. In effect, this is the strongest kind of a *realisability* result one could realistically hope for.

It is also easy to see that the implementation relation is preserved through the refinement pre-order.

Theorem 4.3 *If $Q' \sqsupseteq Q$ and $Q \preceq_{\mathfrak{Ep}} P$ then $Q' \preceq_{\mathfrak{Ep}} P$.* □

5. Operations on extraction patterns

We will now introduce two operations on extraction patterns, corresponding to the parallel and non-deterministic choice compositions in CSP.

To start with, we will say that two extraction patterns $\mathfrak{ep}$ and $\mathfrak{ep}'$ are are *compatible* if $Src = Src'$, $Trg = Trg'$ and $extr(t) = extr'(t)$, for all $t \in Dom \cap Dom'$. In other words, if they operate on the same sets of source and target actions, and interpret in the same way traces belonging to the intersections of their domains. In such a case, we define an extraction pattern $\mathfrak{ep} \sqcap \mathfrak{ep}'$ as follows:

- $Src_{\mathfrak{ep}\sqcap\mathfrak{ep}'} \stackrel{\text{df}}{=} Src = Src'$.
- $Trg_{\mathfrak{ep}\sqcap\mathfrak{ep}'} \stackrel{\text{df}}{=} Trg = Trg'$.
- $dom_{\mathfrak{ep}\sqcap\mathfrak{ep}'} \stackrel{\text{df}}{=} dom \cup dom'$
  (and so $Dom_{\mathfrak{ep}\sqcap\mathfrak{ep}'} = Dom \cup Dom'$).
- For every trace $t \in Dom_{\mathfrak{ep}\sqcap\mathfrak{ep}'}$,

$$extr_{\mathfrak{ep}\sqcap\mathfrak{ep}'}(t) \stackrel{\text{df}}{=} \begin{cases} extr(t) & \text{if } t \in Dom \\ extr'(t) & \text{otherwise .} \end{cases}$$

$$ref_{\mathfrak{ep}\sqcap\mathfrak{ep}'}(t) \stackrel{\text{df}}{=} \begin{cases} ref(t) \cup ref'(t) & \\ \qquad \text{if } t \in Dom \cap Dom' & \\ ref(t) \quad \text{if } t \in Dom - Dom' & \\ ref'(t) \quad \text{otherwise .} & \end{cases}$$

Proposition 5.1 *It $\mathfrak{ep}$ and $\mathfrak{ep}'$ are compatible, then $\mathfrak{ep} \sqcap \mathfrak{ep}'$ is a well-defined extraction pattern.* □

Compatible $\mathfrak{ep}$ and $\mathfrak{ep}'$ are *parallel–compatible* if, for all $t \in Dom \cap Dom'$ and $R \in ref(t)$, $R' \in ref'(t)$, we have $R \cup R' \neq Src$. In such a case, we define an extraction pattern $\mathfrak{ep} \| \mathfrak{ep}'$ as follows:

- $Src_{\mathfrak{ep}\|\mathfrak{ep}'} \stackrel{\text{df}}{=} Src = Src'$.
- $Trg_{\mathfrak{ep}\|\mathfrak{ep}'} \stackrel{\text{df}}{=} Trg = Trg'$.
- $dom_{\mathfrak{ep}\|\mathfrak{ep}'} \stackrel{\text{df}}{=} dom \cap dom'$
  (and so $Dom_{\mathfrak{ep}\|\mathfrak{ep}'} = Dom \cap Dom'$).
- For every trace $t \in Dom_{\mathfrak{ep}\|\mathfrak{ep}'}$,

$$\begin{aligned} extr_{\mathfrak{ep}\|\mathfrak{ep}'}(t) &\stackrel{\text{df}}{=} extr(t) = extr'(t) \ , \\ ref_{\mathfrak{ep}\|\mathfrak{ep}'}(t) &\stackrel{\text{df}}{=} \{R \cup R' \mid R \in ref(t{\restriction}Src) \wedge \\ &\qquad R' \in ref'(t{\restriction}Src')\} \ . \end{aligned}$$

Proposition 5.2 *If $\mathfrak{ep}$ and $\mathfrak{ep}'$ are parallel–compatible, then $\mathfrak{ep}\|\mathfrak{ep}'$ is a well-defined extraction pattern.* □

6. Compositionality results

We can now formulate compositionality properties for the extraction pattern based implementation relation. The first one concerns hiding of visible actions in the implementation and base processes.

