Modelling and Verification of Communicating Processes in the Event of Interface Difference

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.
[2] Dipartimento di Matematica e Informatica, Università degli Studi di Catania,
Viale A.Doria 6, I-95125 Catania, Italy

Abstract. We extend our investigation of the notion that a system built of communicating processes is an acceptable implementation of another base or target system, in the case that respective specification and implementation processes have different interfaces and we combine into a single scheme implementation relations previously presented. We also relax significantly the restrictions placed upon target processes. Using this implementation relation scheme, two basic kinds of results are obtained: realisability and compositionality. The former ensures that implementations may be put to good use; in practice, this means that plugging an implementation into an appropriate environment should yield a conventional implementation of the target. The latter requires that a target composed of several connected systems may be implemented by connecting their respective implementations.
We then give graph-based representations of the formal structures which we use, develop graph theoretic statements of the implementation relations and finally present algorithms for their automatic verification.
Keywords: Theory of parallel and distributed computation, behaviour abstraction, communicating sequential processes, compositionality, verification.

1 Introduction

The approach presented in [8–12] aimed at formalising the notion that a system is an acceptable *implementation* of another *base* or *target* system, in the case that the two systems (respective specification and implementation processes) have different interfaces. We developed and expounded the notion of *extraction pattern* to deal with the fact of this interface difference.

Extraction patterns interpret 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 channel or channels in the specification. The set of extraction patterns defined for all channels in an implementation system may act as a formal parameter in a generic *implementation relation* scheme. The scheme given here unifies into a single presentation the notions of *weak* and *strong* implementability defined in [10, 11], with a set of relations of increasing discriminative power. We now also relax the behavioural restrictions on the target processes allowed, thus significantly widening the applicability of our treatment.

The implementation relation scheme given satisfies two light but very natural and useful requirements. The first, *accessibility* or *realisability*, ensures that the abstraction built into the implementation relation may be put to good use; in practice, this means that plugging an implementation into an appropriate environment[1] should yield a con-

[1] In our treatment of NMR [9], this environment is made up of *disturbers*, feeding faulty but sufficiently redundant input, and *extractors*, which interpret the implementation's output.

ventional implementation of the target. *Distributivity* or *compositionality*, the other constraint on the implementation relation, requires it to distribute over system composition; thus, a target composed of two connected systems may be implemented by connecting two of their respective implementations.

As before, both implementation and specification systems are represented using the FD(failure-divergence) model of CSP [3, 6]. To facilitate proceeding to the development of algorithms for automatic verification of the implementation relations, we give representations for extraction patterns and CSP processes, amenable to computer implementation and also formulate the implementation relations in a manner directly related to our chosen means of representation.

The implementation relations as stated imply the use of state space methods for (automatic) verification. This suggests the use of graph-based structures to represent both extraction patterns and CSP processes. It is then a simple enough matter to convert our implementation relations — presented in the denotational semantics of CSP — into graph-theoretic terms, using the equivalences proven between our CSP and graphical presentations of extraction patterns and processes respectively.

That the implementation relations have the property of *compositionality* has an important consequence when we approach automatic verification. It allows us to verify each component of the implementation system explicitly in terms of its specification component and we avoid one of the great sources of the state explosion problem in concurrency, namely the generation of a state space which is a subset (not necessarily proper) of the product of all the state spaces of a set of component processes composed in parallel.

As a result, we avoid in most cases the need to explicitly generate a state space: the state spaces with which we need to deal are mostly implicit in the structure of the *labelled transition systems* with which we represent CSP processes. In those cases where we have to explicitly generate a state space, namely when testing for trace inclusion, only two processes are involved and the testing is done on-the-fly anyway. This leads to generally favourable complexity characteristics. Also of importance in this respect is the fact that we need only deal with *maximal* (in a sense to be defined) failures when verifying the implementation relations.

We present here implementation relations and verification algorithms for the purpose of verifying that a system is a valid implementation of a target system, in the event that the two systems have different interfaces. It may be the case that the target system must be expressed at too low a level of abstraction to function as a conventional specification. In this event, it may be necessary to verify that the target conforms to another (higher level) specification, also expressed in the FD model of CSP. For this purpose, a tool such as FDR [15] may be used, since any interface difference between the target and the specification may be dealt with using the conventional means of renaming and hiding. Though this (higher level) process of verification will not be able to take advantage of the property of compositionality as described in our treatment, the (highest level) specification process will generally be smaller than the target, and the target process will generally be smaller than the implementation process. It is thus clear that our verification methods with their attendant gains in efficiency will be used at those points in the verification process where the systems to be dealt with are of the greatest size.

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. This is followed by the definition of successively stronger implementation relations, and the proofs of their suitability. Section 4 deals with computer representations of CSP processes and extraction patterns, in both cases employing a variant of a labelled transition system. In section 5 we make the

necessary technical steps to relate the labelled transition systems representing processes and extraction patterns, and in sections 6 and 7 we show how the defining conditions for implementation relations can be verified algorithmically.

2 Preliminaries

In this section, we first recall those parts of the CSP theory which are needed throughout the paper. We then introduce a class of base processes.

2.1 Actions and traces

Communicating Sequential Processes (CSP) [2,3,6,15] is 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*. $b!v$ is said to *occur* at b and to cause v be *exchanged* between processes communicating over b. 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. For a set of channels B, $\alpha B = \bigcup_{b \in B} \alpha b$.

The following notation is similar to that of [6] (below $t, u, t_1, t_2, \ldots$ are traces; b, b', b'' are channels; $B_1, \ldots, B_n$, B are disjoint sets of channels; a is an action; A is a set of actions; and T, T' are non-empty sets of traces):

- $t = \langle a_1, \ldots, a_n \rangle$ is the trace whose i-th element is a_i, and whose 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 $t \neq u$.
- $Pref(T) = \{u \mid \exists t \in T : \ u \leq t\}$ if the *prefix-closure* of T; T is *prefix-closed* if $T = Pref(T)$.
- $t[b'/b]$ is a trace obtained from t by replacing each action $b!v$ by $b'!v$.
- $t\lceil B$ is obtained by deleting from t all the actions that do not occur on the channels in B; for example, $\langle b''!3, b!1, b''!2, b!3, b'!3, b''!6, b'!2\rangle\lceil\{b, b'\} = \langle b!1, b!3, b'!3, b'!2\rangle$.
- χa gives the channel on which the event a occurred; for example, $\chi b!1 = b$. Moreover, $\chi A = \{\chi a \mid a \in A\}$.
- An infinite sequence $t_1, t_2, \ldots$ is *ω-monotonic* 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 $f(t) \leq f(u)$; f is *strict* if $\langle\rangle \in T$ and $f(\langle\rangle) = \langle\rangle$; and f is a *homomorphism* if $t, u, t \circ u \in T$ implies $f(t \circ u) = f(t) \circ f(u)$.
- A family of sets $\mathcal{X}$ is *subset-closed* if $Y \subset X \in \mathcal{X}$ implies $Y \in \mathcal{X}$.

2.2 Processes

We use the divergence model of CSP [3,6,15] 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 $\alpha P^* \times 2^{\alpha P}$, and δP — *divergences* — is a subset of αP^*. The conditions imposed on the three components are given below, where $\tau P = \{t \mid (t, R) \in \phi P\}$ denotes the *traces* of P:

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 $R \subseteq \alpha P$.

If $(t, R) \in \phi P$ then P is said to *refuse* R after t. Intuitively, this means that if the environment offers R as a set of possible events to be executed after t, then P can deadlock. If $t \in \delta P$ then P is said to *diverge* after t. In the CSP model this means the process behaves in a totally uncontrollable way. Such a semantical treatment is based on what is often referred to as 'catastrophic' divergence whereby the process in a diverging state is modelled as being able to accept any trace and generate any refusal. We will also consider *maximal* failures defined as those belonging to the set

$$max\phi P = \{(t, R) \in \phi P \mid (t, S) \in \phi P \wedge R \subseteq S \Rightarrow R = S\} \ .$$

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 $(\alpha P, \phi P, \delta P)$.

