Relating Communicating Processes with Different Interfaces

Jonathan Burton[1], Maciej Koutny[1], and Giuseppe Pappalardo[2]

[1] Department of Computing Science, University of Newcastle
Newcastle upon Tyne NE1 7RU, U.K.
{j.i.burton,Maciej.Koutny}@ncl.ac.uk
[2] Dipartimento di Matematica e Informatica, Università di Catania
I-95125 Catania, Italy
pappalardo@dmi.unict.it

Abstract. We present here an implementation relation intended to formalise the notion that a system built of communicating processes is an acceptable implementation of another base, or target, system in the event that the two systems have different interfaces. Such a treatment has clear applicability in the software development process, where (the interface of) an implementation component may be expressed at a different level of abstraction to (the interface of) the relevant specification component. We extend the results of our previous work and replace implementation relations previously presented by a single, improved scheme. We also remove all the restrictions previously placed upon target processes. Two basic kinds of results are obtained: realisability and compositionality. The latter means that a target composed of several connected systems may be implemented by connecting their respective implementations. The former means that, if target and implementation have the same interface, then the implementation relation they should satisfy collapses into the standard implementation pre-order.
We also show how to represent processes and necessary formal structures in a manner amenable to computer implementation, and detail a graph-theoretic restatement of the conditions defining the implementation relation, whence we derive algorithms for its automatic verification.
Keywords: Theory of parallel and distributed computation, behaviour abstraction, refinement, communicating sequential processes, compositionality, verification.

1 Introduction

The software development process often involves refining a high-level *specification* into a lower-level or more concrete *implementation*.

In the process algebraic context [8, 16, 18], both specification and implementation may be represented as processes, and the notion that a process Q *implements* a process P is based on the idea that Q is more deterministic than (or equivalent to) P in terms of the chosen semantics. In the following, we shall also refer to such specifications as *target* or *base* systems.

The process of refining the target into the implementation also permits the control structure of the latter to be changed. In such a case, Q is said to implement P in the twofold sense that: (i) Q describes the *internal* structure of P in a more concrete and detailed manner; and still (ii), if this new structure is (conceptually) hidden, Q and P will exhibit the same behaviour at their external interface, which is assumed to be the same for both. Indeed, the standard notions of refinement, such as those of [8, 16, 18], are interested only in the behaviour observable at the *interface* of processes, and require the interfaces of the specification and implementation to be the same, so as to facilitate comparison.

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 channel connecting Q to another component process may be unreliable, and so may need to be replaced by a pair of channels, one for data and one for acknowledgments. Or Q itself may be liable to fail, so that its behaviour may have to be replicated, with each new component having its own communication channels to avoid a single channel becoming a bottleneck [13] (such a scenario was one of the major historical motivations behind the current work [11, 15]). Or it may simply be the case that a high-level action of P 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 to a specification process.

In the process algebraic context, dealing with this phenomenon of *interface difference* necessitates the development of what Rensink and Gorrieri [17] have termed a *vertical implementation* relation. This should correctly capture the nature of the relationship holding between a specification and an implementation whose interfaces differ; and should collapse into the standard, *horizontal* one whenever the interfaces happen to coincide.

Within the FD (failure-divergence) model of CSP [9], we have pioneered such an approach in works like [11, 12, 15]), on whose results the present one introduces major advances, as argued in the Conclusions. Our treatment deals with interface difference using the notion of *extraction pattern*. Such a device interprets the behaviour of a system at the level of communication traces, by relating behaviour on a set of channels in the implementation to behaviour on a specific channel in the specification. In addition, it allows the behaviour of an implementation to be suitably constrained, in connection to, e.g., well-formedness of input traces and deadlock properties. The set of all extraction patterns relating the interface of the implementation process to that of the specification appears as a formal parameter in the *implementation relation* we develop.

We now consider the potential applications of the approach outlined to verification, in order to deduce two light but natural restrictions which must be placed upon any sensible vertical implementation relation, and are indeed met by that presented in this work. Suppose the specification system is in the form $P \stackrel{\text{df}}{=} (P_1 \parallel P_2 \parallel \ldots \parallel P_n) \setminus A$, where A is the set of events on which synchronization among the P_i components takes place. Correspondingly, let the implementation system be $Q \stackrel{\text{df}}{=} (Q_1 \parallel Q_2 \parallel \ldots \parallel Q_n) \setminus B$. In general, the communication

interface need not be the same for each P_i and Q_i, but will be assumed to coincide for Q and P. As a result, it would be possible to verify directly, using a standard, horizontal relation, whether or not Q implements P.

However, we wish to verify that Q implements P using a compositional approach: i.e., by verifying that Q_i implements P_i for $1 \leq i \leq n$. There are two, related, motivations for such a choice. To begin with, it gives us a means to tackle the state explosion problem. Furthermore, it allows us to verify the correctness of individual components in isolation, without needing to know in advance the structure of the network in which they will be deployed. Hence the first requirement for our implementation relation, i.e., that it should be *compositional* in the said sense.

To introduce the second requirement, we remark that, once each Q_i component has been established to implement the respective P_i, the composition Q is known, by compositionality, to implement P according to the implementation relation introduced. For the new relation to be meaningful, Q should also implement P in a standard horizontal sense (recall that the interfaces of Q and P are the same). In other words, we require that the implementation relation 'collapse' to a standard horizontal implementation relation, in the event that the specification and implementation processes have the same interface (in our approach, this means that all communication channels of the implementation are 'uninterpreted'). We call this property *accessibility* or realisability.

In the following, in addition to presenting an implementation relation which meets the criteria set out above, we also detail a graph-theoretic restatement of the implementation relation conditions, from which we derive algorithms for their automatic verification.

The paper is organised as follows. In the next section we introduce some basic notions used throughout the paper. In section 3 we first introduce extraction patterns — a central notion to defining the interface of an implementation. Section 4 presents the implementation relation. Section 5 details how we may represent processes and extraction patterns in a manner amenable to computer implementation, and develops results allowing automatic verification of the implementation relation. Section 6 summarises the results and compares them with other related works. All the proofs are included in the appendix.

2 Preliminaries

Processes are represented in this paper using the failures-divergences model of Communicating Sequential Processes (CSP) [9, 18] — 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 be elements of 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. In this paper, structured actions of the form $b.v$ will be used, where v is a *message* and b is a communication *channel*. For every channel b, μb is the *message set* of b —

the set of all v such that $b.v$ is a valid action. We define $\alpha b = \{b.v \mid v \in \mu b\}$ to be the *alphabet* of channel b. It is assumed that μb is always finite and non-empty.

The following notations are similar to that of [9] (below $t, u, t_1, t_2, \ldots$ are traces; b is a channel; $B_1, \ldots, B_n$, B are disjoint sets of channels; 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, $|t|$, 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, $\langle\,\rangle$.
- T^* is the set of all traces $t = t_1 \circ \cdots \circ t_n$ $(n \geq 0)$ such that $t_1, \ldots, t_n \in T$.
- $\leq$ denotes the prefix relation on traces, and $t < u$ if $t \leq u$ and $t \neq u$.
- $\mathit{Pref}(T) \stackrel{\text{df}}{=} \{u \mid \exists t \in T :\ u \leq t\}$ is the *prefix-closure* of T.
- T is *prefix-closed* if $T = \mathit{Pref}(T)$.
- $t{\restriction}B$ is a trace obtained by deleting from t all the actions that do not occur on the channels in B.
- $t_1, t_2, \ldots$ 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 $f(t) \leq f(u)$, and *strict* if $\langle\,\rangle \in T$ and $f(\langle\,\rangle) = \langle\,\rangle$.

We use the model of CSP in which a process P is a triple $(\alpha P, \phi P, \delta P)$ where αP — *alphabet* — is a non-empty finite set of actions, ϕP — *failures* — is a subset of $\alpha P^* \times \mathbb{P}(\alpha P)$, and δP — *divergences* — is a subset of αP^*. Moreover, $\tau P \stackrel{\text{df}}{=} \{t \mid (t, R) \in \phi P\}$ denotes the *traces* of P. The conditions imposed on the components of a CSP process are given below.

CSP1 τP is a non-empty and prefix-closed set.
CSP2 If $(t, R) \in \phi P$ and $S \subseteq R$ then $(t, S) \in \phi P$.
CSP3 If $(t, R) \in \phi P$ and $a \in \alpha P$ satisfy $t \circ \langle a \rangle \notin \tau P$ then $(t, R \cup \{a\}) \in \phi P$.
CSP4 If $t \in \delta P$ then $(t \circ u, R) \in \phi P$, for all $u \in \alpha P^*$ and all $R \subseteq \alpha P$.

We will associate with P a set of channels, χP, and stipulate that the alphabet of P is that of χP. Thus, we shall be able to identify P with the triple $(\chi P, \phi P, \delta P)$ in lieu of $(\alpha P, \phi P, \delta P)$.

A fundamental device to compare CSP processes in the failures-divergences semantical model is the *refinement order*, $\sqsupseteq$, defined so that $Q \sqsupseteq P$ if $\phi Q \subseteq \phi P$ and $\delta Q \subseteq \delta P$. Intuitively, this means that Q is at least as good, and perhaps better, than P for actual deployment in any conceivable environment.

We assume that *base* processes are *non-diverging* CSP processes. This is the only restriction placed upon base processes to be treated using the implementation relation presented in section 4.

CSP operators For our purposes neither the syntax nor the semantics of the whole standard CSP is needed. Essential are only the parallel composition of processes and hiding of the communication over a set of channels. In the examples we also use deterministic choice, $P[]Q$, non-deterministic choice $P \sqcap Q$, renaming of channels $P[b'/b]$, and prefixing, $a \to P$ (see [9, 18] and also appendix A).

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 all actions that are in the intersection of their alphabet. Formally, $\chi(P\|Q) \stackrel{\text{df}}{=} \chi P \cup \chi Q$ and

$$\begin{aligned}\delta(P\|Q) &\stackrel{\text{df}}{=} \{t \circ u \mid (t\restriction\chi P, t\restriction\chi Q) \in (\tau P \times \delta Q) \cup (\delta P \times \tau Q)\}\\ \phi(P\|Q) &\stackrel{\text{df}}{=} \{(t, R \cup S) \mid (t\restriction\chi P, R) \in \phi P \wedge (t\restriction\chi Q, S) \in \phi Q\} \ \cup\\ &\qquad \delta(P\|Q) \times \mathbb{P}(\alpha(P\|Q))\ .\end{aligned}$$

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

Let P be a process and B be a set of channels of P; then $P\backslash B$ is a process that behaves like P with the actions occurring at the channels in B made internal. Formally, $\chi(P\backslash B) \stackrel{\text{df}}{=} \chi P - B$ and

$$\begin{aligned}\delta(P\backslash B) &\stackrel{\text{df}}{=} \{t\restriction\chi(P\backslash B) \circ u \ \mid t \in \delta P \vee\\ &\qquad\qquad \exists a_1, a_2, \ldots \in \alpha B\ \forall n \geq 1 : t \circ \langle a_1, \ldots, a_n\rangle \in \tau P\}\\ \phi(P\backslash B) &\stackrel{\text{df}}{=} \{(t\restriction\chi(P\backslash B), R) \mid (t, R \cup \alpha B) \in \phi P\} \cup \delta(P\backslash B) \times \mathbb{P}(\alpha(P\backslash B))\ .\end{aligned}$$

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

Networks of processes Processes $P_1, \ldots, P_n$ form a *network* if no channel is shared by more than two P_i's. We define $P_1 \otimes \cdots \otimes P_n$ to be the process obtained by taking the parallel composition of the processes and then hiding all interprocess communication, i.e., the process $(P_1\| \cdots \|P_n)\backslash B$, where B is the set of channels shared by at least two different processes P_i. Network composition is commutative and associative. As a result, a network can be obtained by first composing some of the processes into a subnetwork, and then composing the result with the remaining processes. Moreover, the order in which processes are composed does not matter. In the failure model of CSP, where a process P is identified with the pair $(\alpha P, \phi P)$, the former property does not hold, whence the need for the more complicated divergence model.

We can partition the channels of a process P into the input channels, $in\, P$, and output channels, $out\, P$. It is assumed that no two processes in a network have a common input channel or a common output channel and that being an input or output channel is preserved by network composition.

In the diagrams representing processes, outgoing arrows indicate output, and incoming arrows indicate input channels.[1]

3 Extraction patterns

In this section, we first explain the basic mechanism behind our modelling of behaviour abstraction, and then provide a formal definition of extraction patterns. Our examples here will deliberately be *very simple*, in order to better convey the basic ideas rather than to demonstrate a wider applicability of our approach.

[1] It should be noted that being an input or output channel of a process is, in general, a purely syntactic notion, and no semantic properties can therefore be inferred.

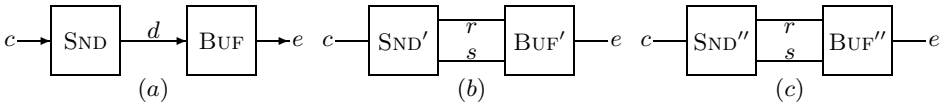


Fig. 1. Two base processes and their implementations.

Consider a pair of base processes, SND and BUF, shown in figure 1(a). SND generates an infinite sequence of 0s or an infinite sequence of 1s, depending on the signal (0 or 1) received on its input channel, c, at the very beginning of its execution. BUF is a buffer process of capacity one, forwarding signals received on its input channel, d. In terms of CSP, we have:

$$\mathrm{SND} \stackrel{\mathrm{df}}{=} (c.0 \rightarrow \mathrm{SND}_0) \,[]\, (c.1 \rightarrow \mathrm{SND}_1)$$
$$\mathrm{BUF} \stackrel{\mathrm{df}}{=} (d.1 \rightarrow e.0 \rightarrow \mathrm{BUF}) \,[]\, (d.1 \rightarrow e.1 \rightarrow \mathrm{BUF})$$

where $\mathrm{SND}_i \stackrel{\mathrm{df}}{=} d.i \rightarrow \mathrm{SND}_i$, for $i = 0, 1$.