Theorem 6.1 *Let $Q \preceq_{\mathfrak{Ep}} P$ and $\mathfrak{ep} \in \mathfrak{Ep}$ be an extraction pattern such that $P \backslash Trg_{\mathfrak{ep}}$ is a divergence-free process and $\tau_{\mathfrak{Ep}-\{\mathfrak{ep}\}} Q = \tau_{\mathfrak{Ep}} Q$. Then*

$$Q \backslash Src_{\mathfrak{ep}} \ \preceq_{\mathfrak{Ep}-\{\mathfrak{ep}\}} \ P \backslash Trg_{\mathfrak{ep}} \ .$$

Proof outline: In the first step, one shows that we have $\delta_{\mathfrak{Ep}-\{\mathfrak{ep}\}}(Q \backslash Src_{\mathfrak{ep}}) = \varnothing$, which can be proven from $\delta(P \backslash Trg_{\mathfrak{ep}}) = \varnothing$, $\tau_{\mathfrak{Ep}-\{\mathfrak{ep}\}} Q = \tau_{\mathfrak{Ep}} Q$ as well as definition 4.1(1,2,3) for Q and P. Furthermore, one can see that $\tau_{\mathfrak{Ep}-\{\mathfrak{ep}\}}(Q \backslash Src_{\mathfrak{ep}}) = (\tau_{\mathfrak{Ep}} Q){\restriction}(\alpha Q - Src_{\mathfrak{ep}})$. Then definition 4.1(1,2,3) for $Q \backslash Src_{\mathfrak{ep}}$ and $P \backslash Trg_{\mathfrak{ep}}$ can be established.

In the second step, one shows that if $(t, R) \in \phi_{\mathfrak{Ep}-\{\mathfrak{ep}\}}(Q \backslash Src_{\mathfrak{ep}})$ then there is $(t', R') \in \phi_{\mathfrak{Ep}} Q$ such that $t = t'{\restriction}(\alpha Q - Src_{\mathfrak{ep}})$ and $R' = R \cup Src_{\mathfrak{ep}}$ and $t'{\restriction}Src_{\mathfrak{ep}} \in dom_{\mathfrak{ep}}$, which follows from what has been shown in the first step as well as definition 4.1(4) for Q and P (note that $Src_{\mathfrak{ep}} \notin ref_{\mathfrak{ep}}(t'{\restriction}Src_{\mathfrak{ep}})$ due to the definition of an extraction pattern). Then definition 4.1(4,5) for (t, R) can be established using the corresponding properties of (t', R'). □

There are two conditions in the above theorem. The first one, that $P \backslash Trg_{\mathfrak{ep}}$ is a divergence-free process, is quite natural to have, as $P \backslash Trg_{\mathfrak{ep}}$ plays the role of a specification process. The second one, $\tau_{\mathfrak{Ep}-\{\mathfrak{ep}\}} Q = \tau_{\mathfrak{Ep}} Q$, is directly

related to the way the implementation relation has been defined. Basically, it states that there can be no trace in Q which can be interpreted after appropriate projection by extraction patterns other than $\mathfrak{ep}$, but not by $\mathfrak{ep}$ itself. Notice that no such trace would be considered by the various parts of definition 4.1 for Q and P, yet after hiding the actions in $Src_{\mathfrak{ep}}$, the trace should be considered by definition 4.1 applied to $Q\backslash Src_{\mathfrak{ep}}$ and $P\backslash Trg_{\mathfrak{ep}}$.

Repeated application of the above result gives:

Corollary 6.2 *Let $Q \preceq_{\mathfrak{Ep}} P$ be such that $P\backslash Trg_{\mathfrak{Ep}}$ is a divergence-free process and $\tau Q = \tau_{\mathfrak{Ep}} Q$. Then*

$$Q\backslash Src_{\mathfrak{Ep}} \preceq P\backslash Trg_{\mathfrak{Ep}} .$$

Note that the condition $\tau Q = \tau_{\mathfrak{Ep}} Q$ above can be checked using the standard *trace refinement* of CSP [6, 13].

The second compositionality result concerns parallel composition.