2.3 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, hiding of the communication over a set of channels and renaming of channels. In the examples we also use deterministic choice, $P \,\Box\, Q$, non-deterministic choice $P \sqcap Q$, and prefixing, $a \rightarrow P$. All these operators are formally defined in the appendix.

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

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 $b'!v$.

A crucial property [3] involving the parallel composition and hiding operators states that if P and Q are two processes then

$$B \subseteq \chi P - \chi Q \Longrightarrow (P \backslash B) \| Q = (P \| Q) \backslash B \ . \tag{1}$$

Its relevance follows from an application to modelling of 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.

Theorem 1. *Let $P_1, \ldots, P_n$ be a network of processes.*

1. $P_1 \otimes P_2 \otimes P_3 \otimes \cdots \otimes P_n = (P_1 \otimes P_2) \otimes P_3 \otimes \cdots \otimes P_n$, *if* $n \geq 3$.
2. $P_1 \otimes \cdots \otimes P_n = P_{i_1} \otimes \cdots \otimes P_{i_n}$, *for any permutation* $i_1, \ldots, i_n$ *of* $1, \ldots, n$.

Proof. (1) Let $B = \chi((P_1 \otimes P_2)\|P_3\| \cdots \|P_n) - \chi((P_1 \otimes P_2) \otimes P_3 \otimes \cdots \otimes P_n)$ and $B' = \chi P_1 \cap \chi P_2$. Then

$$\begin{aligned}(P_1 \otimes P_2) \otimes P_3 \otimes \cdots \otimes P_n &= (((P_1\|P_2)\backslash B')\|P_3\| \cdots \|P_n)\backslash B \\ &= (((P_1\|P_2)\|P_3\| \cdots \|P_n)\backslash B')\backslash B \qquad \text{(by (1))} \\ &= (P_1\|P_2\|P_3\| \cdots \|P_n)\backslash(B \cup B') \\ &= P_1 \otimes \cdots \otimes P_n \ .\end{aligned}$$

(2) Follows immediately from $P_1\| \cdots \|P_n = P_{i_1}\| \cdots \|P_{i_n}$. □

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 [3], whence the need for the more complicated divergence model.

To relate failures of a process network with those of the constituent processes, we will use an auxiliary notation. Let P and Q be two processes forming a network, $Z = P \otimes Q$ and $(t, R) \in \alpha Z^* \times 2^{\alpha Z}$. Then $\Re_{P,Q}(t, R)$ will denote the set of all triples

$$(w, R_P, R_Q) \in (\alpha P \cup \alpha Q)^* \times 2^{\alpha P} \times 2^{\alpha Q}$$

such that $t = w\lceil\chi Z$, $(w\lceil\chi P, R_P) \in \phi P$, $(w\lceil\chi Q, R_Q) \in \phi Q$ and $R_P \cup R_Q = R \cup (\alpha P \cap \alpha Q)$.

Proposition 2 *The following hold.*

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

Proof. Follows directly from the definitions of parallel composition and hiding. □

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).

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

$$\begin{aligned}in\ (P_1 \otimes \cdots \otimes P_n) &= \bigcup_{i=1}^{n} in\ P_i \ - \bigcup_{i=1}^{n} out\ P_i \\ out\ (P_1 \otimes \cdots \otimes P_n) &= \bigcup_{i=1}^{n} out\ P_i - \bigcup_{i=1}^{n} in\ P_i \ .\end{aligned}$$

In the diagrams representing base processes, an outgoing arrow indicates an output channel, and an incoming arrow indicates an input channel. In the diagrams representing implementation processes (other than figure 5), arrowheads are not used.

2.4 Basic processes

A class of base processes considered in [9] was that of *general input/output* processes (*GIO*). These comprise P which: (i) are input-guarded, i.e., for every infinite set T of traces of P, $T\lceil in\ P$ is also infinite; (ii) never refuse any input, i.e., $\phi P \subseteq \alpha P^* \times 2^{\alpha\ out\ P}$;

and (iii) have at least one output channel. In this paper, we relax the restrictions placed on base processes, in the following way. A channel c of a process P is *value independent*, denoted $c \in \mathit{vind}\, P$, if

$$\forall (t, R) \in \phi P : \; c \in \chi R \implies (t, R \cup \alpha c) \in \phi P \; . \tag{2}$$

We then define an *input-output process* to be a non-diverging process P such that $\mathit{in}\, P \subseteq \mathit{vind}\, P$, and denote $P \in IO$.

Since GIO processes never refuse any input, all their input channels are value independent; moreover, GIO processes are divergence-free, and so $GIO \subseteq IO$. The reverse inclusion does not hold. For example, a deterministic buffer of capacity one is an IO but not GIO process.

Intuitively, in an IO process the data component of a message arriving on an input channel c is irrelevant as far as its receiving is concerned; if one such message can be refused then so can any other message. This is not a restrictive property and, in particular, the standard programming receive constructs like $c?x$ give rise to value independent input channels.

Theorem 3. *The class of base IO processes is compositional, i.e., a non-diverging network of IO processes is an IO process.*

Proof. In view of theorem 1, it suffices to show the result for two base processes forming a network.

Let $K, L \in IO$ be as in figure 1 and $\delta(K \otimes L) = \emptyset$. We need to show that $C \cup F \subseteq \mathit{vind}\, K \otimes L$. Without loss of generality, suppose that $c \in C$ and $(t, R) \in \phi(K \otimes L)$ are such that $R \cap \alpha c \neq \emptyset$. Since $K \otimes L$ is non-diverging, by proposition 2(2), there is $(w, R_K, R_L) \in \Re_{K,L}(t, R)$. Now, since $R_K \cap \alpha c \neq \emptyset$ and $K \in IO$, we have $(w \lceil \chi K, R_K \cup \alpha c) \in \phi K$, and so $(w, R_K \cup \alpha c, R_L) \in \Re_{K,L}(t, R \cup \alpha c)$. Hence, by proposition 2(1), $(t, R \cup \alpha c) \in \phi(K \otimes L)$. Thus $c \in \mathit{vind}\, K \otimes L$. □

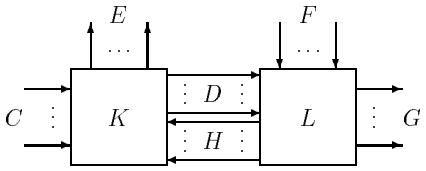


Fig. 1. Processes in the proof of theorem 3

3 Implementation of base processes

In this section, we shall first recall the notion of an extraction pattern. We then define three implementation relations and show that they all satisfy the compsitionality and realisability properties.

3.1 An example

Consider a pair of IO processes, Snd and Buf, shown in figure 2(a). The former generates an infinite sequence of 0s or an infinite sequence of 1s, depending on the value (0 or 1)

received on its input channel, c, at the very beginning of its execution. The latter is a buffer process of capacity one, forwarding values received on its input channel, d. In terms of CSP, we have:

$$Snd = \Box_{i \in \{0,1\}}\, c!i \rightarrow Snd_i$$
$$Buf = \Box_{i \in \{0,1\}}\, d!i \rightarrow B_i$$

where Snd_i and B_i $(i = 0, 1)$ are auxiliary processes defined thus:

$$Snd_i = d!i \rightarrow Snd_i$$
$$B_i \quad = e!i \rightarrow Buf \ .$$

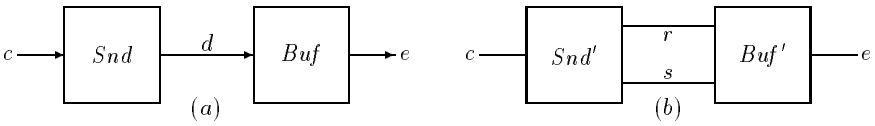


Fig. 2. Two base *IO* processes and their implementations

Suppose now that the communication on d has actually been implemented using two channels, r and s, where r is a data channel, and s is a feedback channel used to pass acknowledgements. 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 is incorporated into suitably modified original processes. The resulting implementation processes shown in figure 2(b), Snd' and Buf', are given by:

$$Snd' = \Box_{i \in \{0,1\}}\, c!i \rightarrow Snd'_i$$
$$Buf' = \Box_{i \in \{0,1\}}\, r!i \rightarrow (s!ack \rightarrow B'_i \sqcap s!nak \rightarrow B)$$

where B, Snd'_i and B'_i $(i = 0, 1)$ are auxiliary processes defined thus:

$$Snd'_i = r!i \rightarrow (s!ack \rightarrow Snd'_i \,\Box\, s!nak \rightarrow r!i \rightarrow Snd'_i)$$
$$B \quad = \Box_{i \in \{0,1\}}\, r!i \rightarrow B'_i$$
$$B'_i \quad = e!i \rightarrow Buf' \ .$$