Suppose that the signal transmission between the two processes has been implemented using two channels, r and s, as shown in figure 1(b). The transmissions on d are now duplicated and the two copies sent along r and s. That is, SND′ sends the duplicated signal, while BUF′ accepts a single copy of the signal and passes it on ignoring the other one. Such a simple scheme clearly works as we have $\mathrm{SND} \otimes \mathrm{BUF} = \mathrm{SND}' \otimes \mathrm{BUF}'$. Suppose now that the transmission of signals is imperfect and two types of faulty behaviour can occur: $\mathrm{SND}_1 \stackrel{\mathrm{df}}{=} \mathrm{SND}' \sqcap \mathrm{STOP}$ and $\mathrm{SND}_2 \stackrel{\mathrm{df}}{=} \mathrm{SND}' \sqcap \overline{\mathrm{SND}}$, where $\overline{\mathrm{SND}}$ is SND′ with all the communication on channel s being blocked. In other words, SND_1 can break down completely, refusing to output any signals, while SND_2 can fail in such a way that although channel s is dead r can still transmit the signals.[2] Since $\mathrm{SND} \otimes \mathrm{BUF} = \mathrm{SND}_2 \otimes \mathrm{BUF}'$ and $\mathrm{SND} \otimes \mathrm{BUF} \neq \mathrm{SND}_1 \otimes \mathrm{BUF}'$ it follows that SND_2 is much 'better' an implementation of the SND process than SND_1. We will now analyse the differences between the behavioural properties of the two processes and at the same time introduce informally some basic concepts used subsequently.

We start by observing that the output of SND_2 can be thought of as adhering to the following two rules:

R1 The transmissions over r and s are consistent w.r.t. message content (the set of all traces over r and s satisfying such a property will be denoted by Dom).

R2 Transmission over r is reliable, but there is no such guarantee for s.

The output produced by SND_1 satisfies the first rule, but fails to satisfy the second one as its behaviour allows both channels to be blocked. To express this

[2] SND_2 could be used to model the following situation: in order to improve performance, a 'slow' channel d is replaced by two channels, a high-speed yet unreliable channel s and a slow but reliable backup channel r.

difference formally we need to render these two conditions in some form of precise notation.

To capture the relationship between traces of SND and SND_2 we will employ an (extraction) mapping *extr* which for a trace over r and s returns the corresponding trace over d. For example, keeping in mind that duplicates of signals should be ignored by the receiving process, we have

$$\begin{aligned} \langle\,\rangle &\mapsto \langle\,\rangle \\ \langle r.0\rangle &\mapsto \langle d.1\rangle \\ \langle s.0\rangle &\mapsto \langle d.1\rangle \\ \langle s.1, r.1\rangle &\mapsto \langle d.1\rangle \\ \langle s.1, r.1, s.0\rangle &\mapsto \langle d.1, d.1\rangle \end{aligned}$$

Notice that the extraction mapping need only be defined for traces satisfying R1, i.e., those in *Dom*. We further observe that, in view of R2, some of the traces in *Dom* may be regarded as *incomplete*. For example, $\langle s.1, r.1, s.0\rangle$ is such a trace since channel r is reliable and so the duplicate of $s.0$ (i.e., $r.0$) is bound to be eventually be offered for transmission. The set of all other traces in *Dom* — i.e., those which in principle may be *complete* — will be denoted by *dom*.[3] For our example, *dom* will contain all traces in *Dom* where the transmission on s has not overtaken that on r.[4]

Although it will play a central role, the extraction mapping alone is not sufficient to identify the 'correct' implementation of SND in the presence of faults since $\tau\text{SND} = \mathit{extr}(\tau\text{SND}_1) = \mathit{extr}(\tau\text{SND}_2)$. What one also needs is an ability to relate the refusals of SND_1 and SND_2 with the possible refusals of the base process SND. This, however, is much harder than relating traces. For suppose that we attempted to 'extract' the refusals of SND_2 using *extr*. Then, we would have had

$$(\langle\,\rangle, \{s.0\}) \in \phi\text{SND}_2 \quad \text{and} \quad \mathit{extr}(\langle\,\rangle, \{s.0\}) = (\langle\,\rangle, \{d.1\}) \notin \phi\text{SND}.$$

This indicates that the crude extraction of refusals is not going to work. What we need is a more sophisticated device, which in our case comes in the form of another mapping, *ref*, constraining the possible refusals a process can exhibit, on channels which will be hidden in the composed system Q, after a given trace $t \in \mathit{Dom}$. More precisely, a sender process can admit a refusal disallowed by $\mathit{ref}(t)$ if the extracted trace $\mathit{extr}(t)$ admits in the target process the refusal of all communication on the corresponding channel and, moreover, the trace t itself is complete, i.e., $t \in \mathit{dom}$. For the example at hand, this roughly amounts to stipulating that an unfinished communication cannot at the same time refuse both $r.0$ and $r.1$.

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

[4] Another example is that if the whole sequence of events $a_1, \ldots, a_k$ is extracted to a single event 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.

Finally, it should be stressed that $ref(t)$ gives a refusal bound on the sender side (more precisely, the process which implements the sender target process). But this is enough since if we want to rule out a deadlock in communication between the sender and receiver (on a localised set of channels), it is now possible to stipulate on the receiver side that no refusal is such that, when combined with any refusal allowed by $ref(t)$, it can yield the whole alphabet of the channels used for transmission.

Networks of processes. Further clarity on the intuition behind the detail of the implementation relation presented in section 4 and the detailed description of extraction patterns presented below with respect to the notion of constraining refusal bounds can be gleaned from considering the specification and implementation *systems* discussed in section 1. There we considered a specification system $P = (P_1 \parallel P_2 \parallel \ldots \parallel P_n) \setminus A$ and an implementation system $Q = (Q_1 \parallel Q_2 \parallel \ldots \parallel Q_n) \setminus B$. We wish to ensure that $(t, R) \in \phi Q$ implies that $(t, R) \in \phi P$. Relating traces is technically not too difficult using the extraction mapping introduced above. However, to ensure that the above implication holds is more difficult.

In general, we must find those component failures from each of the Q_i which may contribute to a failure of the composed system Q. These components will contribute to a failure of Q if the union of their refusal sets contains all actions which are hidden during the composition to get Q. It is obviously difficult to check local failures for this property and it is this problem that the use of refusal bounds deals with. In general, if two component processes keep their refusals within the bounds specified, this will ensure that their composition will always be able to execute at least one action on the channels on which they were composed and so each state reached will always be unstable due to the fact that an internal action is enabled and so none of the local failures will contribute to a failure in the composed process.

If a local failure may contribute to a global failure of Q, then that local failure must correspond in some way to a local failure in the corresponding specification component. In general, the approach taken is to say that comparison will only take place when behaviour in the implementation component is 'complete' in some sense. When behaviour is not complete, then the refusal bounds may not be breached, meaning that local failures where behaviour is not complete cannot play a role in forming a failure of the system Q. This is sensible anyway, since the fact that behaviour is incomplete implies that implementation of a high-level action has been begun but has not been completed and we obviously do not wish to allow our system to deadlock in the middle of implementing an action that is atomic at the high-level.

In general, the refusal bounds may be thought of as ensuring a kind of liveness or progress on sets of channels upon which composition will occur when implementation components are composed to get the full system Q. Since these channels are to be composed upon and so hidden, the progress enforced manifests itself in the final system as the occurrence of an internal transition, which leads to instability of the states in which those internal transitions are enabled.

This then means that these states will not contribute a failure of Q. If we may not be able to force progress after a complete behaviour, then we ensure that progress will not be possible on the corresponding channel in the specification component. Here, lack of progress on internal channels leads to stability and the fact that the relevant state *will* give rise to a failure of P.

3.1 Another example

The previous example can be thought of as modelling a fail-stop communication between two processes (s being a fail-stop channel). The next example is different in that it employs a fault tolerant mechanism based on message retransmission. It is used to illustrate the point that implementations are not forced to preserve the *intuitive* direction of the transfer of messages.

As before, suppose now that the communication on d has been implemented using two channels, r and s, but now r is a data channel, and s is a feedback channel used to pass acknowledgments. It is, moreover, assumed that a given message is sent at most *twice* since a re-transmission always succeeds. This leads to a simple protocol which can be incorporated into suitably modified original processes. The resulting implementation processes shown in figure 1(c), $\textsc{Snd}''$ and $\textsc{Buf}''$, are given by:

$$\textsc{Snd}'' \stackrel{\text{df}}{=} \Box_{i\in\{0,1\}}\, c.i \rightarrow \textsc{Snd}''_i$$

$$\textsc{Buf}'' \stackrel{\text{df}}{=} \Box_{i\in\{0,1\}}\, r.i \rightarrow (s.ack \rightarrow \textsc{buf}'_i \sqcap s.nak \rightarrow \textsc{buf})$$

where $\textsc{buf}$, $\textsc{Snd}''_i$ and buf'_i $(i = 0, 1)$ are auxiliary processes defined thus:

$$\textsc{Snd}''_i \stackrel{\text{df}}{=} r.i \rightarrow (s.ack \rightarrow \textsc{Snd}''_i \,\Box\, s.nak \rightarrow r.i \rightarrow \textsc{Snd}''_i)$$

$$\textsc{buf} \stackrel{\text{df}}{=} \Box_{i\in\{0,1\}}\, r.i \rightarrow \textsc{buf}'_i$$

$$\textsc{buf}'_i \stackrel{\text{df}}{=} e.i \rightarrow \textsc{Buf}'' \,.$$

It may be observed that $\textsc{Snd}'' \otimes \textsc{Buf}'' = \textsc{Snd} \otimes \textsc{Buf} = \textsc{Snd}[e/d]$. One way of showing this would be to compose the two pairs of processes and prove their equality using, e.g., CSP laws [9]. This would be straightforward for $\textsc{Snd} \otimes \textsc{Buf}$, but less so for $\textsc{Snd}'' \otimes \textsc{Buf}''$, at least by hand. However, the results in this paper allow us to proceed compositionally: we show that $\textsc{Snd}''$ and $\textsc{Buf}''$ are implementations of the respective base processes according to suitable *extraction patterns*, and then derive the desired relationship using general results developed in section 4.

3.2 Formal definition

The notion of extraction pattern (introduced in [11, 12], and slightly modified as well as simplified here[5]) relates behaviour on a set of channels in an implementa-

[5] What we define here is a *basic* extraction pattern in the terminology of [11]; [12] also allowed sets of target channels. Moreover, one can also use a partial inverse of the extraction mapping.

tion process to that on a channel in a target process. It has two main functions: that of interpretation of behaviour necessitated by interface difference and the encoding of some correctness requirements.

An *extraction pattern* is a tuple $ep \stackrel{\text{df}}{=} (B, b, dom, extr, ref)$ satisfying the following:

EP1 B is a non-empty set of channels, called *sources* and b is a channel, called *target*.

EP2 dom is a non-empty set of traces over the sources; its prefix-closure is denoted by Dom.

EP3 $extr$ is a strict monotonic mapping defined for traces in Dom; for every t, $extr(t)$ is a trace over the target.

EP4 ref is a mapping defined for traces in Dom such that for every $t \in Dom$:
- $ref(t)$ is a non-empty subset-closed[6] family of proper subsets of αB
- if $a \in \alpha B$ and $t \circ \langle a \rangle \notin Dom$ then $R \cup \{a\} \in ref(t)$, for all $R \in ref(t)$.

As already mentioned, the mapping $extr$ interprets a trace over the source channels B (in the implementation process) in terms of a trace over a channel b (in the target process) and defines functionally correct (i.e., in terms of traces) behaviour over those source channels by way of its domain. 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 channels. dom contains those traces in Dom for which the communication over B may be regarded as complete, and processes may violate the constraint on refusals given by ref only for such traces.

The extraction mapping is monotonic as receiving more information cannot decrease the current knowledge about the transmission. $\alpha B \notin ref(t)$ will be useful in that for an unfinished communication t we do not allow the sender to refuse all possible transmission. The second condition in EP4 is a rendering in terms of extraction patterns of a condition imposed on CSP processes that impossible events can always be refused (see CSP3).

The various components of the extraction patterns can be annotated (e.g., subscripted) to avoid ambiguity and, unless stated otherwise, different extraction patterns will have disjoint sources and distinct targets. Moreover, we lift three of the notions introduced above to any set of extraction patterns $Ep \stackrel{\text{df}}{=} \{ep_1, \ldots, ep_n\}$, where $ep_i \stackrel{\text{df}}{=} (B_i, b_i, dom_i, extr_i, ref_i)$:

- Dom_{Ep} and dom_{Ep} are the set of all traces t over channels $B_1 \cup \ldots \cup B_n$ such that respectively $t{\restriction}B_i \in Dom_i$ and $t{\restriction}B_i \in dom_i$, for every $i \leq n$.
- $extr_{Ep}(\langle\,\rangle) \stackrel{\text{df}}{=} \langle\,\rangle$ and, for every $t \circ \langle a \rangle \in Dom_{Ep}$ with $a \in \alpha B_i$, $extr_{Ep}(t \circ \langle a \rangle) \stackrel{\text{df}}{=} extr_{Ep}(t) \circ u$, where (possibly empty) u is such that $extr_i(t{\restriction}B_i \circ \langle a \rangle) = extr_i(t{\restriction}B_i) \circ u$. Note that such a u is well defined since $extr_i$ is monotonic.

Finally, for any mapping ref and $t \in Dom$, we will denote by $\overline{ref}(t)$ the set of all $X \in \alpha B$ such that, for every $Y \in ref(t)$, $X \cup Y \neq \alpha B$.

[6] A family of sets $\mathcal{X}$ is *subset-closed* if $Y \subset X \in \mathcal{X}$ implies $Y \in \mathcal{X}$.

Uninterpreted channels and identity extraction patterns Not all channels will require an extraction pattern as such to interpret their behaviour. We shall call these channels *uninterpreted* (other channels being called *interpreted*). Intuitively, these channels have the same interface in both implementation and specification.

An uninterpreted channel c will be identified by a special 'degenerated' extraction pattern $ep_c \stackrel{\text{df}}{=} (\{c\}, c, \alpha c^*, id, \bot)$, where extraction mapping id is the identity on αc^*, and the *ref* component is left unspecified.

Another extraction pattern In order to demonstrate that Snd″ and Buf″ are implementations of respectively Snd and Buf, we will need an extraction pattern ep_{twice}. We also observe that channel c is an identity channel as described above.