Theorem 6.3 *Let Q, Q', P and P' be processes such that*

$$Q \preceq_{\mathfrak{Ep}\uplus\{\mathfrak{ep}_1,\ldots,\mathfrak{ep}_n\}} P \text{ and } Q' \preceq_{\mathfrak{Ep}'\uplus\{\mathfrak{ep}'_1,\ldots,\mathfrak{ep}'_n\}} P' ,$$

where:

$$\begin{array}{rcccl} Src_{\mathfrak{Ep}} \cap \alpha Q' & = & Src_{\mathfrak{Ep}'} \cap \alpha Q & = & \varnothing \\ Trg_{\mathfrak{Ep}} \cap \alpha P' & = & Trg_{\mathfrak{Ep}'} \cap \alpha P & = & \varnothing \end{array}$$

and for each $i \leq n$, $\mathfrak{ep}_i$ and $\mathfrak{ep}'_i$ are parallel–compatible extraction patterns. Then

$$Q\|Q' \preceq_{\mathfrak{Ep}\uplus\mathfrak{Ep}'\uplus\{\mathfrak{ep}_1\|\mathfrak{ep}'_1,\ldots,\mathfrak{ep}_n\|\mathfrak{ep}'_n\}} P\|P' .$$

Proof outline: Let us denote:

$$\begin{array}{rcl} \mathfrak{Ep}_0 & \stackrel{\text{df}}{=} & \mathfrak{Ep} \uplus \{\mathfrak{ep}_1, \ldots, \mathfrak{ep}_n\} \\ \mathfrak{Ep}_1 & \stackrel{\text{df}}{=} & \mathfrak{Ep}' \uplus \{\mathfrak{ep}'_1, \ldots, \mathfrak{ep}'_n\} \\ \mathfrak{Ep}_2 & \stackrel{\text{df}}{=} & \mathfrak{Ep} \uplus \mathfrak{Ep}' \uplus \{\mathfrak{ep}_1\|\mathfrak{ep}'_1, \ldots, \mathfrak{ep}_n\|\mathfrak{ep}'_n\} . \end{array}$$

In the first step, one shows that $\delta_{\mathfrak{Ep}_2}(Q\|Q') = \varnothing$, which can be proven from $\delta_{\mathfrak{Ep}_0} Q = \delta_{\mathfrak{Ep}_1} Q' = \varnothing$. Furthermore, one can see that $\tau_{\mathfrak{Ep}_2}(Q\|Q')$ is the set of all traces t over $\alpha Q \cup \alpha Q'$ such that $t{\restriction}\alpha Q \in \tau_{\mathfrak{Ep}_0} Q$ and $t{\restriction}\alpha Q' \in \tau_{\mathfrak{Ep}_1} Q'$. Then definition 4.1(1,2,3) for $Q\|Q'$ and $P\|P'$ can be established.

In the second step, one shows that if $(t, R) \in \phi_{\mathfrak{Ep}_2}(Q\|Q')$ then there are failures $(t', R') \in \phi_{\mathfrak{Ep}_0} Q$ and $(t'', R'') \in \phi_{\mathfrak{Ep}_1} Q'$ such that $t' = t{\restriction}\alpha Q$ and $t'' = t{\restriction}\alpha Q'$ and $R = R' \cup R''$. Moreover, if extraction pattern $\mathfrak{ep} \in \mathfrak{Ep}_2$ is such that $R \cap Src_{\mathfrak{ep}} \notin ref_{\mathfrak{ep}}(t{\restriction}Src_{\mathfrak{ep}})$, then $t{\restriction}Src_{\mathfrak{ep}} \in dom_{\mathfrak{ep}}$ (we only note that in the case $\mathfrak{ep} = \mathfrak{ep}_i\|\mathfrak{ep}'_i$ we have that $R \cap Src_{\mathfrak{ep}_i\|\mathfrak{ep}'_i} \notin ref_{\mathfrak{ep}_i\|\mathfrak{ep}'_i}(t{\restriction}Src_{\mathfrak{ep}_i\|\mathfrak{ep}'_i})$ implies, due to the parallel-compatibility, that $R \cap Src_{\mathfrak{ep}_i} \notin ref_{\mathfrak{ep}_i}(t{\restriction}Src_{\mathfrak{ep}_i})$ or $R \cap Src_{\mathfrak{ep}'_i} \notin ref_{\mathfrak{ep}'_i}(t{\restriction}Src_{\mathfrak{ep}'_i})$, and so we have that $t{\restriction}Src_{\mathfrak{ep}_i} \in dom_{\mathfrak{ep}_i}$ or $t{\restriction}Src_{\mathfrak{ep}'_i} \in dom_{\mathfrak{ep}'_i}$). Then definition 4.1(4,5) for (t, R) can be established using the corresponding properties of (t', R') and (t'', R''). □