It may be observed that $Snd' \otimes Buf' = Snd \otimes Buf = Snd[e/d]$. One way of showing this would be to compose the two pairs of processes and prove their equality using, e.g., CSP laws [6]. This would be straightforward for $Snd \otimes Buf$, but less so for $Snd' \otimes Buf'$, at least by hand. Alternatively, the compositional way in which we intend to proceed is to show that Snd' and Buf' are implementations of the respective base processes according to suitable *extraction patterns*, and then derive the desired relationship using general results developed later in this section.

3.2 Extraction patterns

The notion of extraction pattern (introduced in [9–11]) relates behaviour on a set of channels in an implementation process to that on a channel or channels 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*[2] is a tuple $ep = (B, b, dom, extr, ref, inv)$ satisfying the following conditions:

EP0 B is a non-empty set of channels, called *sources* and b is a channel, called *target*.

EP1 dom is a non-empty set of traces over the sources; its prefix-closure will be denoted by Dom.

EP2 $extr$ is a strict monotonic mapping defined for traces in Dom; for every t, $extr(t)$ is a trace over the target.

EP3 ref is a mapping defined for traces in Dom; for every t, $ref(t)$ is a non-empty subset-closed family of subsets of αB such that $\alpha B \notin ref(t)$. It is assumed that if $a \in \alpha B$ and $t \circ \langle a \rangle \notin Dom$ then $R \cup \{a\} \in ref(t)$, for all $R \in ref(t)$.

EP4 inv is a homomorphism from traces over the target to traces in Dom; for every trace w over the target, $extr(inv(w)) = w$.

The mapping $extr$ interprets a trace over the source channels in the implementation process in terms of the interface of the target 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.

The extraction mapping is monotonic as receiving more information cannot decrease the current knowledge about the transmission. $\alpha B \notin ref(t)$ means that for an unfinished communication t we do not allow the sender to refuse all possible transmission. The second condition in EP3 is a rendering of CSP3 in terms of extraction patterns. Note that since inv is a trace homomorphism, it suffices to define it for single actions over the target only.

To demonstrate that Snd' and Buf' are implementations of respectively Snd and Buf, we will need to define two kinds of extraction patterns, id_c and ep_{twice}.

An *identity* extraction pattern for a channel c, id_c, is one for which $B = \{c\}$, $b = c$, $dom = Dom = \alpha c^*$, $extr(t) = inv(t) = t$ and $ref(t) = 2^{\alpha c} - \{\alpha c\}$.

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

$$dom = \{\langle r!0, s!ack\rangle, \langle r!0, s!nak, r!0\rangle, \langle r!1, s!ack\rangle, \langle r!1, s!nak, r!1\rangle\}^*$$

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

$$ref(t \circ u) = \begin{cases} 2^{\alpha r} & \text{if } u = \langle r!v\rangle \\ \{R \in 2^{\alpha r \cup \alpha s} \mid \alpha r \not\subseteq R\} & \text{if } u = \langle\,\rangle \\ \{R \in 2^{\alpha r \cup \alpha s} \mid r!v \notin R\} & \text{if } u = \langle r!v, s!nak\rangle \end{cases}$$

$$inv(d!v) = \{\langle r!v, s!ack\rangle\}\ .$$

The various components of the extraction patterns can be annotated (e.g., subscripted) to avoid ambiguity. Unless explicitly stated, different extraction patterns will have disjoint sources and distinct targets.

[2] What we define here is a *basic* extraction pattern in the terminology of [9]; [11] also allowed sets of target channels.

Sets of extraction patterns For notational convenience, we lift some of the notions to finite sets of extraction patterns. Let $ep = \{ep_1, \ldots, ep_n\}$ be a non-empty set of extraction patterns $ep_i = (B_i, b_i, dom_i, extr_i, ref_i, inv_i)$; moreover, let $B = B_1 \cup \ldots \cup B_n$ and $C = \{b_1, \ldots, b_n\}$. Then:

EP5 $dom_{ep} = \{t \in \alpha B^* \mid \forall i \leq n : t\lceil B_i \in dom_i\}$.
EP6 $Dom_{ep} = \{t \in \alpha B^* \mid \forall i \leq n : t\lceil B_i \in Dom_i\}$.
EP7 $extr_{ep}(\langle\rangle) = \langle\rangle$ and, for every $t \circ \langle a\rangle \in Dom_{ep}$ with $a \in \alpha B_i$,

$$extr_{ep}(t \circ \langle a\rangle) = extr_{ep}(t) \circ u$$

where u is a (possibly empty) trace such that

$$extr_i(t\lceil B_i \circ \langle a\rangle) = extr_i(t\lceil B_i) \circ u \ .$$

EP8 inv_{ep} is a homomorphism from traces over C to traces over B such that $inv_{ep}(a) = inv_i(a)$, for all $i \leq n$ and $a \in \alpha b_i$.

Note that u in EP7 is well defined since $extr_i$ is monotonic, and that inv_{ep} is well defined since $b_i \neq b_j$ for $i \neq j$.

Proposition 4

1. *Dom_{ep} is the prefix-closure of dom_{ep}.*
2. *$extr_{ep}$ is monotonic and strict.*

Proof. (1) To show the prefix-closure of Dom_{ep}, let $t \circ \langle a\rangle \in Dom_{ep}$. Then, for every $i \leq n$, $(t \circ \langle a\rangle)\lceil B_i \in Dom_i$. Since, by EP1, each Dom_i is prefix-closed and $t\lceil B_i \leq (t \circ \langle a\rangle)\lceil B_i$ then, for every $i \leq n$, $t\lceil B_i \in Dom_i$. Hence $t \in Dom_{ep}$.

We next show that, for every $t \in Dom_{ep}$, there is $u \in dom_{ep}$ such that $t \leq u$. Let $t \in Dom_{ep}$. By EP6, for every $i \leq n$, $t\lceil B_i \in Dom_i$. By EP1, for every $i \leq n$, there are $u_i \in dom_i$ and w_i such that $t\lceil B_i \circ w_i = u_i$. It then follows that $u = t \circ w_1 \circ \cdots \circ w_n \in dom_{ep}$ is such that $t \leq u$ and $u\lceil B_i = u_i$, for every $i \leq n$ (note that the B_i's are disjoint sets).

To conclude the proof, we observe that $t \in Dom_{ep}$, for every $t \in dom_{ep}$. Indeed, by EP5, for every $i \leq n$, $t\lceil B_i \in dom_i$ and so $t\lceil B_i \in Dom_i$, meaning that $t \in Dom_{ep}$, by EP6.