For the ep_{twice} extraction pattern, $B \stackrel{\text{df}}{=} \{s, r\}$ are the source channels and $b \stackrel{\text{df}}{=} d$ is the target channel; moreover $\mu d = \mu r \stackrel{\text{df}}{=} \{0, 1\}$ and $\mu s \stackrel{\text{df}}{=} \{ack, nak\}$. The remaining components are defined as follows, where $t \in dom$ and $t \circ u \in Dom$:

$$dom \stackrel{\text{df}}{=} \{\langle r.0, s.ack\rangle, \langle r.0, s.nak, r.0\rangle, \langle r.1, s.ack\rangle, \langle r.1, s.nak, r.1\rangle\}^*$$

$$extr(t \circ u) \stackrel{\text{df}}{=} \begin{cases} \langle\,\rangle & \text{if } t \circ u = \langle\,\rangle \\ extr(t) \circ \langle d.v\rangle & \text{if } u = \langle r.v, s.ack\rangle \\ & \text{or } u = \langle r.v, s.nak, r.v\rangle \\ extr(t) & \text{if } u = \langle r.v\rangle \text{ or } u = \langle r.v, s.nak\rangle \end{cases}$$

$$ref(t \circ u) \stackrel{\text{df}}{=} \begin{cases} \mathbb{P}(\alpha r) & \text{if } u = \langle r.v\rangle \\ \{R \in \mathbb{P}(\alpha r \cup \alpha s) \mid \alpha r \not\subseteq R\} & \text{if } u = \langle\,\rangle \\ \{R \in \mathbb{P}(\alpha r \cup \alpha s) \mid r.v \notin R\} & \text{if } u = \langle r.v, s.nak\rangle\ . \end{cases}$$

Intuitively, we can extract $\langle d.1\rangle$ from the following two sequences of communications: $\langle r.0, s.ack\rangle$ and $\langle r.0, s.nak, r.0\rangle$ (and similarly for $\langle d.1\rangle$). Thus a valid trace in Dom is one which is a concatenation of a series of 'complete' segments of this kind, possibly followed by an initial fragment of one of them. Any trace for which the latter is true, is *incomplete* and belongs to $Dom - dom$; otherwise it belongs to dom.

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 two sets of extraction patterns, In and Out. The former (with sources $in\,Q$ and targets $in\,P$) will be used to relate the communication on the input channels of P and Q; the latter will serve a similar purpose for the output channels.

Let P be a base process as in figure 2 and, for every $i \leq m + n$, let $ep_i \stackrel{\text{df}}{=} (B_i, b_i, dom_i, extr_i, ref_i)$ be an extraction pattern such that $B_i \cap B_j = \varnothing$, for all $i \neq j$. We will denote by In the set of the first m extraction patterns ep_i, and

Fig. 2. Base process P and its implementation Q.

by *Out* the remaining n extraction patterns. Moreover, $All \stackrel{\text{df}}{=} In \cup Out$ and *Idch* will denote the set of all uninterpreted channels b_i.

We then take any process Q with the input channels $B_1 \cup \ldots \cup B_m$ and output channels $B_{m+1} \cup \ldots \cup B_{m+n}$, as shown in figure 2, where thick arrows represent *sets* of channels. For such a process, we denote:

- $\tau_{Dom}Q$ is the set of all traces of Q which belong to Dom_{All}.
- $\phi_{Dom}Q$ and $\phi_{dom}Q$ are the sets of those failures of Q in which the trace component belongs to Dom_{All} and dom_{All}, respectively.

Intuitively, $\tau_{Dom}Q$ — which is subsequently referred to as the *domain* of Q — is the set of those traces of Q which are of actual interest and, consequently, $\phi_{Dom}Q$ is the set of failures of actual interest too. Given a failure $(t, R) \in \phi_{Dom}Q$, we will say that an interpreted channel b_i is *blocked* if

$$\alpha B_i \cap R \notin \begin{cases} \overline{ref}_i(t{\restriction}B_i) & \text{if } i \leq m \\ ref_i(t{\restriction}B_i) & \text{otherwise} \end{cases}$$

and denote this by $b_i \in Blocked(t, R)$. Note that in both cases this signifies that the refusal bound imposed by the ref_i has been breached.

We then call Q an *implementation* of P w.r.t. sets of extraction patterns *In* and *Out*, denoted $Q \preceq_{Out}^{In} P$, if the following hold.

DP If t is a trace of Q such that $t{\restriction}in\, Q \in Dom_{In}$, then $t \in \tau_{Dom}Q$.
DF $\tau_{Dom}Q \cap \delta Q = \varnothing$.
TE $extr_{All}(\tau_{Dom}Q) \subseteq \tau P$.
GE If $(t_i)_{i\in\mathbb{N}}$ is an ω-sequence in $\tau_{Dom}Q$, then $(extr_{All}(t_i))_{i\in\mathbb{N}}$ is also ω-sequence.
LC If $(t, R) \in \phi_{Dom}Q$ and $b_i \in Blocked(t, R)$, then $t{\restriction}B_i \in dom_i$.
RE If $(t, R) \in \phi_{dom}Q$, then $(extr_{All}(t), \alpha Blocked(t, R) \cup (R \cap \alpha Idch)) \in \phi P$.

We interpret the above conditions in the following way. DP expresses the *domain preservation* property, which says that if a trace of Q projected on the input channels can be interpreted by *In*, then it must be possible to interpret the projection on the output channels by *Out*. Note that such a condition is a simple *rely/guarantee* property in the sense of [7]. DF can be interpreted as *divergence freedom* within the domain of Q (recall that CSP divergences signify totally unacceptable behaviour). TE simply states that applying *trace extraction* to the domain of Q yields traces of P. GE states that an unboundedly growing sequence of traces in the domain of Q is a sequence of traces unboundedly *growing after extraction* (notice, however, that we place no restriction on the relative growth of the ω-sequences $(t_i)_{i\in\mathbb{N}}$ and $(extr_{All}(t_i))_{i\in\mathbb{N}}$). LC means that

going outside the bounds of allowed refusals indicates that the communication on a given interpreted channel may be seen as *locally completed* (notice also that the communication on any uninterpreted channel is always seen as locally completed). Finally, RE states a condition for *refusal extraction*, which means that if a trace is locally completed on all channels, then any blocking of an interpreted channel of P in Q is transformed into the refusal of its whole alphabet in P; moreover, the refusals on the uninterpreted channels in Q should be matched in P.

4.1 Realisability and compositionality

A direct comparison of an implementation process Q with the corresponding base process P is only possible if there is no difference in the communication interfaces, and all communication is interpreted in exactly the same way. This corresponds to the situation that all of the channels of Q (and of P) are uninterpreted. In such a case, we simply denote $Q \preceq P$ and then we can directly compare the semantics of the two processes in question. Having observed that if all channels are uninterpreted, DF states that $\delta Q = \varnothing$ and RE reduces to $\phi Q \subseteq \phi P$, we obtain

Theorem 1 (realisability). *If $Q \preceq P$ then $Q \sqsupseteq P$.*

That is, the implementation relation 'collapses' to standard CSP refinement in the event that all channels are uninterpreted. And the next result demonstrates that the implementation relation is compositional in the sense that it is preserved by the network composition operation. Taken with the above realisability result, we have met both of the requirements stated in the introduction, and so have a means of compositional verification in the event that corresponding specification and implementation component processes have different interfaces.

Theorem 2 (compositionality). *Let K and L be two base IO processes whose composition is non-diverging, as in figure 3, and, for $X \in \{C, D, E, F, G, H\}$, let EpX be a set of extraction patterns whose targets are the channel set X. Then*

$$M \preceq^{EpC \cup EpH}_{EpD \cup EpE} K \ \wedge \ N \preceq^{EpD \cup EpF}_{EpG \cup EpH} L \implies M \otimes N \preceq^{EpC \cup EpF}_{EpE \cup EpG} K \otimes L \ .$$

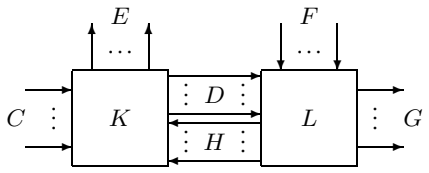


Fig. 3. Base processes in theorem 2

Hence the implementation relation is preserved through network composition, and the only restriction is that the network of the base processes should be designed in a divergence-free way. However, the latter is a standard requirement in the CSP approach (recall again that divergences are regarded as totally unacceptable).

For the example in section 3, it can be shown that $\textsc{Snd}'' \preceq^{ep_c}_{ep_{twice}} \textsc{Snd}$ and $\textsc{Buf}'' \preceq^{ep_{twice}}_{ep_e} \textsc{Buf}$. Hence, by theorem 2 and $\textsc{Snd} \otimes \textsc{Buf} = \textsc{Snd}[e/d]$,

$$\textsc{Snd}'' \otimes \textsc{Buf}'' \preceq \textsc{Snd} \otimes \textsc{Buf} = \textsc{Snd}[e/d] \ ,$$

and so, by theorem 1, $\textsc{Snd}'' \otimes \textsc{Buf}'' \sqsupseteq \textsc{Snd}[e/d]$.

5 Verifying the implementation relation

Extraction patterns and processes may be infinite objects. We therefore need a means to represent them in a finite way in order to allow a computer implementation. To deal with processes we will use the standard device of a transition system, while extraction patterns will be represented by the novel notion of an extraction graph.

5.1 Communicating transition systems

For the purposes of this paper we simply assume that a process is given in the form of a labelled transition system, without worrying about how such a representation has been obtained.[7]

A *communicating transition system* is a tuple $CTS \stackrel{\text{df}}{=} (V, C, D, A, v_0)$ such that: V is a set of states (nodes); $v_0 \in V$ is the initial state; C and D are finite disjoint sets of channels (C will represent input and D output channels); and $A \subseteq V \times (\alpha C \cup \alpha D \cup \{\tau\}) \times V$ is the set of labelled directed arcs, called *transitions*, where τ is a distinguished symbol denoting an *internal* action. We will use the following notation:

- If $(v, a, w) \in A$, we denote $v \stackrel{a}{\longrightarrow} w$ and $a \in en(v)$, calling a *enabled* at v.
- If $\tau \notin en(v)$ then v belongs to the set V_{stb} of *stable* states.
- If $v_1 \stackrel{a_1}{\longrightarrow} v_2 \cdots \stackrel{a_n}{\longrightarrow} v_{n+1}$, we denote $v_1 \stackrel{\langle a_1 \rangle \circ \cdots \circ \langle a_n \rangle}{\longrightarrow} v_{n+1}$, where $\langle \tau \rangle \stackrel{\text{df}}{=} \langle\,\rangle$.
- If $v \stackrel{t}{\rightharpoonup} w$, we denote $v \rightharpoonup w$ or $v \stackrel{t}{\rightharpoonup}$; moreover, $v \stackrel{\langle\,\rangle}{\rightharpoonup} v$, for every $v \in V$.

We shall assume that a CTS is finite, i.e., both V and A are finite.

The implementation relation which we want to verify algorithmically is expressed in the denotational semantics of CSP, and so we must know how to derive information on divergences, traces and failures from a given CTS. For a communicating transition system $CTS = (V, C, D, A, v_0)$, we set $P_{CTS} \stackrel{\text{df}}{=} (C, D, \Phi, \Delta)$

[7] It can be obtained, e.g., using the *operational semantics* defined for CSP in [18].

to be a tuple such that the following hold (below $\mathcal{A} \stackrel{\text{df}}{=} \alpha C \cup \alpha D$):

$$\Delta \stackrel{\text{df}}{=} \{t \circ u \in \mathcal{A}^* \mid \exists v, w \in V : v_0 \stackrel{t}{\rightarrow} v \xrightarrow{\tau} w \stackrel{\langle\rangle}{\rightarrow} v\}$$
$$\Phi \stackrel{\text{df}}{=} \{(t, R) \in \mathcal{A}^* \times \mathbb{P}(\mathcal{A}) \mid \exists v \in V_{stb} : v_0 \stackrel{t}{\rightarrow} v \ \wedge \ R \cap en(v) = \varnothing\} \ \cup \ \Delta \times \mathbb{P}(\mathcal{A}) \ .$$

Note that a divergence is represented by a cycle composed only of τ-labelled transitions which is reachable from the initial state.

Proposition 3. *Let CTS be a communicating transition system as above.*

1. *P_{CTS} is a CSP process.*
2. *$\tau P_{CTS} = \{t \mid v_0 \stackrel{t}{\rightarrow}\}$, provided that $\Delta = \varnothing$.*

Figure 4 shows the graphs of four communicating transition systems. It is not difficult to check that:

$$\begin{array}{ll} P_{CTS_{buf}} = \textsc{Buf} & P_{CTS_{snd}} = \textsc{Snd} \\ P_{CTS_{buf''}} = \textsc{Buf}'' & P_{CTS_{snd''}} = \textsc{Snd}'' \ . \end{array}$$

For a base process P, a CTS representation CTS will later be modified by applying the *normalisation* procedure detailed in [18]. Its result can be given as a finite CTS denoted by CTS^n which is deterministic (i.e., if $v \xrightarrow{a} w$ and $v \xrightarrow{a} w'$ then $w = w'$) and τ-free (i.e., if $v \xrightarrow{a} w$ then $a \neq \tau$), together with a mapping for its nodes, κ, such that:

$$\kappa(p) \stackrel{\text{df}}{=} \{en(v) \mid v \in V \wedge \exists t : p_0 \stackrel{t}{\rightarrow} p \wedge v_0 \stackrel{t}{\rightarrow} v\} \ , \tag{1}$$

where V are the nodes of CTS, and v_0 and p_0 are respectively the initial states of CTS and CTS^n.

5.2 Extraction graphs

We now turn to the representation of extraction patterns. An *extraction graph* is a tuple $EG \stackrel{\text{df}}{=} (B, b, V, A, v_0, \varrho, V^\delta)$ such that: B is a non-empty finite set of channels; b is a channel; V is a set of nodes; $A \subseteq V \times (\alpha B \times \alpha b^*) \times V$ is a set of labelled arcs; $v_0 \in V$ is the initial node; ϱ is a mapping returning for every node in V a non-empty subset-closed family[8] of proper subsets of αB; and $V^\delta \subseteq V$. Intuitively, ϱ corresponds to *ref*, and V^δ indicates traces in *dom*. We will use the following notation:

- If $(v, a, t, w) \in A$, we denote $v \xrightarrow{a:t} w$.
- If $v_1 \xrightarrow{a_1:t_1} v_2 \cdots \xrightarrow{a_n:t_n} v_{n+1}$, we denote $v_1 \xrightarrow{\langle a_1,\ldots,a_n\rangle : t_1 \circ \cdots \circ t_n} v_{n+1}$.

[8] As we intended to concentrate on basic ideas behind verification algorithms, we simplified several issues which a practical implementation would need to address, e.g., each $\varrho(v)$ could be represented by the set of its maximal elements.