Hence the implementation relation is preserved through parallel composition, under the proviso that the interface between two implementation processes is governed by parallel–compatible extraction patterns. Together with theorem 6.1, this means that the proposed scheme can be used to deal with distributed networks of CSP processes.

The final compositionality property concerns nondeterministic choice. It is included here mainly to illustrate a point that compositionality of implementations can be extended to other operators of CSP.

Theorem 6.4 *Let Q, Q', P and P' be processes such that*

$$Q \preceq_{\{\mathfrak{ep}_1,\ldots,\mathfrak{ep}_n\}} P \text{ and } Q' \preceq_{\{\mathfrak{ep}'_1,\ldots,\mathfrak{ep}'_n\}} P' ,$$

where: $\alpha Q = \alpha Q'$, $\alpha P = \alpha P'$ and for each $i \leq n$, $\mathfrak{ep}_i$ and $\mathfrak{ep}'_i$ are compatible extraction patterns. Then

$$Q \sqcap Q' \preceq_{\{\mathfrak{ep}_1\sqcap\mathfrak{ep}'_1,\ldots,\mathfrak{ep}_n\sqcap\mathfrak{ep}'_n\}} P \sqcap P' .$$

Proof outline: Let us denote:

$$\begin{array}{rcl} \mathfrak{Ep}_0 & \stackrel{\text{df}}{=} & \{\mathfrak{ep}_1, \ldots, \mathfrak{ep}_n\} \\ \mathfrak{Ep}_1 & \stackrel{\text{df}}{=} & \{\mathfrak{ep}'_1, \ldots, \mathfrak{ep}'_n\} \\ \mathfrak{Ep}_2 & \stackrel{\text{df}}{=} & \{\mathfrak{ep}_1 \sqcap \mathfrak{ep}'_1, \ldots, \mathfrak{ep}_n \sqcap \mathfrak{ep}'_n\} . \end{array}$$

In the first step, one shows that $\delta_{\mathfrak{Ep}_2}(Q\sqcap Q') = \varnothing$, which can be proven from $\delta_{\mathfrak{Ep}_0} Q = \delta_{\mathfrak{Ep}_1} Q' = \varnothing$. Furthermore, one can see that $\tau_{\mathfrak{Ep}_2}(Q \sqcap Q') = \tau_{\mathfrak{Ep}_0} Q \cup \tau_{\mathfrak{Ep}_1} Q'$. Then definition 4.1(1,2,3) for $Q \sqcap Q'$ and $P \sqcap P'$ can be established.

In the second step, one shows that if $(t, R) \in \phi_{\mathfrak{Ep}_2}(Q \sqcap Q')$ then, without loss of generality, we have $(t, R) \in \phi_{\mathfrak{Ep}_0} Q$, and if $R \cap Src_{\mathfrak{ep}_i\sqcap\mathfrak{ep}'_i} \notin ref_{\mathfrak{ep}_i\sqcap\mathfrak{ep}'_i}(t{\restriction}Src_{\mathfrak{ep}_i\sqcap\mathfrak{ep}'_i})$, then $t{\restriction}Src_{\mathfrak{ep}_i\sqcap\mathfrak{ep}'_i} = t{\restriction}Src_{\mathfrak{ep}_i} \in dom_{\mathfrak{ep}_i} \subseteq dom_{\mathfrak{ep}_i\sqcap\mathfrak{ep}'_i}$. Then definition 4.1(4,5) for (t, R) can be established. □

7. Concluding remarks

This paper pursues further the approach we proposed in [4, 3] for the formalisation of implementation in the presence of interface difference. For the new scheme, the properties referred to as compositionality and realisability in the prior works still hold. Some fundamental extensions have been obtained.

To begin with, no restrictions are now placed on the class of specification processes allowed, other than a quite natural divergence-freedom requirement. Realisability holds in its purest form, in the sense that implementation in the absence of interface difference collapses into standard CSP refinement. Moreover, we can now deal with process networks where the one-to-one communication constraint need

not be adhered to; this allows group communication to be modelled.