(2) $extr_{ep}$ is strict by definition. Moreover, monotonicity follows from $extr_{ep}(t \circ \langle a\rangle) = extr_{ep}(t) \circ u$, in EP7. □

Proposition 5 *Let $B' = B_{i_1} \cup \ldots \cup B_{i_k}$, $C' = \{b_{i_1}, \ldots, b_{i_k}\}$ and $ep' = \{ep_{i_1}, \ldots, ep_{i_k}\}$, for some distinct $i_1, \ldots, i_k$ $(k \geq 1)$.*

1. *$extr_{ep'}(t\lceil B') = extr_{ep}(t)\lceil C'$, for every $t \in Dom_{ep}$.*
2. *$inv_{ep'}(w\lceil C') = inv_{ep}(w)\lceil B'$, for every $w \in C^*$.*

Proof. (1) The proof proceeds by induction on the length of t. In the base case, $t = \langle\rangle$, we have $extr_{ep'}(\langle\rangle\lceil B') = \langle\rangle = extr_{ep}(\langle\rangle)\lceil C'$ since $extr_{ep}$ and $extr_{ep'}$ are strict by proposition 4(2). In the induction step, we consider $t' = t \circ \langle a\rangle$. By EP7,

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

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

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

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

(2) Since, by EP8, inv_{ep} and $inv_{ep'}$ are homomorphisms, for $w = \langle a_1 \circ \cdots \circ a_m \rangle$, we have

$$\begin{aligned} inv_{ep}(w) \lceil B' &= (inv_{ep}(a_1) \circ \cdots \circ inv_{ep}(a_m)) \lceil B' \\ &= inv_{ep}(a_1) \lceil B' \circ \cdots \circ inv_{ep}(a_m) \lceil B' \end{aligned}$$

$$\begin{aligned} inv_{ep'}(w \lceil C') &= inv_{ep'}(\langle a_1 \rangle \lceil C' \circ \cdots \circ \langle a_m \rangle \lceil C') \\ &= inv_{ep'}(\langle a_1 \rangle \lceil C') \circ \cdots \circ inv_{ep'}(\langle a_m \rangle \lceil C') \ . \end{aligned}$$

Thus it suffices to show that $inv_{ep}(a) \lceil B' = inv_{ep'}(\langle a \rangle \lceil C')$, for all $a \in \alpha C$. We first observe that, by EP8, $inv_{ep'}(\langle a \rangle \lceil C') = inv_{ep}(\langle a \rangle \lceil C')$, and then consider two cases.

Case 1: $a \notin \alpha C'$. Then $\langle a \rangle \lceil C' = \langle\rangle$ and so $inv_{ep}(\langle a \rangle \lceil C') = inv_{ep}(\langle\rangle) = \langle\rangle$. Moreover, since the source channels of the ep_i's are disjoint, $inv_{ep}(a) \lceil B' = \langle\rangle$.

Case 2: $a \in \alpha C'$. Then $\langle a \rangle \lceil C' = \langle a \rangle$ and so $inv_{ep}(\langle a \rangle \lceil C') = inv_{ep}(a)$. Moreover, $inv_{ep}(a) \lceil B' = inv_{ep}(a)$. □

3.3 Implementation relations

We are now ready to introduce three implementation relations which will provide a means to relate the base and implementation processes.

Let P be a base IO process as in figure 3 and, for every $i \leq m + n$,

$$ep_i = (B_i, b_i, dom_i, extr_i, ref_i, inv_i)$$

be an extraction pattern. We assume that the B_i's are mutually disjoint channel sets, and denote $ep = \{ep_1, \ldots, ep_m\}$ and $ep' = \{ep_{m+1}, \ldots, ep_{m+n}\}$. We then take a process Q such that $in\,Q = B_1 \cup \ldots \cup B_m$ and $out\,Q = B_{m+1} \cup \ldots \cup B_{m+n}$, as shown in figure 3.

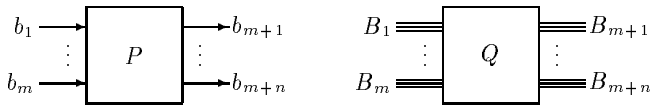


Fig. 3. Base IO process P and its implementation Q

The three implementation relations are defined thus. For $i = 1, 2, 3$, we denote $Q \in Impl_i(P, ep, ep')$ if respectively IR1–IR3, IR1–IR4 and IR1–IR3&IR5 below hold.

IR1 If $t \in \tau Q$ and $t \lceil in\,Q \in Dom_{ep}$ then:
(a) $t \in Dom_{ep \cup ep'}$.
(b) $t \notin \delta Q$.
(c) $extr_{ep \cup ep'}(t) \in \tau P$.

IR2 If $t_1, t_2, \ldots$ is an ω-monotonic sequence of traces in $\tau Q \cap Dom_{ep \cup ep'}$, then the following sequence is also ω-monotonic:

$$extr_{ep \cup ep'}(t_1), extr_{ep \cup ep'}(t_2), \ldots$$

IR3 If $(t,R) \in \phi Q$ and $B = \{b_{i_1}, \ldots, b_{i_k}\} \subseteq \chi P$ are such that $t \in Dom_{ep \cup ep'}$ and

$$b_i \in in\, P \implies \alpha B_i - R \in ref_i(t\lceil B_i)$$
$$b_i \in out\, P \implies \alpha B_i \cap R \notin ref_i(t\lceil B_i)$$

for every $b_i \in B$, then the following are satisfied:
(a) $t\lceil(B_{i_1} \cup \ldots \cup B_{i_k}) \in dom_{\{ep_{i_1}, \ldots, ep_{i_k}\}}$.
(b) $(extr_{ep \cup ep'}(t), \alpha B) \in \phi P$ provided that $t \in dom_{ep \cup ep'}$.

IR4 $inv_{ep \cup ep'}(\tau P) \subseteq \tau Q$.

IR5 If $B \subseteq \chi P$ and $(t, \alpha B) \in \phi P$, then

$$(inv_{ep \cup ep'}(t), \{a \in \bigcup_{b_i \in B} \alpha B_i \mid inv_{ep \cup ep'}(t) \circ \langle a \rangle \in Dom_{ep \cup ep'}\}) \in \phi Q\ .$$

The implementation relations defined above form a hierarchy.

Proposition 6 $Impl_3(P, ep, ep') \subseteq Impl_2(P, ep, ep') \subseteq Impl_1(P, ep, ep')$.

Proof. It suffices to observe that IR5 with $B = \emptyset$ is equivalent to IR4. □

Crucially, all three implementation relations are compositional in the sense that they are preserved by the network composition operation.

Theorem 7. *Let K and L be two base processes whose composition is non-diverging, as in figure 1, and let c, d, e, f, g and h be sets of extraction patterns whose targets are respectively the channel sets C, D, E, F, G and H. Moreover, let $i \in \{1,2,3\}$. If $M \in Impl_i(K, c \cup h, d \cup e)$ and $N \in Impl_i(L, d \cup f, g \cup h)$ then*

$$M \otimes N \in Impl_i(K \otimes L, c \cup f, e \cup g)\ .$$

Proof. In the proof, we use $\mathcal{X}$ to denote the sources, and X to denote the targets, of an extraction pattern set x, for each $x \in \{c,d,e,f,g,h\}$. We also use $\mathcal{Z}$ to denote the channel set of a process Z, for $Z \in \{I,J,K,L,M,N,O,S\}$, where $I = K\|L$, $J = K \otimes L$, $S = M\|N$ and $O = M \otimes N$. The union of sets of channels or extraction patterns, such as $\mathcal{C} \cup \mathcal{D}$ or $c \cup f \cup g$, 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 4 may be useful in following the proof details.