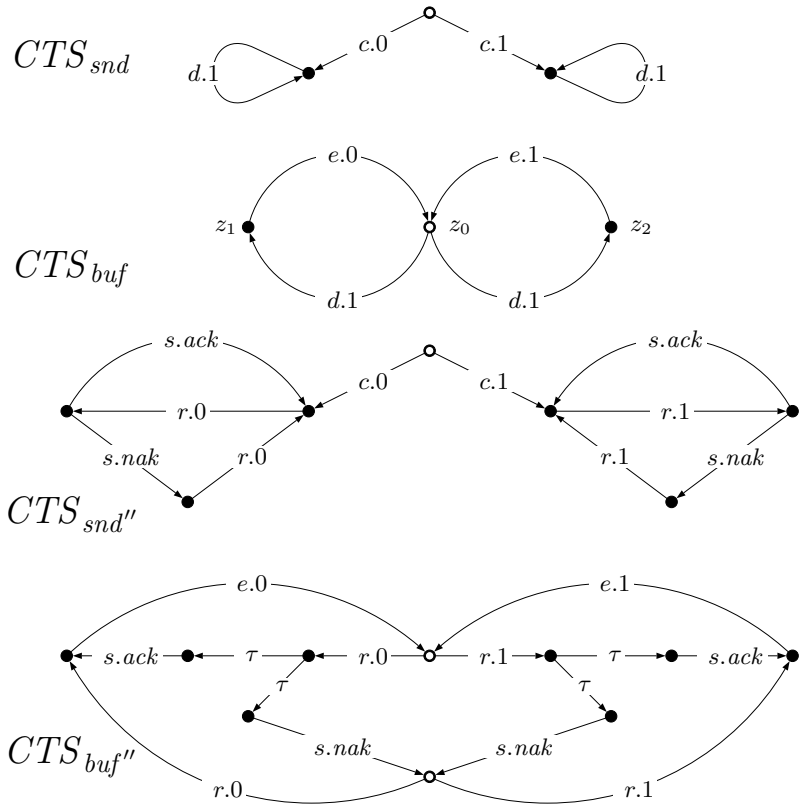


Fig. 4. Communicating transition systems representing CSP processes used throughout the paper (initial states have white centres).

– If $v \xrightarrow{u:t} w$, we denote $v \rightharpoonup w$ and $v \xrightarrow{u:t}$; moreover, $v \xrightarrow{\langle\rangle:\langle\rangle} v$, for every $v \in V$.

We then impose the following restrictions on every node $v \in V$:

EG1 There is $w \in V^\delta$ such that $v_0 \rightharpoonup v \rightharpoonup w$.

EG2 If $v \xrightarrow{a:t} w$ and $v \xrightarrow{a:t'} w'$, then $t = t'$ and $w = w'$.

EG3 If $R \in \varrho(v)$ and $a \in \alpha B$ are such that there is no t satisfying $v \xrightarrow{\langle a\rangle:t}$, then $R \cup \{a\} \in \varrho(v)$.

We now define the extraction pattern an extraction graph represents. For an extraction graph $EG = (B, b, V, A, v_0, \varrho, V^\delta)$, Dom_{EG} is a set of traces and $ep_{EG} \stackrel{\text{df}}{=} (B, b, dom_{EG}, extr_{EG}, ref_{EG})$ is a tuple, given in the following way:

– $Dom_{EG} \stackrel{\text{df}}{=} \{u \in \alpha B^* \mid \exists t : v_0 \xrightarrow{u:t}\}$.

– $dom_{EG} \stackrel{\text{df}}{=} \{u \in \alpha B^* \mid \exists v \in V^\delta \ \exists t : v_0 \stackrel{u:t}{\rightarrow} v\}$.
– By EG2, for every $u \in Dom_{EG}$, there are *unique* t and v such that $v_0 \stackrel{u:t}{\rightarrow} v$. We then set $extr_{EG}(u) \stackrel{\text{df}}{=} t$ and $ref_{EG}(u) \stackrel{\text{df}}{=} \varrho(v)$.

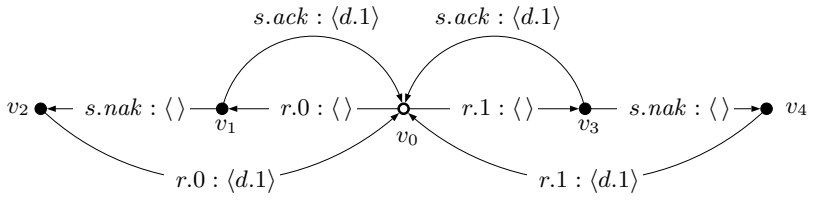
From the point of view of practical implementation, we will be interested only in those extraction graphs which are finite, i.e., have a finite number of nodes V (note that the finiteness of A follows then from αB being finite and EG2).

Proposition 4. *ep_{EG} is an extraction pattern, and $Dom_{EG} = Dom_{ep_{EG}}$.*

Conversely, one can easily see that for every extraction pattern ep there is an extraction graph EG such that $ep = ep_{EG}$. For the identity extraction pattern ep_c, the corresponding extraction graph EG_c will have the mapping ϱ undefined, $V = V^\delta \stackrel{\text{df}}{=} \{v_0\}$, and $A \stackrel{\text{df}}{=} \{(v_0, a, \langle a \rangle, v_0) \mid a \in \alpha c\}$. The extraction pattern ep_{twice} defined in section 3.2, can be represented by the extraction graph EG_{twice}, shown in figure 5. It is easy to check that $ep_{EG_{twice}} = ep_{twice}$ and $ep_{EG_c} = ep_c$. For example, that $t \stackrel{\text{df}}{=} \langle r.1, s.ack, r.0 \rangle$ belongs to the domain Dom of ep_{twice} follows from the existence of the path

$$v_0 \xrightarrow{r.1:\langle\,\rangle} v_3 \xrightarrow{s.ack:\langle d.1\rangle} v_0 \xrightarrow{r.0:\langle\,\rangle} v_1$$

in EG_{twice}. Moreover, we have $extr(t) = \langle\,\rangle \circ \langle d.1\rangle \circ \langle\,\rangle = \langle d.1\rangle$, $ref(t) = \varrho(v_1) = \mathbb{P}(\{r.0, r.1\})$, and $t \notin dom$ since $v_1 \notin V^\delta$.



v	$\varrho(v)$
v_0	$\mathbb{P}(\{s.ack, s.nak, r.0\}) \cup \mathbb{P}(\{s.ack, s.nak, r.1\})$
v_1 and v_3	$\mathbb{P}(\{r.0, r.1\})$
v_2	$\mathbb{P}(\{s.ack, s.nak, r.1\})$
v_4	$\mathbb{P}(\{s.ack, s.nak, r.0\})$

Fig. 5. Extraction graph EG_{twice}, where $V^\delta \stackrel{\text{df}}{=} \{v_0\}$, representing the extraction pattern ep_{twice}.

In the rest of the paper, we will assume that we can extract at most one event out of a single event over the source channels, i.e., for every extraction mapping and a trace $t \circ \langle a \rangle$ in its domain,

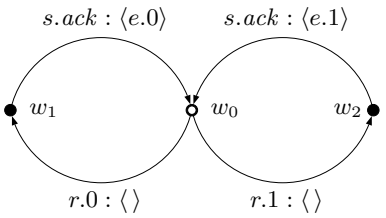
$$|extr(t \circ \langle a \rangle)| \leq 1 + |extr(t)| \,. \tag{2}$$

In terms of extraction graphs, this means that $|t| \leq 1$ whenever $v \xrightarrow{a:t} w$. The above assumption has been introduced to simplify the presentation; it could be omitted at the cost of slightly complicating (but not losing) the subsequent results.

5.3 Unambiguous CTSs of implementation processes

Extraction patterns and so extraction graphs are defined for channels in a base process P and channels in an implementation process Q. As a result, more than one EG will usually be required to interpret the behaviour of the implementation process Q as a whole. Moreover, it is possible that the CTS representing Q will be ambiguous (in the sense explained below) with respect to interpretation in terms of the EGs.

Let us again consider a base process BUF modelling a buffer of capacity one, with input channel d and output channel e, defined in section 3.1, which is modelled by the communicating transition system CTS_{buf} shown in figure 4. Let us also consider two extraction patterns, ep_1 and ep_2, given by the extraction graphs EG_1 and EG_2, i.e., $ep_i = ep_{EG_i}$, for $i = 1, 2$. The first extraction graph, $EG_1 \stackrel{\text{df}}{=} EG_d$, is the identity extraction graph for the uninterpreted channel d. The second one, over the sources $\{r, s\}$ and target e, is given in figure 6. Note that unlike in our previous example, it is now the input channel of BUF which is uninterpreted, while the communication on the output channel, e, is implemented using a simple 'ping-pong' communication protocol.



v	$\varrho(v)$
w_0	$\mathbb{P}(\{s.ack, r.0\}) \cup \mathbb{P}(\{s.ack, r.1\})$
w_1 and w_2	$\mathbb{P}(\{r.0, r.1\})$

Fig. 6. Extraction graph EG_2, where $V^\delta \stackrel{\text{df}}{=} \{w_0\}$.

We would like to verify the implementation conditions, with respect to ep_1 and ep_2, for a process Q such that $in\,Q \stackrel{\text{df}}{=} \{d\}$ and $out\,Q \stackrel{\text{df}}{=} \{r, s\}$, and whose behaviour is described by the communicating transition system CTS shown in figure 7(a), i.e., $Q = P_{CTS}$. Although it is not difficult to see that $Q \preceq^{ep_1}_{ep_2}$ BUF, it may not be clear what needs to be done to verify this using communicating transition systems and extraction graphs. In particular, suppose that we want to verify that TE holds, i.e., $extr_{\{ep_1, ep_2\}}(\tau_{Dom} Q) \subseteq \tau$BUF. A possible attempt would be to replace each of the arc annotations in CTS by its 'extracted' version given by the corresponding extraction pattern. This could be done for all the actions except $s.ack$ from which we can extract either $\langle e.0 \rangle$ or $\langle e.1 \rangle$, depending on the previous actions executed by the process. Thus CTS is an *ambiguous*

representation of Q w.r.t. the extraction graph EG_2. Note that the problem is caused by our wish to represent an extraction mapping (from traces to traces) in terms of individually labelled arcs, and so a node in a CTS needs to encode the appropriate history of reaching it from the initial state. But, in our case, x_3 can reached in two different ways

$$x_0 \xrightarrow{d.0} x_1 \xrightarrow{r.0} x_3 \quad \text{and} \quad x_0 \xrightarrow{d.1} x_2 \xrightarrow{r.0} x_3$$

which imply different interpretation of the arc $x_3 \xrightarrow{s.ack} x_0$. A solution we propose is to remove this ambiguity, by suitably modifying CTS. More precisely, we split the node x_3 of CTS and separate the two arcs incoming to it, obtaining CTS' shown in figure 7(b). We can now unambiguously interpret each of the arc annotations, which leads to the graph G shown in figure 7(c). To verify that TE holds, it now suffices to check that the traces generated by G are also generated by CTS_{buf}.

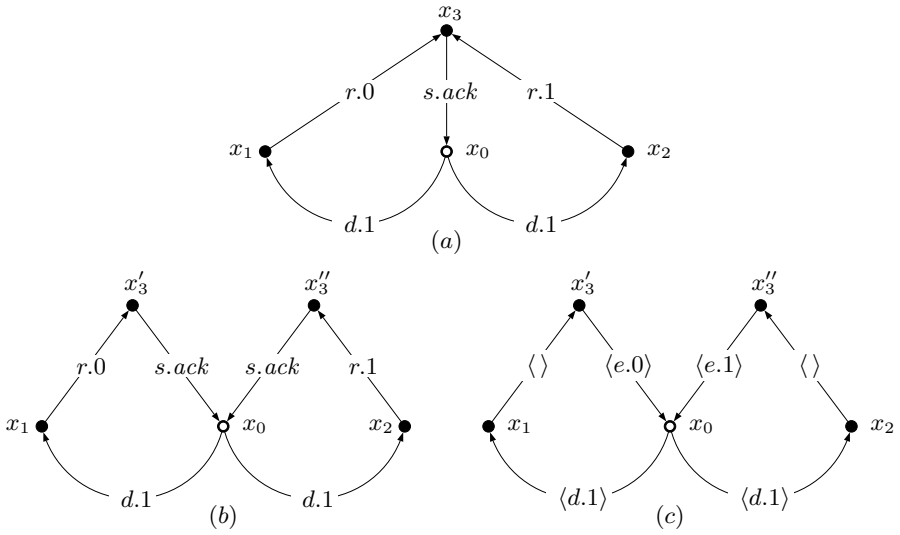


Fig. 7. Disambiguating an implementation of a buffer of capacity one.

The following algorithm makes the above construction precise, generating an equivalent unambiguous CTS, from a given CTS and a set of extraction graphs.

Algorithm 1. *For $i = 1, \ldots, m+n$, let $EG_i \stackrel{\text{df}}{=} (B_i, b_i, V_i, A_i, v_{0i}, \varrho_i, V_i^{\delta})$ be extraction graphs such that the B_i's are mutually disjoint and the b_i's distinct. Moreover, let $CTS = (V, C, D, A, v_0)$ be a communicating transition system such that $C = B_1 \cup \ldots \cup B_m$ and $D = B_{m+1} \cup \ldots \cup B_{m+n}$. The algorithm generates a communicating transition system CTS^u, in two steps.*

Step 1: *We first generate a labelled directed graph G, with the set of nodes $V \times V_1 \times \cdots \times V_n$, as follows. Let $q = (v, v_1, \ldots, v_n)$ be a node in G. The arcs outgoing from q are derived from those outgoing from v, and for each arc $v \xrightarrow{a} w$ in CTS we proceed according to exactly one of the following four cases.*

1. *$a = \tau$. Then we add a transition $q \xrightarrow{\tau} (w, v_1, \ldots, v_n)$.*
   We also set $extr(q, \tau) \stackrel{\text{df}}{=} \tau$.
2. *$a \neq \tau$ and there is an arc $v_i \xrightarrow{a:t} w_i$ in EG_i, for some $i \geq 1$.*[9] *Then we add a transition $q \xrightarrow{a} (w, w_1, \ldots, w_n)$ where $w_j = v_i$, for all $j \neq i$.*
   We also set $extr(q, a) \stackrel{\text{df}}{=} \tau$ if $t = \langle\,\rangle$, and $extr(q, a) \stackrel{\text{df}}{=} b$ if $t = \langle b \rangle$.[10]
3. *$a \in \alpha C$ and there is no t and w_i such that $v_i \xrightarrow{a:t} w_i$, for any $i \leq m$. Then we do nothing.*
4. *$a \in \alpha D$ and there is no t and w_i such that $v_i \xrightarrow{a:t} w_i$, for any $i > m$. Then we mark q as an* unfinished *node (each node is assumed to be* finished *at the beginning).*

Step 2: *From G we obtain a communicating transition system CTS^u with the same channels as CTS, by taking $q_0 \stackrel{\text{df}}{=} (v_0, v_{01}, \ldots, v_{0n})$ as the initial node, and then adding all the nodes reachable from q_0, together with all the interconnecting transitions. If any of the reachable nodes is marked as unfinished, we* reject *CTS^u.*[11]

The above algorithm will be executed on the CTS representation of the *implementation* process Q. Its main characteristic is that the definition of the nodes allows the unambiguous interpretation of the arc labels through the extraction mappings (c.f. proposition 5). In practice, one can avoid generating the whole graph G, by performing a depth first search starting from the initial node q_0. Then only the nodes of CTS^u will be visited.