Last but not least, we generalise compositionality results, through a treatment intended to show that the implementation relation distributes over the main CSP operators, beginning from non-deterministic choice. Such results are apt to prove beneficial in the compositional verification of systems. It is worth noting, in particular, that more general scenarios can now be handled than process network topologies as in Figure 1, where implementation and base processes are paired in a unique way. For example, in a situation as in Figure 5 we can apply Theorems 6.3 and 6.1 to prove that Q_{net} implements P_{net} in a stepwise fashion, by showing that: (i) Q_1 implements P_1, and (ii) the parallel composition of Q_2 and Q_3, with their mutual interaction hidden, implements P.

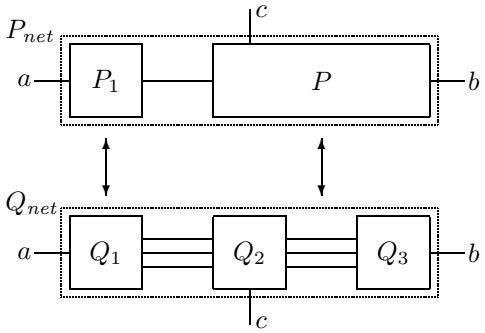


Figure 5. Two networks with different topology.

This novel approach is based on introducing operations over extraction patterns, mimicking (and being compatible with) the corresponding operations over processes. We feel this constitutes a first step in the development of an algebra of abstraction for communicating processes in the CSP model.

In future work, we plan to extend the algebra of extraction patterns to deal with other commonly used CSP operators, and to develop model checking techniques for the proposed implementation relation, based on the ideas contained in [3].

A general comparison of the approach presented here with those of [1, 2, 7, 12, 14] can be found in [3].

References

[1] M. Abadi and L. Lamport: The Existence of Refinement Mappings. *Theoretical Computer Science* 82 (1991) 253-284.

[2] E. Brinksma, B. Jonsson, and F. Orava: Refining Interfaces of Communicating Systems. Proc. of *Coll. on Combining Paradigms for Software Development*, Springer-Verlag, Lecture Notes in Computer Science 494 (1991).

[3] J. Burton, M. Koutny and G. Pappalardo: Implementing Communicating Processes in the Event of Interface Difference. *Fundamenta Informaticae* 59 (2004) 1-37.

[4] J. Burton, M. Koutny, G. Pappalardo and M. Pietkiewicz-Koutny: Compositional Development in the Event of Interface Difference. In: *Concurrency in Dependable Computing*, P. Ezhilchelvan and A. Romanovsky (Eds.). Kluwer Academic Publishers (2002) 3-22.

[5] R. Gerth, R. Kuiper and J. Segers: Interface Refinement in Reactive Systems. Proc. of *CONCUR '92*, Springer-Verlag, Lecture Notes in Computer Science 630 (1992) 77-93.

[6] C. A. R. Hoare: *Communicating Sequential Processes*. Prentice Hall (1985).

[7] B. Jonsson: Compositional Specification and Verification of Distributed Systems. *ACM TOPLAS* 16 (1994) 259-303.

[8] M. Koutny, L. Mancini and G. Pappalardo: Two Implementation Relations and the Correctness of Communicated Replicated Processing. *Formal Aspects of Computing* 9 (1997) 119-148.

[9] L. Lamport: The Implementation of Reliable Distributed Multiprocess Systems. *Computer Networks* 2 (1978) 95-114.

[10] L. V. Mancini, G. Pappalardo: Towards a Theory of Replicated Processing. Proc. of *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Springer-Verlag, Lecture Notes in Computer Science 331 (1988) 175–192.

[11] R. Milner: *Communication and Concurrency*. Prentice Hall (1989).

[12] A. Rensink and R. Gorrieri: Vertical Implementation. *Information and Computation* 170 (2001) 95–133.

[13] A. W. Roscoe: *The Theory and Practice of Concurrency*. Prentice-Hall (1998).

[14] H. Schepers and J. Hooman: Trace-based Compositional Reasoning About Fault-tolerant Systems. Proc. of *PARLE'93*, Springer-Verlag, Lecture Notes in Computer Science 694 (1993).