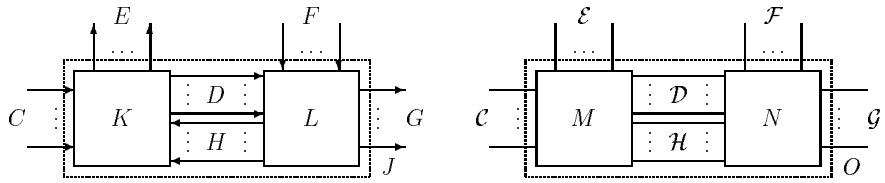


Fig. 4. Processes in the proof of theorem 7; moreover, $I = K\|L$ and $S = M\|N$

Let $W = \{t \in \tau S \mid t\lceil\mathcal{CF} \in Dom_{cf}\}$ and $W' = \{t \in \tau O \mid t\lceil\mathcal{CF} \in Dom_{cf}\}$. Note that both W and W' are prefix-closed sets of traces which follows from CSP1 and proposition 4(1). Our next observation is that

$$t \in W \ \wedge\ t\lceil\mathcal{M} \in \tau M \ \wedge\ t\lceil\mathcal{N} \in \tau N \implies t \in Dom_{cdefgh} \qquad (3)$$

which can be shown by a straightforward induction on the length of the prefixes of t, using the prefix-closure of the Dom's and IR1(a) for M and N. We now observe that

$$W \cap \delta S = \emptyset \ . \tag{4}$$

For suppose that $W \cap \delta S \neq \emptyset$. Then, by $Pref(W) = W$ and without loss of generality, there is $t \in W \cap \delta S$ such that $t\lceil\mathcal{M} \in \delta M \subseteq \tau M$ and $t\lceil\mathcal{N} \in \tau N$. By (3), $t \in Dom_{cdefgh}$ which implies that $t\lceil\mathcal{CH} \in Dom_{ch}$. Thus $t\lceil\mathcal{M} \in \delta M$ and $(t\lceil\mathcal{M})\lceil\mathcal{CH} \in Dom_{ch}$, producing a contradiction with IR1(b) for M. Thus (4) holds. We then note that from (3,4) it follows that

$$W \subseteq Dom_{cdefgh} \ . \tag{5}$$

Suppose now that $W' \cap \delta O \neq \emptyset$. Then, by (4,5), there is an ω-monotonic sequence of traces $t_1, t_2, \ldots$ in $\tau S \cap Dom_{cdefgh}$ such that $t_i\lceil\mathcal{O} = t_j\lceil\mathcal{O}$, for all $i, j \geq 1$. Let $w_i = extr_{cdefgh}(t_i)$ for all $i \geq 1$. By IR1(c) for M and N and proposition 5(1), $w_i\lceil\mathcal{K} \in \tau K$ and $w_i\lceil\mathcal{L} \in \tau L$, for all $i \geq 1$. Moreover, by IR2 for M and N, both $w_1\lceil\mathcal{K}, w_2\lceil\mathcal{K}, \ldots$ and $w_1\lceil\mathcal{L}, w_2\lceil\mathcal{L}, \ldots$ are ω-monotonic sequences of traces satisfying $w_i\lceil\mathcal{K} = w_j\lceil\mathcal{K}$ and $w_i\lceil\mathcal{L} = w_j\lceil\mathcal{L}$, for all $i, j \geq 1$ (which follows from proposition 5(1) and $t_i\lceil\mathcal{O} = t_j\lceil\mathcal{O}$, for all $i, j \geq 1$). Hence $w_1\lceil\mathcal{J} \in \delta J$, producing a contradiction with $J = K \otimes L$ being non-diverging. Thus

$$W' \cap \delta O = \emptyset \quad \text{and} \quad W' = W\lceil\mathcal{O} \ . \tag{6}$$

We now proceed with the proof proper. That IR1 holds for O follows from (5,6), proposition 5(1) and the assumption that IR1(c) holds for M and N.

To show IR2 suppose that $t_1, t_2, \ldots$ is an ω-monotonic sequence of traces in $\tau O \cap Dom_{cefg}$. Then, by (5,6), there is a sequence of traces $w_1, w_2, \ldots$ in $\tau S \cap Dom_{cdefgh}$ such that $t_i = w_i\lceil\mathcal{O}$, for all $i \geq 1$. Thus, by König's Lemma, there is an ω-monotonic sequence $w_{i_1}, w_{i_2}, \ldots$ such that $i_1 < i_2 < \ldots$ which means, by IR2 for M and N, that $extr_{cdefgh}(w_{i_1}), extr_{cdefgh}(w_{i_2}), \ldots$ is an ω-monotonic sequence of traces. Hence, by proposition 5(1), $extr_{cefg}(t_{i_1}), extr_{cefg}(t_{i_2}), \ldots$ is an ω-monotonic sequence of traces. Thus, by proposition 4(2), $extr_{cefg}(t_1), extr_{cefg}(t_2), \ldots$ is ω-monotonic.

To show IR3, let $(t, R) \in \phi O$ be such that $t \in Dom_{cefg}$. Moreover, let $Z_X \subseteq X$, for $X \in \{C, E, F, G\}$, be (possibly empty) sets of channels of base processes such that, for every $z \in Z_E \cup Z_G$, $\alpha B_z \cap R \notin ref_z(t\lceil B_z)$ and, for every $z \in Z_C \cup Z_F$, $\alpha B_z - R \in ref_z(t\lceil B_z)$. By (6) there is $(w, R_M, R_N) \in \Re_{M,N}(t, R)$. Thus, it follows directly from IR3(a) for M and N that IR3(a) holds for O as well. To show that IR3(b) also holds, we assume additionally that, for every channel $z \in \mathcal{J}$, $w\lceil B_z \in dom_z$ and proceed thus.

Let $Z_D \subseteq D$ and $Z_H \subseteq H$ be the sets of all channels z such that: (i) $\alpha B_z \cap R_N \notin ref_z(w\lceil B_z)$, for every $z \in Z_H$; and (ii) $\alpha B_z \cap R_M \notin ref_z(w\lceil B_z)$, for every $z \in Z_D$. We then observe that, from $\alpha\mathcal{D} \cup \alpha\mathcal{H} \subseteq R_M \cup R_N$ and EP3, it follows that: (iii) $\alpha B_z - R_M \in ref_z(w\lceil B_z)$, for every $z \in H - Z_H$; and (iv) $\alpha B_z - R_N \in ref_z(w\lceil B_z)$, for every $z \in D - Z_D$. Thus, by (i-iv) and IR3(a) for M and N it follows that $w\lceil B_z \in dom_z$, for every channel $z \in D \cup H$. We now can use IR3(b) for M and N to conclude that

$$(extr_{cdefgh}(w), \alpha Z) \in \phi I$$

where $Z = Z_C \cup Z_D \cup Z_E \cup Z_F \cup Z_G \cup Z_H \cup (D - Z_D) \cup (H - Z_H)$. Thus

$$(extr_{cdefgh}(w), \alpha Z') \in \phi J$$

where $Z' = Z_C \cup Z_E \cup Z_F \cup Z_G$. This completes the proof for $i = 1$ (i.e., for $Impl_1$).

It is straightforward to show that the result holds also for $i = 2$, using proposition 5(2). To show it holds for $i = 3$, let $Y \subseteq \mathcal{J}$ and $(t, \alpha Y) \in \phi J$. Denote $Y_K = Y \cap \mathcal{K}$ and $Y_L = Y \cap \mathcal{L}$. Since $\delta J = \emptyset$, there is $(w, R_K, R_L) \in \Re_{K,L}(t, \alpha Y)$.