The graph G for the example in figure 6 is shown in figure 8(a), where the $*$'s indicate unfinished nodes. After restricting ourselves only to the relevant subgraph (comprising nodes reachable from the initial one), we obtain the graph shown in figure 8(b) which is isomorphic to CTS' obtained informally before. Note that $extr((x_3, v_0, w_1), s.ack) = e.0$ and $extr((x_1, v_0, w_0), r.0) = \tau$.

We may now state why CTS^u can be regarded as *unambiguous*.

Proposition 5. *Let All be the set of extraction patterns generated by the EG_i's. If $q_0 \xrightarrow{a_1} q_1 \cdots \xrightarrow{a_k} q_k$ in CTS^u and $t = \langle a_1 \rangle \circ \cdots \circ \langle a_k \rangle$, then $t \in Dom_{All}$ and $extr_{All}(t) = u_1 \circ \cdots \circ u_k$, where $u_i \stackrel{\text{df}}{=} \langle extr(q_{i-1}, a_i) \rangle$, for every $1 \leq i \leq k$.*

[9] There can only be one such EG_i since the B_i's are mutually disjoint.
[10] Note that $extr(q, a)$ is a well defined single action by EG2 and (2).
[11] Since this means that the traces generated by CTS^u do not satisfy the condition DP (c.f. the proof of proposition 6).

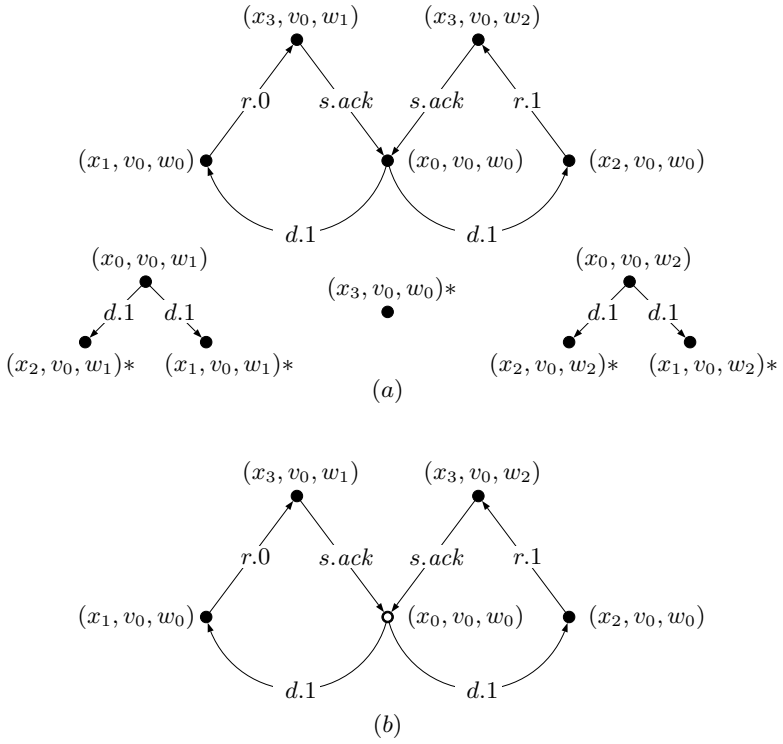


Fig. 8. Applying disambiguating algorithm.

5.4 Checking the implementation conditions

In this section, we will outline how to check the implementation conditions, formulated in terms of the denotational semantics of CSP, using communicating transition systems and extraction graphs. We will assume the following:

- P, Q, $Idch$ and the ep_is are as in section 4.
- CTS_P and CTS_Q are communicating transition systems representing P and Q respectively, i.e., $P = P_{CTS_P}$ and $Q = P_{CTS_Q}$.
- For every $i \leq m + n$, EG_i is an extraction graph such that $ep_{EG_i} = ep_i$.
- CTS^u is a disambiguated version of CTS_Q as in algorithm 1, with the initial state q_0.
- CTS^n is the normalised version of CTS_P (see section 5.1), with the initial state p_0, and κ is the mapping given in (1).

We first obtain that testing for DP is done while generating CTS^u, and testing for DF amounts to checking for the presence of τ-loops in the graph of CTS^u.

Proposition 6. *Q satisfies DP and DF* iff *CTS^u has not been rejected (see Step 2 of algorithm 1), and there are no nodes v and w in CTS^u such that $v \xrightarrow{\tau} w \xrightarrow{\langle\rangle} v$.*

From now on, we assume that CTS^u has been successfully generated and does not contain any τ-loops, and so DP and DF hold for Q.

A relation $sim \subseteq V_{CTS^u} \times V_{CTS^n}$ is a *simulation* for CTS^u and CTS^n if $(q_0, p_0) \in sim$ and, for every $(q, p) \in sim$,

$$q \xrightarrow{a} q' \implies \exists p' : p \xrightarrow{\langle extr(q,a)\rangle} p' \ \wedge \ (q', p') \in sim \ . \tag{3}$$

Proposition 7. *Q satisfies TE* iff *there is a simulation for CTS^u and CTS^n.*

Note also that since CTS^n is deterministic and τ-free, if there exists a simulation for CTS^u and CTS^n, then there exists the smallest one, sim_{min}.

One can attempt to construct the minimal simulation sim_{min}, by a depth-first traversal of the product $V_{CTS^u} \times V_{CTS^n}$, starting at (q_0, p_0) and then following the arcs given by the first component in the nodes representing the product. If the construction is successful, the set of all the nodes reachable from (q_0, p_0) gives the minimal simulation.

We will now additionally assume that Q satisfies TE. Then, testing for the next implementation condition, GE, amounts to checking for extracted τ-loops in the graph of CTS^u.

Proposition 8. *Q does not satisfy GE* iff *there are nodes $v_1, \ldots, v_k$ ($k \geq 2$) in CTS^u such that $v_1 \xrightarrow{a_1} v_2 \cdots \xrightarrow{a_{k-1}} v_k (= v_1)$ and $extr(v_i, a_i) = \tau$, for every $i < k$.*

Finally, for every stable state $q = (v, v_1, \ldots, v_{m+n})$ of CTS^u, let $Blocked(q)$ denote the set of all interpreted channels b_i such that

$$\alpha B_i - en(q) \notin \begin{cases} \overline{\varrho}_i(v_i) & \text{if } i \leq m \\ \varrho_i(v_i) & \text{otherwise} \ , \end{cases}$$

where $\overline{\varrho}_i(q_i)$ comprises all $X \subseteq \alpha B_i$ such that $X \cup Y \neq \alpha B_i$, for all $Y \in \varrho_i(q_i)$.

Proposition 9. *Q satisfies LC and RE* iff *the following are satisfied, for every stable state $q = (v, v_1, \ldots, v_{m+n})$ of CTS^u:*

1. *$b_i \in Blocked(q)$ implies $v_i \in V_i^\delta$.*
2. *$(q, p) \in sim_{min}$ and $v_i \in V_i^\delta$ (for all $1 \leq i \leq m+n$) implies that there is $A \in \kappa(p)$ satisfying $(\alpha Blocked(q) \cup (\alpha Idch - en(q))) \cap A = \varnothing$.*

The first condition in the last proposition can by checked by traversing the graph of CTS^u, while the second condition can be checked on the occasion of testing for TE.

6 Conclusions

In this work we have investigated the notion that a system (the *implementation*) implements another one (the *base* or *specification*), in the event that their interfaces differ. Such an issue has an obvious interest with respect to system design and development, in which interface *refinement* is often a major aspect of the added detail provided by the implementation compared to the specification.

The proposed treatment is based upon the combined use of:

- The standard CSP process model [9, 18], in which we describe both specifications and implementations.
- The notion of *extraction pattern*, aimed to capture the semantics of the interface refinement by which specification and implementation differ.
- An *implementation relation*, required to hold between the behaviour of the specification and that of the implementation, after the latter has been interpreted according to the extraction patterns that describe how their respective interfaces differ.

On all the above aspects, the present work significantly extends our previous work [11, 12, 5]: the class of admissible specifications has been enlarged to include any, non-diverging, CSP process; the definition of extraction pattern has been technically improved; and the multiple implementation relation schemes previously proposed have been unified into a single one. Furthermore, the latter appears to be most appropriate as the vertical implementation relation for CSP, given that, by realisability (theorem 1), it collapses into the standard horizontal implementation of [9]. In other words, an implementation in the sense of this work turns out to be an implementation in the standard sense, as soon as it and its intended specification possess the *same* interface (i.e., no interface refinement has actually been performed, and the extraction patterns in the implementation relation are the identity ones).

It is worth noting that the implementation relation has been 'trimmed', in the sense that its 'weak' and 'strong' counterparts of [12] were in fact too much so. This was immaterial as to the restricted base process class treated there, but prevented a straightforward extension of that approach to encompass all non-diverging base process, as we manage to do now.

Our implementation relation has also been shown to be useful for verification, for it enjoys a kind of compositionality, in that a specification composed of several connected systems may be implemented by connecting their respective implementations (theorem 2). This means that the process of verification may deal separately with individual component processes. This avoids a major cause of the state explosion problem, and permits a bottom-up style *reuse* of verifications carried out separately for specific components.

In the framework at hand, in which interfaces of the implementation and the specification may differ, compositionality, together with realisability, ensures that the specification can actually be built by 'plugging' the implementation into a suitable environment. A technical treatment of this issue has not been provided here, for it would essentially adhere to the lines of our work [11].

Finally, in the present work we build on the preliminary results of [4] (which was based on older implementation relations), and develop efficient algorithms for automatically verifying the present implementation relation. In this process, we take further advantage of compositionality, which allows us to verify each component of an implementation system separately, against the respective specification component. This avoids, now from the point of computer-aided verification, one of the great sources of the state explosion problem, i.e., the generation of a state space which is a (substantial) subset of the product of all the component state spaces.

As the basis for mechanical verification, we render both processes and extraction patterns as graphs (communicating transition systems and extraction graphs, respectively). This enables us to establish a graph-theoretic characterization of the implementation relation, from which we directly derive efficient algorithms for the modular verification of the conditions DP, DF, TE, GE, LC, RE whose conjunction amounts to the relation as a whole.

Future work will explore possibilities for further optimisation, as well as including examples and a case study to evaluate the performance of the verification algorithms in practice. Envisaged application areas range over the entire field of distributed systems; in particular, fruitful results can be expected in the realm of fault tolerance, exploiting the experience accrued from our analysis of N-Modular Redundancy [11] and Coordinated Atomic Actions [6].

Related work We now compare the work presented in this paper with other approaches whose goal is similar or somehow related.

Looking at the general issue of behaviour abstraction, some approaches (e.g., [2]) describe system behaviour by sequences of state tuples with an internal component; they then require that, for every possible state sequence of a correct implementation, there should exist one of the specification such that the two sequences coincide after deleting the internal state component. A similar treatment is presented in [10], using infinite action sequences (i.e., infinite traces) instead of state sequences; the interface of the specification must be a subset of the interface of the implementation, and it is required that every trace of the implementation can be turned into one of the specification by deleting actions not in the specification's interface. These two approaches and other comparable ones (like, e.g., [14, 20]) are based on abstraction *by hiding*. In contrast, our notion of abstraction is essentially based on the interpretation of traces over a set of channels, as traces over another channel. Abstraction by hiding is certainly useful but, unlike our notion of abstraction by interpretation, does not cater for interface refinement, an essential tool for system design.

The interface displacement approach proposed in [3] has interesting similarities with our work: (1) their interface transducers play a role comparable to that of our disturbers and extractors of [11]; (2) both treatments constrain the system's and the environment's mutual refusals.

The essential difference is that the treatment of [3] is geared to the refinement of a specification *compound* system, whose components interact through an interface, into a compound implementation, whose components interact through a

different (typically refined) interface. This refinement is then validated by checking that the interface change is acceptable in some appropriate sense, and, above all, that the compound implementation is correct with respect to the compound specification. This notion of correct implementation is however a traditional (*horizontal* in the words of [17]) one, for compound specification and implementation have exactly the same interface, the (different) interfaces where their respective components interact being hidden. Essentially, in the interface displacement, it is possible to compare only processes at one side of the interface; the addition or removal of the interface transducers serves as the parameter of the interface refinement.

Abstraction and refinement are clearly complementary and apt to be related, once the formal framework to work in is selected. However, it should be noted that in many refinement-based approaches, such as that of *action refinement* dating back to [1], every high level action must be refined into a precise behaviour made up of low level actions. It would appear, then, that this kind of refinement is mainly suited for the design of a well-identified implementation from a specification. It can, for example, be employed in contexts where the implementor decides to replace any actions a and b performed by the specification, by the sequences $a_1a_2a_3$ and $b_1b_2b_3$ by the implementation.

While we do not see any reason why abstraction could not be used in a similar way, we believe it also affords a distinct benefit. It allows one to stipulate, for example, that whenever an implementation performs any of a_i and a_j for $i, j = 1, 2, 3, i{\neq}j$, *whatever other actions occur between them*, this should be construed as a occurring at the abstract level. This is clearly suited to modelling the relative unpredictability of a fault-prone environment in important fault-tolerant applications like N-Modular Redundancy, as we did, e.g., in [11]. It is difficult to see how similar accomplishments could be performed within most treatments adhering to the action refinement approach.

Similar remarks can be found in Rensink and Gorrieri's work [17], which aims at overcoming these and other limitations, while staying within the realm of action refinement. Their solution is, like ours, based upon a parametric *vertical* implementation relation. Their parameter is a refinement function; conversely, ours is essentially an abstraction mapping. They maintain their approach has some decisive advantages over other action refinement based ones:

- flexibility: it allows multiple implementations for a given specification, and does not dictate a strict ordering for the low level actions implementing an high level one;
- simplicity: it does not require the introduction of a concurrency model more complex than any of the standard interleaving ones;
- the vertical implementation relation collapses to a standard horizontal one when the refinement is the identity;
- deadlock properties carry over from the abstract to the concrete level;
- it allows compositional verification in the same sense as our approach;
- it can be endowed with a decision procedure for verification of the implementation relation.

We have highlighted these merits simply to remark that they also characterize the approach of the present paper. We do lack, however, at least for the time being, a proof system where the implementation relation can be decided.

Nonetheless, our approach and that of [17] differ greatly from a technical point of view. This is a consequence of the different concurrency models employed. In this respect, it can be noted that our CSP model has the advantage of being firmly rooted in intuitive notions like traces and refusals. This has two appealing consequences: (i) our abstraction mappings can be defined in a natural fashion, directly over traces and refusals, (ii) we do not need to change the model in any way for our purposes. On the other hand, the treatment of [17] is based upon bisimulation semantics and refines actions into processes, as is customary in action refinement. As a result, in order to regain flexibility, it has to tweak horizontal bisimulation quite a bit in order to obtain the intended vertical version. Of course, any preference for either approach, in this respect, may be a matter of taste.

A crucial issue for comparison, instead, would be to assess to what extent the approach of [17] is applicable to fault-tolerant systems in the sense highlighted above for ours (see also [11]). The fact that in [17] actions can only be refined into deterministic (τ free) processes could indicate that some limitations can be expected in this field of application.

Finally with respect to [17], we mention the fact that their decision procedure may be used only in very restricted cases, since, during verification, when a concrete action is explored in the implementation it may be necessary to 'guess' which abstract action it is refining, unless the restrictions are imposed.

It is also interesting to compare the implications of the differences between the concurrency models underlying our work and [10]. Both models are action-based, but [10] employs infinite traces and only caters for asynchronous communication. In contrast, our CSP model has finite traces and refusals, permitting asynchronous as well as synchronous communication, and enabling deadlock properties to be described. As a result, no artificial constraint has to be placed on the behaviour of the *implementation* system: faulty channels need not communicate asynchronously, and faulty modules (within fault-tolerant implementations) need not be processes of a restricted class (e.g., *IO*). This is certainly desirable in order to reflect faithfully the unpredictable nature of faults, but is impossible in the model of [10] or in other models of asynchronous communication. On the other hand, it is fair to add that liveness, in the sense of [10] or temporal logic, cannot be expressed with our finite traces; however, the ability to place constraints on refusals is generally deemed a reasonable alternative in the CSP philosophy.

The approach of [19] to the formal analysis of fault-tolerance is quite different from ours, which basically provides a criterion whereby a system, albeit fault-prone, may be abstractly viewed as an implementation of a correct specification system. In contrast, in [19] the system under study (modelled using the CSP *trace model*) may embed fault-prone components, but its externally observable behaviour must be correct itself, without the filter of abstraction.

Acknowledgments

The first author was supported by an EPSRC grant. We would also like to thank Marta Pietkiewicz-Koutny for her comments on an earlier version of this paper.

References

1. L. Aceto and M. C. B. Hennessy: Towards Action-Refinement in Process Algebras. *Information and Computation* 103 (1993) 204-269.
2. M. Abadi and L. Lamport: The Existence of Refinement Mappings. *Theoretical Computer Science* 82 (1991) 253-284.
3. 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) .
4. J. Burton, M. Koutny and G. Pappalardo: Verifying Implementation Relations in the Event of Interface Difference. Proc. of *FME 2001*, J. N. Oliveira and P. Zave (Eds.). LNCS 2021 (2001) 364-383.
5. J. Burton, M. Koutny and G. Pappalardo: Implementing Communicating Processes in the Event of Interface Difference. Proc. of *ACSD 2001*, A. Valmari and A. Yakovlev(Eds.). IEEE Computer Society (2001) 87-96.
6. 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) 1-20.
7. P. Collette and C. B. Jones: Enhancing the Tractability of Rely/Guarantee Specifications in the Development of Interfering Operations. Technical Report CUMCS-95-10-3, Department of Computing Science, Manchester University (1995).
8. M. C. B. Hennessy: *Algebraic Theory of Processes*. MIT Press (1988).
9. C. A. R. Hoare: *Communicating Sequential Processes*. Prentice Hall (1985).
10. B. Jonsson: Compositional Specification and Verification of Distributed Systems. *ACM TOPLAS* 16 (1994) 259-303.
11. 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.
12. M. Koutny and G. Pappalardo: Behaviour Abstraction for Communicating Sequential Processes. *Fundamenta Informaticae* 48 (2001) 21-54.
13. L. Lamport: The Implementation of Reliable Distributed Multiprocess Systems. *Computer Networks* 2 (1978) 95-114.
14. N. A. Lynch and M. R. Tuttle: Hierarchical Correctness Proofs for Distributed Algorithms. Proc. of *6th PODC*, ACM (1987) 137-151.
15. L. V. Mancini, G. Pappalardo: Towards a Theory of Replicated Processing. Proc. of *Formal Techniques in Real-Time and Fault-Tolerant Systems*, M. Joseph (Ed.). Springer-Verlag, Lecture Notes in Computer Science 331 (1988) 175–192.
16. R. Milner: *Communication and Concurrency*. Prentice Hall (1989).
17. A. Rensink and R. Gorrieri: Vertical Implementation. *Information & Computation* 170 (2001) 95–133.
18. A. W. Roscoe: *The Theory and practice of Concurrency*. Prentice-Hall (1998).
19. H. Schepers and J. Hooman: Trace-based Compositional Reasoning About Fault-tolerant Systems. Proc. of *PARLE'93*, A. Bode, M. Reeve, and G. Wolf (Eds.). Springer-Verlag, Lecture Notes in Computer Science 694 (1993) .

20. E. W. Stark: Proving Entailment Between Conceptual State Specifications. *Theoretical Computer Science* 56 (1988) 135-154.

A CSP operators

We here provide formal definitions of CSP operators used in our examples. In the first two operations on processes it is assumed that P and Q have the same input channels and the same output channels. Then the non-deterministic choice $(P \sqcap Q)$ and deterministic choice $(P[]Q)$ are processes with the same channels as P and Q, the divergences being the union of those of P and Q, and the failures given respectively by $\phi(P \sqcap Q) \stackrel{\text{df}}{=} \phi P \cup \phi Q$ and

$$\phi(P[]Q) \stackrel{\text{df}}{=} \{(\langle\,\rangle, R) \mid (\langle\,\rangle, R) \in \phi P \cap \phi Q\} \cup \{(t, R) \mid t \neq \langle\,\rangle \ \wedge \ (t, R) \in \phi P \cup \phi Q\} \ .$$

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

$$\begin{aligned} \delta(a \rightarrow P) &\stackrel{\text{df}}{=} \{\langle a \rangle \circ t \mid t \in \delta P\} \\ \phi(a \rightarrow P) &\stackrel{\text{df}}{=} \{(\langle a \rangle \circ t, R) \mid (t, R) \in \phi P\} \cup \{\langle\,\rangle\} \times \mathbb{P}(\alpha P - \{a\}) \ . \end{aligned}$$

The last operator we need is channel renaming. Let P be a process with a channel $b \in \chi P$, and b' be a channel not in χP such that $\mu b = \mu b'$. Then $P[b'/b]$ is a process that behaves like P except that each action $b.v$ is replaced by $b'.v$. Formally, $\chi P[b'/b] \stackrel{\text{df}}{=} \chi P - \{b'\} \cup \{b\}$ and

$$\begin{aligned} \delta P[b'/b] &\stackrel{\text{df}}{=} \{t[b'/b] \mid t \in \delta P\} \\ \phi P[b'/b] &\stackrel{\text{df}}{=} \{(t[b'/b], R[b'/b]) \mid (t, R) \in \phi P\} \end{aligned}$$

where $R[b'/b]$ is R with each action $b.v$ replaced by $b'.v$, and $t[b'/b]$ is a trace obtained from t by replacing each action $b.v$ by $b'.v$.

We also use $\textsc{Stop}_B$, or simply $\textsc{Stop}$ if B is clear from the context, to denote a divergence-free process with channel set B and the failures $\{\langle\,\rangle\} \times \mathbb{P}(\alpha B)$.

In 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 $\textsc{Stop}$ constructs. For example, $P \stackrel{\text{df}}{=} (a \rightarrow P)[](b \rightarrow \textsc{Stop})$ 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., [18] for a full account of this aspect of CSP.

To relate failures of a composite process with those of the constituent processes, we will use an auxiliary notation. Let P and Q be two processes forming a network, $Z \stackrel{\text{df}}{=} P \otimes Q$ and $(t, R) \in \alpha Z^* \times \mathbb{P}(\alpha Z)$. Then $\Re_{P,Q}(t, R)$ will denote the set of all triples

$$(w, R_P, R_Q) \in (\alpha P \cup \alpha Q)^* \times \mathbb{P}(\alpha P) \times \mathbb{P}(\alpha Q)$$

such that $t = w{\restriction}\chi Z$, $(w{\restriction}\chi P, R_P) \in \phi P$, $(w{\restriction}\chi Q, R_Q) \in \phi Q$ and $R_P \cup R_Q = R \cup (\alpha P \cap \alpha Q)$. Directly from the definitions of parallel composition and hiding we obtain

Lemma 10. *Let P and Q be as above.*

1. *If $\Re_{P,Q}(t,R) \neq \varnothing$ then $(t,R) \in \phi Z$.*
2. *If $(t,R) \in \phi Z$ and $t \notin \delta Z$, then $\Re_{P,Q}(t,R) \neq \varnothing$.*

Intuitively, $\Re_{P,Q}(t,R)$ comprises *realisations* of a failure (t,R) of $P \otimes Q$ in terms of failures of the underlying processes, P and Q. In general, there may be more than one realisation of a given (t,R).

B Proofs for section 4

Lemma 11. *Let $Ep \stackrel{\text{df}}{=} \{ep_1, \ldots, ep_n\}$ be a set of extraction patterns with disjoint sources and distinct targets. Moreover, for some distinct $i_1, \ldots, i_k$ ($k \geq 1$), let $B' \stackrel{\text{df}}{=} B_{i_1} \cup \ldots \cup B_{i_k}$, $C' \stackrel{\text{df}}{=} \{b_{i_1}, \ldots, b_{i_k}\}$ and $Ep' = \{ep_{i_1}, \ldots, ep_{i_k}\}$.*

1. *Dom_{Ep} is the prefix-closure of dom_{Ep}.*
2. *$extr_{Ep}$ is monotonic and strict.*
3. *$Dom_{Ep'} = Dom_{Ep}{\restriction}B'$ and $dom_{Ep'} = dom_{Ep}{\restriction}B'$.*
4. *$extr_{Ep'}(t{\restriction}B') = extr_{Ep}(t){\restriction}C'$, for every $t \in Dom_{Ep}$.*

Proof. (1) To show that Dom_{Ep} is prefix-closed, consider $t \circ \langle a \rangle \in Dom_{Ep}$. Then, for every $i \leq n$, $(t \circ \langle a \rangle){\restriction}B_i \in Dom_i$. Since, by EP2, each Dom_i is prefix-closed and $t{\restriction}B_i \leq (t \circ \langle a \rangle){\restriction}B_i$ then, for every $i \leq n$, $t{\restriction}B_i \in Dom_i$. Hence $t \in Dom_{Ep}$, and so $Pref(Dom_{Ep}) = Dom_{Ep}$.

To show that $Dom_{Ep} \subseteq Pref(dom_{Ep})$, suppose that $t \in Dom_{ep}$. Then, for every $i \leq n$, $t{\restriction}B_i \in Dom_i$. Moreover, for every $i \leq n$, there are $u_i \in dom_i$ and w_i such that $t{\restriction}B_i \circ w_i = u_i$. It then follows that $u \stackrel{\text{df}}{=} t \circ w_1 \circ \cdots \circ w_n \in dom_{Ep}$ is such that $t \leq u$ and $u{\restriction}B_i = u_i$, for every $i \leq n$ (note that the B_i's are disjoint sets). Hence $Dom_{Ep} \subseteq Pref(dom_{Ep})$.

Finally, we observe that $dom_{Ep} \subseteq Dom_{Ep}$, as $dom_i \subseteq Dom_i$, for all i.

(2) $extr_{Ep}$ is strict by definition. Moreover, monotonicity follows directly from $extr_{Ep}(t \circ \langle a \rangle) = extr_{Ep}(t) \circ u$, in its definition.

(3) Follows directly from the definitions and the fact that $(t{\restriction}B'){\restriction}B_i = t{\restriction}B_i$, for any $B_i \subseteq B'$.

(4) We proceed by induction on the length of t. In the base case, $t = \langle\,\rangle$, we have $extr_{Ep'}(\langle\,\rangle{\restriction}B') = \langle\,\rangle = extr_{Ep}(\langle\,\rangle){\restriction}C'$ since $extr_{Ep}$ and $extr_{Ep'}$ are strict by part (2). In the induction step, we consider $t' = t \circ \langle a \rangle$. Then

$$\begin{aligned} extr_{Ep'}(t'{\restriction}B') &= extr_{Ep'}(t{\restriction}B' \circ \langle a \rangle{\restriction}B') = extr_{Ep'}(t{\restriction}B') \circ r \\ extr_{Ep}(t'){\restriction}C' &= extr_{Ep}(t){\restriction}C' \circ u{\restriction}C' \,, \end{aligned}$$

where r is (possibly empty) trace, and u such that $extr_{Ep}(t \circ \langle a \rangle) = extr_{Ep}(t) \circ u$. By the induction hypothesis, it suffices to show $r = u{\restriction}C'$. We consider two cases.

Case 1: $a \notin \alpha B'$. Then $\langle a \rangle \restriction B' = \langle\,\rangle$ and so $extr_{ep'}(t' \restriction B') = extr_{ep'}(t \restriction B')$. Moreover, since the target channels of the ep_i's are distinct, $u \restriction C' = \langle\,\rangle$. We thus have $r = \langle\,\rangle = u \restriction C'$.

Case 2: $a \in \alpha B'$. Then $\langle a \rangle \restriction B' = \langle a \rangle$ and $extr_{Ep'}(t' \restriction B') = extr_{Ep'}(t \restriction B' \circ \langle a \rangle) = extr_{Ep'}(t \restriction B') \circ u$ (note that $(t \restriction B') \restriction B_i = t \restriction B_i$, for any $B_i \subseteq B'$). Moreover, $u \restriction C' = u$. We thus have $r = u = u \restriction C'$. □

Proof of theorem 2. We will use $\mathcal{X}$ to denote the sources of the extraction pattern EpX, and $\mathcal{Z}$ to denote the channel set of a process Z, for $Z \in \{I, J, K, L, M, N, O, S\}$, where $I \stackrel{\text{df}}{=} K \| L$, $J \stackrel{\text{df}}{=} K \otimes L$, $S \stackrel{\text{df}}{=} M \| N$ and $O \stackrel{\text{df}}{=} M \otimes N$. The union of sets of channels or extraction patterns, such as $\mathcal{C} \cup \mathcal{D}$ or $EpC \cup EpF \cup EpG$, is simply denoted as $\mathcal{CD}$ or CFG, respectively. For a channel z in $\mathcal{K} \cup \mathcal{L}$, we denote by ep_z that extraction pattern which has the target z, its components being indexed by z. Figure 9 may be useful in following the proof details.