Let $Y_D \subseteq D$ and $Y_H \subseteq H$ be the sets of all channels z such that: (i) $R_L \cap \alpha z \neq \emptyset$, for $z \in D$; and (ii) $R_K \cap \alpha z \neq \emptyset$, for $z \in H$. Then, by $\alpha D \cup \alpha H \subseteq R_K \cup R_L$, we have: (iii) $\alpha Y'_D \subseteq R_K$ where $Y'_D = D - Y_D$; and (iv) $\alpha Y'_H \subseteq R_L$ where $Y'_H = H - Y_H$. Since both K and L are base processes and so their input channels are value independent, $(w\lceil\mathcal{K}, R_K \cup \alpha Y_H) \in \phi K$ and $(w\lceil\mathcal{L}, R_L \cup \alpha Y_D) \in \phi L$. Thus, by IR5 for M and N, as well as by IR1(a) for M and N and CSP3, we have

$$\left(\begin{array}{l} inv_{cdeh}(w\lceil\mathcal{K}), \cup \\ \qquad \begin{array}{l} \{a \in \bigcup_{z \in Y_K \cup Y_H \cup Y'_D} \alpha B_z \mid inv_{cdeh}(w\lceil\mathcal{K}) \circ \langle a \rangle \in Dom_{cdeh}\} \\ \{a \in \bigcup_{z \in D} \alpha B_z \mid inv_{cdeh}(w\lceil\mathcal{K}) \circ \langle a \rangle \notin Dom_{cdeh}\} \end{array} \end{array} \right) \in \phi M$$

$$\left(\begin{array}{l} inv_{dfgh}(w\lceil\mathcal{L}), \cup \\ \qquad \begin{array}{l} \{a \in \bigcup_{z \in Y_L \cup Y_D \cup Y'_H} \alpha B_z \mid inv_{dfgh}(w\lceil\mathcal{L}) \circ \langle a \rangle \in Dom_{dfgh}\} \\ \{a \in \bigcup_{z \in H} \alpha B_z \mid inv_{dfgh}(w\lceil\mathcal{L}) \circ \langle a \rangle \notin Dom_{dfgh}\} \end{array} \end{array} \right) \in \phi N \ .$$

One can then easily see that

$$(inv_{cefg}(t), \{a \in \bigcup_{z \in Y} \alpha B_z \mid inv_{cefg}(t) \circ \langle a \rangle \in Dom_{cefg}\}) \in \phi O$$

which completes the proof of IR5 for O. □

3.4 Realisability relations

In this section, we assume that P is a (specification) base IO process and Q is its implementation which can be used in place of P in an environment T providing inputs and accepting outputs, as shown in figure 5. We will define three implementation relations, of increasing complexity, but also guaranteeing progressively better approximation of the behaviour of the specification process.

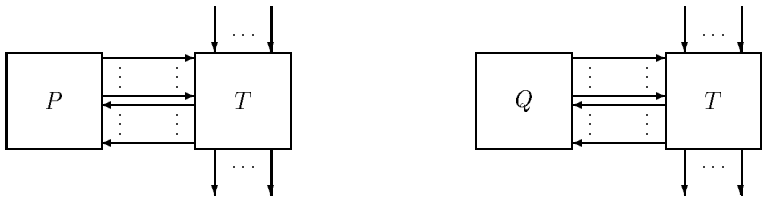


Fig. 5. Relating a base process and its implementation

In the definitions of the three realisability relations, we assume that P, T and Q are non-diverging processes such that $in\,P = in\,Q \subseteq out\,T$ and $out\,P = out\,Q \subseteq in\,T$. Moreover, P and T are IO processes whose composition is non-diverging, i.e., $\delta(P \otimes T) = \emptyset$. The realisability relations are defined thus.

For $i = 1, 2, 3$, we denote $Q \preceq_i P$ if respectively RR1, RR1&RR2 and RR1&RR3 below hold.

RR1 If $C \subseteq in\,P$, $D \subseteq out\,P$ and $(t, R \cup \alpha D) \in \phi Q$ are such that $C \subseteq \chi R$, then $(t, \alpha C \cup \alpha D) \in \phi P$.

RR2 $\tau P \subseteq \tau Q$.
RR3 If $B \subseteq \chi P$ and $(t, \alpha B) \in \phi P$, then $(t, \alpha B) \in \phi Q$.

We then obtain a result characterising the degree to which each of the realisability relations approximates the behaviours of the base processes.

Theorem 8. *Let $k \in \{1,2,3\}$ and $Q \preceq_i P$.*

1. *If $k = 1$ then $\delta(Q \otimes T) = \emptyset$ and $\phi(Q \otimes T) \subseteq \phi(P \otimes T)$ and $\tau(Q \otimes T) \subseteq \tau(P \otimes T)$.*
2. *If $k = 2$ then $\delta(Q \otimes T) = \emptyset$ and $\phi(Q \otimes T) \subseteq \phi(P \otimes T)$ and $\tau(Q \otimes T) = \tau(P \otimes T)$.*
3. *If $k = 3$ then $Q \otimes T = P \otimes T$.*

Proof. (1) We first observe that $\tau Q \subseteq \tau P$ since it suffices to apply RR1 with $C = D = \emptyset$ and $(t, \emptyset) \in \phi Q$. Hence, since $P \otimes T$ and Q are non-diverging processes, $\delta(Q \otimes T) = \emptyset$.

Let $(t, R) \in \phi(Q \otimes T)$. Then, since $\delta(Q \otimes T) = \emptyset$, there is $(w, R_Q, R_T) \in \Re_{Q,T}(t, R)$. Let C be the set of all the channels $c \in in\, Q$ such that $R_Q \cap \alpha c \neq \emptyset$, and D be the set of all the channels $d \in out\, Q$ such that $\alpha d \subseteq R_Q$. Then, by RR1, $(w\lceil \alpha Q, \alpha C \cup \alpha D) \in \phi P$.

We now observe that, for every channel $c \in in\, P - C \subseteq out\, T$, it is the case that $\alpha c \subseteq R_T$ (since $\alpha c \subseteq R_Q \cup R_T$ and $\alpha c \cap R_Q = \emptyset$). Moreover, for every channel $d \in out\, P - D \subseteq in\, T$, it is the case that $R_T \cap \alpha d \neq \emptyset$ (since $\alpha d \subseteq R_Q \cup R_T$ and $\alpha d \not\subseteq R_Q$). Hence, since T is a base process, $(w\lceil \chi T, R_T \cup \alpha(out\, P - D)) \in \phi T$. As a result, $(w, R_T \cup \alpha P) \in \phi(P \| T)$. Hence $(t, R) \in \phi(P \otimes T)$. Thus $\phi(Q \otimes T) \subseteq \phi(P \otimes T)$ and so $\tau(Q \otimes T) \subseteq \tau(P \otimes T)$.

(2) To show $\tau(P \otimes T) \subseteq \tau(Q \otimes T)$, let $t \in \tau(P \otimes T)$. By $\delta(P \otimes T) = \emptyset$, there is $w \in \tau(P \| T)$ such that $t = w\lceil \chi(P \otimes T)$. Hence, by RR2, $w \in \tau(Q \| T)$ and so $t = w\lceil \chi(Q \otimes T) \in \tau(Q \otimes T)$.

(3) In view of part (1), it suffices to show that $\phi(P \otimes T) \subseteq \phi(Q \otimes T)$. Let $(t, R) \in \phi(P \otimes T)$. Then, since $\delta(P \otimes T) = \emptyset$, there is $(w, R_P, R_T) \in \Re_{P,T}(t, R)$.

Let C be the set of all the channels $c \in in\, P$ such that $R_P \cap \alpha c \neq \emptyset$, and D be the set of all the channels $d \in out\, P$ such that $\alpha d \subseteq R_P$. Since P is a base process, we have $(w\lceil \chi P, R_P \cup \alpha C) \in \phi P$. Thus, by RR3, $(w\lceil \alpha P, \alpha C \cup \alpha D) \in \phi Q$. The rest of the proof is similar to the argument made in part (1). □

We have defined three realisabilty relations and demonstrated how they can be used to approximate the behaviour of a base process in an environment provided by another *IO* process. We now will show that these realisability notions correspond to the implementation relations developed in the previous section under the proviso that only identity extraction patterns are involved. Referring to the notation used in section 3.3, we first observe that if ep and ep' are sets of identity extraction patterns, then IR2 is vacuously true and that IR1&IR3–IR5 reduce to the following.