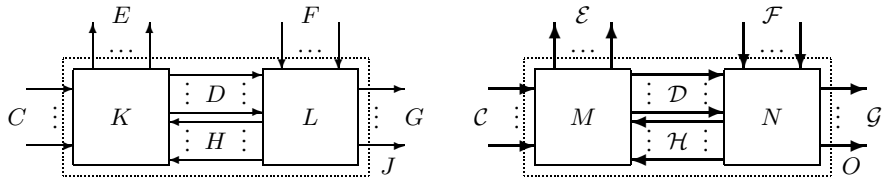


Fig. 9. Processes in the proof of theorem 2; moreover, $I \stackrel{\text{df}}{=} K \| L$ and $S \stackrel{\text{df}}{=} M \| N$

Let $W \stackrel{\text{df}}{=} \{t \in \tau S \mid t \restriction \mathcal{CF} \in Dom_{CF}\}$ and $W' \stackrel{\text{df}}{=} \{t \in \tau O \mid t \restriction \mathcal{CF} \in Dom_{CF}\}$. Note that both W and W' are prefix-closed sets of traces which follows from CSP1 and lemma 11(1). Our next observation is that

$$t \in W \ \wedge\ t \restriction \mathcal{M} \in \tau M \ \wedge\ t \restriction \mathcal{N} \in \tau N \implies t \in Dom_{CDEFGH} \tag{4}$$

which can easily be shown by induction on the length of the prefixes of t, using the prefix-closure of the Dom's and DP for M and N. We now observe that

$$W \cap \delta S = \varnothing\ . \tag{5}$$

For suppose that $W \cap \delta S \neq \varnothing$. Then, by $Pref(W) = W$ and without loss of generality, there is $t \in W \cap \delta S$ such that $t \restriction \mathcal{M} \in \delta M \subseteq \tau M$ and $t \restriction \mathcal{N} \in \tau N$. By (4), $t \in Dom_{CDEFGH}$ which implies that $t \restriction \mathcal{CH} \in Dom_{CH}$. Thus $t \restriction \mathcal{M} \in \delta M$ and $(t \restriction \mathcal{M}) \restriction \mathcal{CH} \in Dom_{CH}$, producing a contradiction with DF for M. Thus (5) holds. We then note that from (4,5) it follows that

$$W \subseteq Dom_{CDEFGH}\ . \tag{6}$$

Suppose now that $W' \cap \delta O \neq \varnothing$. Then, by (5,6), there is an ω-sequence of traces $(t_i)_{i \in \mathbb{N}}$ in $\tau S \cap Dom_{CDEFGH}$ such that $t_i \restriction \mathcal{O} = t_j \restriction \mathcal{O}$, for all $i, j \geq 1$. Let

$w_i \stackrel{\text{df}}{=} extr_{CDEFGH}(t_i)$, for all $i \geq 1$. By TE for M and N and lemma 11(4), $w_i{\restriction}\mathcal{K} \in \tau K$ and $w_i{\restriction}\mathcal{L} \in \tau L$, for all $i \geq 1$. And, by GE for M and N, both $(w_i{\restriction}\mathcal{K})_{i\in\mathbb{N}}$ and $(w_i{\restriction}\mathcal{L})_{i\in\mathbb{N}}$ are ω-sequences of traces satisfying $(w_i{\restriction}\mathcal{K}){\restriction}\mathcal{CE} = (w_j{\restriction}\mathcal{K}){\restriction}\mathcal{CE}$ and $(w_i{\restriction}\mathcal{L}){\restriction}\mathcal{FG} = (w_j{\restriction}\mathcal{L}){\restriction}\mathcal{FG}$, for all $i, j \geq 1$ (which follows from lemma 11(4) and $t_i{\restriction}\mathcal{O} = t_j{\restriction}\mathcal{O}$, for all $i, j \geq 1$). Moreover, since $(w_i{\restriction}\mathcal{K}){\restriction}\mathcal{DH} = (w_j{\restriction}\mathcal{L}){\restriction}\mathcal{DH}$ we obtain that $w_i \in \tau I$, for all $i \geq 1$. Hence $w_1{\restriction}\mathcal{J} \in \delta J$, producing a contradiction with $J = K \otimes L$ being non-diverging. Thus

$$W' \cap \delta O = \varnothing \quad \text{and} \quad W' = W{\restriction}\mathcal{O}\ . \tag{7}$$

We now proceed with the proof proper. That DP, DF and TE hold for O follows from (6,7), lemma 11(4) and since TE holds for M and N.

To show GE suppose that $(t_i)_{i\in\mathbb{N}}$ is an ω-sequence of traces in $\tau O \cap Dom_{CEFG}$. Then, by (6,7), there is a sequence of traces $(w_i)_{i\in\mathbb{N}}$ in $\tau S \cap Dom_{CDEFGH}$ such that $t_i = w_i{\restriction}\mathcal{O}$, for all $i \geq 1$. Thus, by König's Lemma, there is an ω-sequence $(w_{i_j})_{j\in\mathbb{N}}$ such that $i_1 < i_2 < \ldots$ which means, by GE for M and N, that $(extr_{CDEFGH}(w_{i_j}))_{j\in\mathbb{N}}$ is ω-sequence in τI. As a result, by $\delta J = \varnothing$ and lemma 11(4), we obtain that $(extr_{CEFG}(t_{i_j}))_{j\in\mathbb{N}}$ is ω-sequence. Hence, by lemma 11(2), $(extr_{CEFG}(t_i))_{i\in\mathbb{N}}$ is ω-sequence.

To show LC and RE, let $(t, R) \in \phi O$ be such that $t \in Dom_{CEFG}$. By (7) and lemma 10(2) there is $(w, R_M, R_N) \in \Re_{M,N}(t, R)$ such that $w \in Dom_{CDEFGH}$. Thus, it follows directly from LC for M and N that LC holds for O as well. To show that RE also holds, we denote by Id the set of all uninterpreted channels of M and N, and assume additionally that, for every channel $z \in \mathcal{J}$, $w{\restriction}B_z \in dom_z$. We then observe that from $\alpha\mathcal{D} \cup \alpha\mathcal{H} \subseteq R_M \cup R_N$ and EP4 it follows that $\mathcal{D} \cup \mathcal{H} - Id \subseteq Blocked(w{\restriction}\mathcal{M}, R_M) \cup Blocked(w{\restriction}\mathcal{N}, R_N)$. Hence, by LC for M and N as well as the fact that behaviour projected over uninterpreted channels is always complete, it follows that $w{\restriction}B_z \in dom_z$, for all $z \in D \cup H$. We can now use RE for M and N to conclude that

$$(extr_{CDEFGH}(w), B \cup (R_M \cap Id) \cup (R_N \cap Id)) \in \phi I\ ,$$

where $B \stackrel{\text{df}}{=} Blocked(w{\restriction}\mathcal{M}, R_M) \cup Blocked(w{\restriction}\mathcal{N}, R_N)$, and so by $\alpha Id \cap (R_M \cup R_N) \subseteq R_M \cup R_N$, we obtain that

$$(extr_{CEFG}(w{\restriction}\mathcal{J}), (B \cap \alpha J) \cup (R_M \cap \alpha Id \cap \alpha J) \cup (R_N \cap \alpha Id \cap \alpha J)) \in \phi J\ ,$$

from which $(extr_{CEFG}(t), Blocked(t, R) \cup (R \cap \alpha Id)) \in \phi J$ easily follows. □

C Proofs for section 5 (part I)

Proof of proposition 3. As required by the definition of a process, C and D are finite, disjoint sets of channels. Below, we denote $\mathcal{T} \stackrel{\text{df}}{=} \{t \mid (t, R) \in \Phi\}$ and $\mathcal{A} \stackrel{\text{df}}{=} \alpha C \cup \alpha D$.

(1) We will show that P_{CTS} satisfies CSP1–4. To prove that $\mathcal{T}$ is prefix-closed, let $(t, R) \in \Phi$. We consider two cases.

Case 1: $t \notin \Delta$. Then, by definition, there are $v_1, \ldots, v_n \in V$ such that $v_n \in V_{stb}$ and $v_0 \xrightarrow{a_1} v_1 \cdots \xrightarrow{a_n} v_n$ and $t = \langle a_1 \rangle \circ \cdots \circ \langle a_n \rangle$. Let u be a trace such that $u < t$. Then there is $0 \leq i \leq n$ such that $u = \langle a_1 \rangle \circ \cdots \circ \langle a_i \rangle$.

If $v_i \in V_{stb}$ then, by $\varnothing \cap en(v_i) = \varnothing$, $(u, \varnothing) \in \Phi$. In the case that $v_i \notin V_{stb}$, since $t \notin \Delta$ and so no prefix of t may belong to Δ, there is $v' \in V_{stb}$ such that $v_i \xrightarrow{\langle\,\rangle} v'$. And, by $\varnothing \cap en(v') = \varnothing$, $(u, \varnothing) \in \Phi$. Thus, in either case, $u \in \mathcal{T}$.

Case 2: $t \in \Delta$. If there is no prefix of t belonging to Δ, then there is $v \in V$ such that $v_0 \xrightarrow{t} v$. We then follow similar reasoning as for Case 1 above to show that if $u < t$ then $(u, \varnothing) \in \Phi$. If t has a prefix in Δ, let u be the trace such that $u < t$, $u \in \Delta$ and where, for every r such that $r < u$, $r \notin \Delta$. By definition of Δ, for every trace s such that $u \leq s < t$, we have $s \in \Delta$ and so $(s, \varnothing) \in \Phi$. We then follow similar reasoning as in the first part of Case 2, to show that for every y such that $y < u$, $(y, \varnothing) \in \Phi$.

We next show that $\mathcal{T}$ is non-empty. If v_0 is a stable node, then $\varnothing \cap en(v_0) = \varnothing$ and $v_0 \xrightarrow{\langle\,\rangle} v_0$, and so $(\langle\,\rangle, \varnothing) \in \Phi$. If $v_0 \notin V_{stb}$, then either there is $v \in V_{stb}$ such that $v_0 \xrightarrow{\langle\,\rangle} v$, or $\langle\,\rangle \in \Delta$. In the former case, $(\langle\,\rangle, \varnothing) \in \Phi$ by $\varnothing \cap en(v) = \varnothing$. In the latter case, $(\langle\,\rangle, \varnothing) \in \Phi$, by definition of Δ.

To show CSP2, let $(t, R) \in \Phi$ and $S \subseteq R$. Suppose that $t \notin \Delta$. Then there is $v \in V_{stb}$ such that $v_0 \xrightarrow{t} v$ and $R \cap en(v) = \varnothing$. Thus, since $S \subseteq R$, we have $S \cap en(v) = \varnothing$ and so $(t, S) \in \Phi$. Suppose now that $t \in \Delta$. Then $(t, S) \in \Phi$, for every $S \subseteq \mathcal{A}$; in particular, for every $S \subseteq R$.

To prove CSP3, let $(t, R) \in \Phi$, $a \in \mathcal{A}$ and $t \circ \langle a \rangle \notin \mathcal{T}$. We consider two cases.

Case 1: $t \notin \Delta$. Then there is $v \in V_{stb}$ such that $R \cap en(v) = \varnothing$. Since $t \circ \langle a \rangle \notin \mathcal{T}$, we have $a \notin en(v)$, and so $(R \cup \{a\}) \cap en(v) = \varnothing$. Hence $(t, R \cup \{a\}) \in \Phi$.

Case 2: $t \in \Delta$. Then $t \circ \langle a \rangle \in \Delta$ and, by definition of Δ, $(t \circ \langle a \rangle, \varnothing) \in \Phi$ yielding a contradiction with $t \circ \langle a \rangle \notin \mathcal{T}$.

Finally, we show CSP4. If $t \in \Delta$, then $t \circ u \in \Delta$, for all $u \in \mathcal{A}^*$. Moreover, by definition, $(t \circ u, R) \in \Phi$, for every $R \subseteq \mathcal{A}$.

(2) We observe that if $\Delta = \varnothing$ then

$$\Phi = \{(t, R) \in \mathcal{A}^* \times \mathbb{P}(\mathcal{A}) \mid \exists v \in V_{stb} : v_0 \xrightarrow{t} v \ \wedge \ R \cap en(v) = \varnothing\}\ ,$$

and so $\mathcal{T} = \{t \mid \exists v \in V_{stb} : v_0 \xrightarrow{t} v\}$. Moreover, as $\Delta = \varnothing$, for every node $v \in V$ there is a stable node w such that $v \xrightarrow{\langle\,\rangle} w$. Then, by an argument similar to that used in Case 1 when proving CSP1, $\mathcal{T} = \{t \mid v_0 \xrightarrow{t}\}$. □

Proof of proposition 4. EP1 follow directly from the definitions, and EP2 follows from EG1. The strictness of $extr_{EG}$ follows from $v_0 \xrightarrow{\langle\,\rangle : \langle\,\rangle} v_0$, and monotonicity from EG2. Moreover, if $extr_{EG}(u) = t$, then u is a trace in Dom_{EG} and t is a trace over b, by $A \subseteq V \times (\alpha B \times \alpha b^*) \times V$. We then observe that ref_{EG} is defined for all traces in Dom_{EG}. For every $u \in Dom_{EG}$, $ref_{EG}(u)$ is subset-closed by definition; moreover, it is non-empty and contains only proper subsets

of αB, by definition of ϱ. The second part of EP3 follows from EG3. Finally, $Dom_{EG} = Dom_{ep_{EG}}$ follows from EG1. □

D Auxiliary notation

Throughout the rest of this appendix, $q^{(i)} \stackrel{\text{df}}{=} w_i$ for any node $q = (w_0, w_1, \ldots, w_n)$ of CTS^u defined as in algorithm 1 and $0 \leq i \leq m+n$. Two derivations of equal length and the same transition labels,

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} q_k \text{ in } CTS^u \qquad (\dagger)$$
$$v_0 \xrightarrow{a_1} v_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} v_k \text{ in } CTS \qquad (\ddagger)$$

will be called *matching* if $q_i^{(0)} = v_i$, for every $i \leq k$. Moreover, when referring to $(\dagger)$ or $(\ddagger)$, we will denote $\Upsilon \stackrel{\text{df}}{=} \langle a_1 \rangle \circ \cdots \circ \langle a_k \rangle$ and $\Upsilon_i \stackrel{\text{df}}{=} \Upsilon \restriction B_i$, for $1 \leq i \leq m+n$.