IR1' $\delta Q = \emptyset$ and $\tau Q \subseteq \tau P$.
IR3' If $(t, R) \in \phi Q$ and $B \subseteq \chi P$ are such that

$$b_i \in in\, P \implies \alpha B_i \cap R \neq \emptyset$$
$$b_i \in out\, P \implies \alpha B_i \subseteq R$$

for every $b_i \in B$, then $(t, \alpha B) \in \phi P$.
IR4' $\tau P \subseteq \tau Q$.
IR5' If $B \subseteq \chi P$ and $(t, \alpha B) \in \phi P$, then $(t, \alpha B) \in \phi Q$.

Theorem 9. *Let $i \in \{1,2,3\}$ and ep and ep' be sets of identity extraction patterns. Then $Q \in Impl_i(P, ep, ep')$ if and only if $Q \preceq_i P$.*

Proof. Notice that RR1, RR2 RR3 are nothing but IR3', IR4' and IR5', respectively. Moreover, from RR1 with $C = D = \emptyset$, we obtain $\tau Q \subseteq \tau P$. □

The realisability results can be strengthened after assuming that the base process P is *deterministic* by which we mean that:

D1 If $(t, \{a\}) \in \phi P$ then $t \circ \langle a \rangle \notin \tau P$.
D2 If $t \circ \langle a \rangle, t \circ \langle b \rangle \in \tau P$ and $\chi a = \chi b \in out\, P$, then $a = b$.

Note that D1 is the usual defining condition for deterministic CSP processes [6]. We have added D2 which essentially means that P is not allowed to produce different results on any of its output channels for a given set of input messages.

Proposition 10 *If P is a deterministic IO process and $Q \preceq_3 P$ then $Q = P$.*

Proof. From the definitions of an *IO* process and $\preceq_3$, it follows that $\tau Q = \tau P$ and $\delta Q = \delta P = \emptyset$. Thus, by D1 and CSP3, $\phi P \subseteq \phi Q$. Hence it suffices to show that $\phi P \subseteq \phi Q$. This, in turn, will follow from $\tau Q = \tau P$ and

$$(t, \langle a \rangle) \in \phi Q \implies t \circ \langle a \rangle \notin \tau Q \ . \tag{7}$$

Suppose $(t, \langle a \rangle) \in \phi Q$ and $a \in \alpha b_i$. We consider two cases.

Case 1: $b_i \in in\, Q$. Then, by RR1, $(t, \alpha b_i) \in \phi P$. So, by D1, $t \circ \langle a \rangle \notin \tau P = \tau Q$.

Case 2: $b_i \in out\, Q$. Suppose that $t \circ \langle a \rangle \in \tau Q$. Then, by D2 and $\tau Q = \tau P$, we have $t \circ \langle b \rangle \notin \tau Q$, for all $b \in \alpha b_i - \{a\}$. Hence, by CSP3, $(t, \alpha b_i) \in \phi Q$ and so, by RR1, $(t, \alpha b_i) \in \phi P$. This, however, contradicts D1 and $t \circ \langle a \rangle \in \tau P$. □

3.5 Example revisited

It can be shown that

$$Snd' \in Impl_3(Snd, \{id_c\}, \{ep_{twice}\}) \ \text{ and } \ Buf' \in Impl_3(Buf, \{ep_{twice}\}, \{id_e\}) \ .$$

Hence, by theorems 7 and 9, and $Snd \otimes Buf = Snd[e/d]$, we have:

$$Snd' \otimes Buf' \preceq_3 Snd \otimes Buf = Snd[e/d] \ .$$

Moreover, *Snd* is a deterministic *IO* process and so, by proposition 10,

$$Snd' \otimes Buf' = Snd \otimes Buf = Snd[e/d] \ .$$

4 Representing extraction patterns and CSP processes

Extraction patterns and CSP processes are potentially infinite objects. We therefore need a means to represent them in a finite way in order to allow a computer implementation. To deal with CSP 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.

4.1 Communicating transition systems

In order to represent processes in a manner amenable to computer representation, we take advantage of the *operational semantics* defined for CSP in [15]. This uses a *labelled transition system* (LTS) to represent a CSP process. An LTS may be derived from an algebraic representation of a CSP process using the inference system detailed in [15]. For the purposes of this paper, however, we can simply assume that a process is given in the form of an LTS, without having to worry about how such a representation has been obtained.

We shall represent a process in terms of a *communicating transition system* (CTS), which is essentially an LTS incorporating additional information about channels. A *communicating transition system* is a tuple

$$CTS = (V, C, D, A, v_0)$$

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 $v \stackrel{a}{\longrightarrow} w$.
- If $v_1 \stackrel{a_1}{\longrightarrow} v_2 \stackrel{a_2}{\longrightarrow} \cdots \stackrel{a_n}{\longrightarrow} v_{n+1}$, we denote $v_1 \stackrel{\langle a_1 \rangle \circ \cdots \circ \langle a_n \rangle}{\Longrightarrow} v_{n+1}$ where it is assumed that $\langle \tau \rangle = \langle\rangle$; moreover, $v \stackrel{\langle\rangle}{\Longrightarrow} v$, for every $v \in V$.
- If $v \stackrel{a}{\longrightarrow} w$, we denote $a \in en(v)$ and call a *enabled* at v.
- A state $v \in V$ is *stable* if $\tau \notin en(v)$; the set of stable states will be denoted by V_{stb}.
- If $v \stackrel{t}{\Longrightarrow} w$, we denote $v \Longrightarrow w$ or $v \stackrel{t}{\Longrightarrow}$.

We shall assume that a transition system is finite, i.e., both V and A are finite. Figure 6 shows the graph of a communicating transition system such that $C = \{d\}$ and $D = \{e\}$, where $\mu d = \mu e = \{0, 1\}$. Note that the initial state is indicated by the node with a white centre.

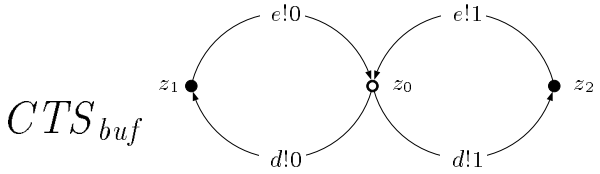


Fig. 6. CTS representing a buffer of capacity one

4.2 Traces and failures information

The implementation relations which we want to verify algorithmically are all expressed in the denotational semantics, and so we must know how to derive information on divergences, traces and failures from a given CTS.

Let $CTS = (V, C, D, A, v_0)$ be a communicating transition system. Then $P_{CTS} = (C, D, \Phi, \Delta)$ is a tuple such that the following hold (below $\alpha CTS = \alpha C \cup \alpha D$):

$$\Delta = \{t \circ u \in \alpha CTS^* \mid \exists k \geq 1\ \exists v_1, \ldots, v_k \in V : v_0 \stackrel{t}{\Longrightarrow} v_1 \stackrel{\tau}{\longrightarrow} v_2 \cdots \stackrel{\tau}{\longrightarrow} v_k \stackrel{\tau}{\longrightarrow} v_1\}$$

$$\Phi = \{(t, R) \in \alpha CTS^* \times 2^{\alpha CTS} \mid \exists v \in V_{stb} : v_0 \stackrel{t}{\Longrightarrow} v \ \wedge\ R \cap en(v) = \emptyset\} \ \cup\ \Delta \times 2^{\alpha CTS}\ .$$

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

It is not difficult to check that the communicating transition system in figure 6 models the buffer of capacity one defined in section 3.1; i.e., $P_{CTS_{buf}} = Buf$.

Proposition 11 *P_{CTS} is a CSP process. Moreover, if $\Delta = \emptyset$, then*

$$\tau P_{CTS} = \{t \in \alpha CTS^* \mid v_0 \stackrel{t}{\Longrightarrow}\}\ .$$

Proof. As required by the definition of a process, C and D are finite, disjoint sets of channels. Below, we denote $\tau P_{CTS} = \{t \mid (t, R) \in \Phi\}$. We now show that P_{CTS} satisfies CSP1–4.