Lemma 12. *Assume the notation as in algorithm 1. Moreover, let All be the set of extraction patterns generated by the EG_i's, and let the components of each extraction pattern ep_{EG_i} be indexed by i.*

1. *For every derivation $(\dagger)$ and every $1 \leq i \leq m+n$,*
   (a) *$\Upsilon_i \in Dom_i$ and $q_0^{(i)} = v_{0i} \xrightarrow{\Upsilon_i : extr_i(\Upsilon_i)} q_k^{(i)}$.*
   (b) *$ref_i(\Upsilon_i) = \varrho_i(q_k^{(i)})$.*
   (c) *If $a \in en(q_k) \cap \alpha B_i$, then $extr_i(\Upsilon_i \circ \langle a \rangle) = extr_i(\Upsilon_i) \circ \langle extr(q_k, a) \rangle$.*
2. *For every $(\dagger)$ there is exactly one matching $(\ddagger)$.*
3. *For every $(\ddagger)$ such that $\Upsilon \in Dom_{All}$, there is exactly one matching $(\dagger)$.*
4. *Suppose that CTS^u has not been rejected and $1 \leq i \leq m+n$. If $(\dagger)$ and $(\ddagger)$ are matching derivations, and $b_i \in Idch$ or $i > m$, then $en(q_k) \cap \alpha B_i$ is equal to $en(v_k) \cap \alpha B_i$, and otherwise to $en(v_k) \cap \alpha B_i - \{a \in \alpha B_i \mid \Upsilon_i \circ \langle a \rangle \notin Dom_i\}$.*

Proof. (1) Since parts (b,c) follow from (a), we only show the latter, proceeding by induction on k. In the base case, $k = 0$, we have $q_k = q_0$ and $\Upsilon = \langle\,\rangle$, and so the result holds since $\langle\,\rangle \in Dom_i$ and $extr_i$ is strict. In the induction step, suppose that the result holds for $k-1$, and denote $\Upsilon' \stackrel{\text{df}}{=} \langle a_1 \rangle \circ \cdots \circ \langle a_{k-1} \rangle$ and $\Upsilon'_i \stackrel{\text{df}}{=} \Upsilon' \restriction B_i$. We consider two cases.

Case 1: $a_k \notin \alpha B_i$. Then, by algorithm 1(1,2), $q_k^{(i)} = q_{k-1}^{(i)}$ and $\Upsilon'_i = \Upsilon_i$. Hence the result holds by the induction hypothesis.

Case 2: $a_k \in \alpha B_i$. Then, by algorithm 1(2), $q_{k-1}^{(i)} \xrightarrow{a_k : u} q_k^{(i)}$ for some u, and so, by the induction hypothesis, $\Upsilon_i = \Upsilon'_i \circ \langle a_k \rangle \in Dom_i$. Moreover, by the induction hypothesis,

$$v_{0i} \xrightarrow{\Upsilon'_i : extr_i(\Upsilon'_i)} q_{k-1}^{(i)} ,$$

and, by the definition of $extr_{EG_i}$, $extr_i(\Upsilon_i) = extr_i(\Upsilon'_i \circ \langle a_k \rangle) = extr_i(\Upsilon'_i) \circ u$. Hence, we obtain

$$v_{0i} \xrightarrow{\Upsilon_i : extr_i(\Upsilon_i)} q_k^{(i)} .$$

(2) The existence of (‡) follows by induction on k, from algorithm 1(1,2). The uniqueness of (‡) follows from the definition of matching.

(3) We proceed by induction on k. In the base case, $k = 0$, the result follows immediately. In the inductive step, assume that the result holds for $k-1$, and that $\Upsilon \in Dom_{All}$. We consider two cases.

Case 1: $a_k = \tau$. Then, by algorithm 1(1), $q_{k-1} \xrightarrow{\tau} q_k$ where $q_k^{(0)} = v_k$, for exactly one q_k (note that $q_k^{(i)} = q_{k-1}^{(i)}$, for all $i \geq 1$).

Case 2: $a_k \in \alpha B_i$. From $\Upsilon \in Dom_{All}$ and part (1a) of the lemma, it follows that we can apply algorithm 1(2) for q_{k-1} and a_k. Then we proceed similarly as in Case 1.

(4) Follows from part (1), and the definition of algorithm 1. □

E Proofs for section 5 (part II)

Proof of proposition 5. Follows from lemma 12(1). □

Proof of proposition 6. ($\Longrightarrow$) If CTS^u has been rejected, then (also by lemma 12(2)) there are matching derivations (†) and (‡) such that q_k has been marked as unfinished. Then, by algorithm 1(4) and lemma 12(1a), there is $a \in en(v_k) \cap \alpha D$ such that $\Upsilon \in Dom_{All}$ and $\Upsilon \circ \langle a \rangle \notin Dom_{All}$, where v_k and Υ are as in lemma 12. But this means that Q does not satisfy DP, a contradiction.

To show the second part, suppose that (†) and $l < k$ are such that $q_l = q_k$ and $a_{l+1} = \cdots = a_k = \tau$. By lemma 12(2), there is a matching (‡). Thus, by lemma 12(1a), Q does not satisfy DF, giving a contradiction.

($\Longleftarrow$) Suppose now that Q does not satisfy DF. Then there is a derivation (‡) and $l < k$ such that $\Upsilon \in Dom_{All}$, $v_l = v_k$ and $a_{l+1} = \cdots = a_k = \tau$. Thus, by lemma 12(3), there is a matching derivation (†). Moreover, $q_k = q_l$ due to $a_{l+1} = \cdots = a_k = \tau$ and algorithm 1(1), a contradiction.

If Q does not satisfy DP then, since it satisfies DF, there is a derivation (‡) and $a \in en(v_k) \cap \alpha D$ such that $\Upsilon \in Dom_{All}$ and $\Upsilon \circ \langle a \rangle \notin Dom_{All}$. By lemma 12(3) there is is a matching derivation (†). Moreover, by lemma 12(4), $a \in en(q_k)$, contradicting the first part of lemma 12(1a). □

Lemma 13. $\tau P_{CTS^u} = \tau Q \cap Dom_{All}$.

Proof. Under the assumption that DP and DF hold for Q, and by proposition 6, the result follows from lemma 12(1,2,3) and $Q = P_{CTS_Q}$. □

Proof of proposition 7. Since Q is assumed to satisfy DP and DF, from propositions 6 and 3(2), and lemma 13 it follows that TE for Q is now equivalent to the following:

$$\text{For every derivation (‡), there is } p \text{ satisfying } p_0 \xrightarrow{extr_{All}(\Upsilon)}{}^{*} p\,. \tag{8}$$

Hence, in order to prove the result, it suffices to show that (8) holds *iff* there is a simulation for CTS^u and CTS^n.

($\Longrightarrow$) Consider the following relation:

$$sim \stackrel{\text{df}}{=} \{(q,p) \in V_{CTS^u} \times V_{CTS^n} \mid \exists t:\ q_0 \stackrel{t}{\rightharpoonup} q \wedge p_0 \stackrel{extr_{All}(t)}{\longrightarrow} p\} \ .$$

We will show that sim is a simulation for CTS^u and CTS^n. Clearly, $(q_0, p_0) \in sim$, by the strictness of $extr$. Suppose now that $(q,p) \in sim$ on account of

$$q_0 \stackrel{t}{\rightharpoonup} q \quad \text{and} \quad p_0 \stackrel{extr_{All}(t)}{\longrightarrow} p \ ,$$

and that $q \stackrel{a}{\longrightarrow} q'$. If $extr(q,a) = \tau$ then we clearly have $(q',p) \in sim$, so suppose that $a' = extr(q,a) \neq \tau$. Since (8) holds, by proposition 5 and lemma 12(1c), we know that there is p' in CTS^n such that

$$p_0 \stackrel{extr_{All}(t) \circ \langle a' \rangle}{\longrightarrow} p' \ ,$$

and so $(q',p') \in sim$. Moreover, since CTS^n is τ-free and deterministic, $p \stackrel{a'}{\longrightarrow} p'$.

($\Longleftarrow$) Let sim be a simulation for CTS^u and CTS^n. We proceed by induction on k showing (8) strengthened by adding the condition $(q_k, p) \in sim$.

In the base case, $k = 0$, the property holds. In the inductive step, we assume that the strengthened (8) holds for $k-1$, and denote $\Upsilon' \stackrel{\text{df}}{=} \langle a_1 \rangle \circ \ldots \circ \langle a_{k-1} \rangle$. We consider two cases.

Case 1: $extr(q_{k-1}, a_k) = \tau$, which means that $extr_{All}(\Upsilon) = extr_{All}(\Upsilon')$. Then, by the induction hypothesis, $p_0 \stackrel{extr_{All}(\Upsilon)}{\longrightarrow} p$ and $(q_k, p) \in sim$.

Case 2: $a' = extr(q_{k-1}, a_k) \neq \tau$, which means that $extr_{All}(\Upsilon) = extr_{All}(\Upsilon') \circ \langle a' \rangle$. Then, by the induction hypothesis, there is p' such that $(q_k, p') \in sim$ and $p \stackrel{\langle a' \rangle}{\rightharpoonup} p'$. Hence, by the induction hypothesis, $p_0 \stackrel{extr_{All}(\Upsilon)}{\longrightarrow} p'$. □

Proof of proposition 8. Since Q is assumed to satisfy DP and DF, from propositions 6 and 3(2) and lemma 13, it follows that Q does not satisfy GE *iff* there is an infinite derivation $q_0 \stackrel{a_1}{\longrightarrow} q_1 \stackrel{a_2}{\longrightarrow} q_2 \cdots$ and $k \geq 1$ such that $extr(q_i, a_i) = \tau$, for all $i \geq k$. And, by the finiteness of CTS^u, the latter holds *iff* there is a derivation as in the formulation of proposition 8. □

Lemma 1. *For any two matching derivations (†) and (‡),*

1. *q_k is stable* iff *v_k is stable, and*
2. *if the above holds, then $Blocked(q_k) = Blocked(\Upsilon, R)$, where $R = \alpha C \cup \alpha D - en(v_k)$ is a refusal of Q after trace Υ.*

Proof. The first part follows directly from algorithm 1, and the second from lemma 12(1a,4) and the second part of EP4. □

Proof of proposition 9. We first show that Q satisfies LC *iff* for every stable state q of CTS^u, $b_i \in Blocked(q)$ implies $q^{(i)} \in V_i^\delta$.

($\Longrightarrow$) Suppose that a stable $q = q_k$ is as in (†) and $b_i \in Blocked(q_k)$. By lemmata 12(2) and 1(1), there is a matching derivation (‡) such that v_k is stable. Thus, by lemma 1(2), $b_i \in Blocked(\Upsilon, \alpha C \cup \alpha D - en(v_k))$. Hence, by LC for Q, $\Upsilon_i \in dom_i$. Thus $q_k^{(i)} \in V_i^\delta$, by the definition of dom_{EG_i} and lemma 12(1a).

($\Longleftarrow$) Suppose that $(t, R') \in \phi Q$, where $t \in Dom_{All}$, is such that $b_i \in Blocked(t, R')$. Then, by Q satisfying DF and lemmata 12(3) and 1(1), there are matching derivations (†) and (‡) such that q_k and v_k are stable, $t = \Upsilon$ and $R' \subseteq R = \alpha C \cup \alpha D - en(v_k)$. Hence, by $b_i \in Blocked(t, R') \subseteq Blocked(t, R)$ and lemma 1(2), we have $b_i \in Blocked(q_k)$. Thus $q_k^{(i)} \in V_i^\delta$ and so $t{\restriction}B_i = \Upsilon_i \in dom_i$, by the assumption and lemma 12(1a).

We next show that Q satisfies RE *iff* for every stable state q of CTS^u, it is the case that $(q, p) \in sim_{min}$ and $q^{(i)} \in V_i^\delta$ (for all $i \geq 1$) implies that there is $A \in \kappa(p)$ satisfying

$$(\alpha Blocked(q) \cup (\alpha Idch - en(q))) \cap A = \varnothing \ .$$

($\Longrightarrow$) Since sim_{min} is the minimal simulation for CTS^u and CTS^n, we can assume that there is a derivation (†) such that $q = q_k$ and $p_0 \xrightarrow{extr_{All}(\Upsilon)} p$ (see lemma 12(1c)). By lemmata 12(2) and 1(1), there is a matching (‡) with stable v_k. Moreover, by $q_k^{(i)} \in V_i^\delta$ (for all $i \geq 1$) and lemma 12(1a), $\Upsilon \in dom_{All}$.

Let $R \stackrel{\text{df}}{=} \alpha\alpha C \cup \alpha D - en(v_k)$. We have $(\Upsilon, R) \in \phi Q$, so by RE for Q and lemma 1(2),

$$(extr_{All}(\Upsilon), \alpha Blocked(q_k) \cup (R \cap \alpha Idch)) \in \phi P \ .$$

From the determinism and τ-freeness of CTS^n and lemma 12(4), it then follows that there is $A \in \kappa(p)$ satisfying

$$(\alpha Blocked(q_k) \cup (\alpha Idch - en(q_k))) \cap A = \varnothing \ .$$

($\Longleftarrow$) Suppose that $(t, R') \in \phi Q$, where $t \in dom_{All}$. Then, by Q satisfying DF and lemmata 12(3) and 1(1), there are matching derivations (†) and (‡) such that q_k and v_k are stable, $t = \Upsilon$ and $R' \subseteq R = \alpha C \cup \alpha D - en(v_k)$. Moreover, by lemmata 12(1a) and 1(2), we have $Blocked(q_k) = Blocked(\Upsilon, R)$ and $q_k^{(i)} \in V_i^\delta$ (for all $i \geq 1$).

Let p be such that $(q_k, p) \in sim_{min}$ and $p_0 \xrightarrow{extr_{All}(\Upsilon)} p$. By the assumption, there is $A \in \kappa(p)$ such that $(\alpha Blocked(q_k) \cup (\alpha Idch - en(q_k))) \cap A = \varnothing$. Hence, by lemma 12(4),

$$(extr_{All}(\Upsilon), \alpha Blocked(\Upsilon, R) \cup (R \cap \alpha Idch)) \in \phi P \ ,$$

and so $(extr_{All}(\Upsilon), \alpha Blocked(\Upsilon, R') \cup (R' \cap \alpha Idch)) \in \phi P$. □