Proving CSP1: We first show that τP_{CTS} 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 $v_0 \stackrel{a_1}{\longrightarrow} v_1 \stackrel{a_2}{\longrightarrow} \cdots \stackrel{a_n}{\longrightarrow} v_n$ 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 is a stable node then R exists (e.g., $R = \emptyset$) such that $R \cap en(v_i) = \emptyset$. In the case that $v_i \notin V_{stb}$, since $t \notin \Delta$ and so no prefix of t may be a divergent trace, it follows that there exists a stable node v' such that $v_i \stackrel{\langle\rangle}{\Longrightarrow} v'$. Again, an R exists such that $R \cap en(v') = \emptyset$. Thus, in either case, there is R such that $(u, R) \in \Phi$ and so $u \in \tau P_{CTS}$.

Case 2: $t \in \Delta$. If there is no prefix of t which is divergent, then there exists a node $v \in V$ such that $v_0 \stackrel{t}{\Longrightarrow} v$. We then follow similar reasoning as for Case 1 above to show that if $u < t$ then there is R such that $(u, R) \in \Phi$. If t has a divergent prefix, 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 $u \leq s < t$, $s \in \Delta$. It therefore follows that $(s, R) \in \Phi$, where $R \in 2^{\alpha CTS}$. We then follow similar reasoning as in the first part of Case 2, to show that for every y such that $y < u$, there exists R such that $(y, R) \in \Phi$.

We next show that τP_{CTS} is non-empty. If v_0 is a stable node, then $\emptyset \cap en(v_0) = \emptyset$ and $v_0 \stackrel{\langle\rangle}{\Longrightarrow} v_0$, and so $(\langle\rangle, \emptyset) \in \Phi$. If $v_0 \notin V_{stb}$, either there is a node $v \in V_{stb}$ such that $v_0 \stackrel{\langle\rangle}{\Longrightarrow} v$, or $\langle\rangle \in \Delta$. In the former case, $\emptyset \cap en(v) = \emptyset$ and, therefore, $(\langle\rangle, \emptyset) \in \Phi$. In the latter case, $(\langle\rangle, R) \in \Phi$, for every $R \in 2^{\alpha CTS}$.

Proving CSP2: Let $(t, R) \in \Phi$ and $S \subseteq R$. We consider two cases.

Case 1: $t \notin \Delta$. Then there exists a stable node v such that $v_0 \stackrel{t}{\Longrightarrow} v$ and $R \cap en(v) = \emptyset$. Thus, since $S \subseteq R$, we have $S \cap en(v) = \emptyset$ and so $(t, S) \in \Phi$.

Case 2: $t \in \Delta$. Then $(t, S) \in \Phi$, for every $S \in 2^{\alpha CTS}$; in particular, for every $S \subseteq R$.

Proving CSP3: Let $(t, R) \in \Phi$ and $a \in \alpha CTS$ be such that $t \circ \langle a \rangle \notin \tau P_{CTS}$. We consider two cases.

Case 1: $t \notin \Delta$. Then there exists a stable node $v \in V$ such that $R \cap en(v) = \emptyset$. Since $\tau \notin en(v)$ and $t \circ \langle a \rangle \notin \tau P_{CTS}$, $a \notin en(v)$ and so $(R \cup \{a\}) \cap en(v) = \emptyset$. It follows that $(t, R \cup \{a\}) \in \Phi$.

Case 2: $t \in \Delta$. Then $t \circ \langle a \rangle \in \Delta$ and, by definition of Δ, there exists R such that $(t \circ \langle a \rangle, R) \in \Phi$. Hence $t \circ \langle a \rangle \in \tau P_{CTS}$, yielding a contradiction with $t \circ \langle a \rangle \notin \tau P_{CTS}$.

Proving CSP4: If $t \in \Delta$, then $t \circ u \in \Delta$, for all $u \in \alpha CTS^*$. Moreover, by definition, $(t \circ u, R) \in \Phi$, for every $R \in 2^{\alpha CTS}$.

We have shown that P_{CTS} meets CSP1–4, i.e., that it is a CSP process. To show the second part of the proposition, we observe that if $\Delta = \emptyset$ then

$$\Phi = \{(t, R) \in \alpha CTS^* \times 2^{\alpha CTS} \mid \exists v \in V_{stb} : v_0 \stackrel{t}{\Longrightarrow} v \ \wedge \ R \cap en(v) = \emptyset\}$$

and so $\tau P_{CTS} = \{t \in \alpha CTS^* \mid \exists v \in V_{stb} : v_0 \stackrel{t}{\Longrightarrow} v\}$. Moreover, since $\Delta = \emptyset$, for every node $v \in V$ there is a stable node v' such that $v \stackrel{\langle\rangle}{\Longrightarrow} v'$. Then, by an argument similar to that used in Case 1 for CSP1, $\tau P_{CTS} = \{t \in \alpha CTS^* \mid v_0 \stackrel{t}{\Longrightarrow}\}$. □

Let Buf, Snd, Buf' and Snd' be processes defined in section 3.1. As already mentioned, $P_{CTS_{buf}} = Buf$ where CTS_{buf} is as in figure 6. Moreover, $P_{CTS_{snd}} = Snd$, $P_{CTS_{buf'}} = Buf'$ and $P_{CTS_{snd'}} = Snd'$, where CTS_{snd}, $CTS_{buf'}$ and $CTS_{snd'}$ are as in figure 7.

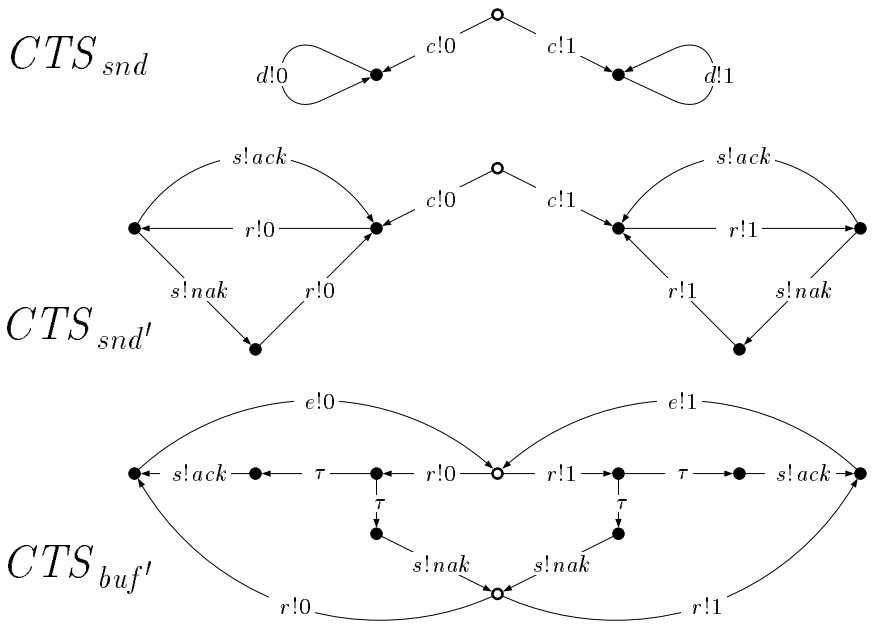


Fig. 7. Communicating transition systems

Calculating failures On the basis of the previous definition, we proceed to elaborate on how failures information may be explicitly derived from a CTS.

The set of *maximal* refusals R, for any particular non-diverging t such that $(t, R) \in \phi P_{CTS}$ may be generated simply by considering the events not on offer at all stable

nodes v reachable from the initial state, $v_0 \stackrel{t}{\Longrightarrow} v$. All other refusals can be derived from the maximal ones.

In order that all sets of nodes representing the same trace may be grouped together, a *normalisation* process is used, as detailed in [15] specifically with respect to the operational semantics of CSP.[3] This produces a CTS with two fundamental properties: (i) there are no τ transitions; and (ii) each node has a unique successor on each visible action it can perform. This normalisation serves two main purposes.

- It creates a deterministic CTS in order that trace inclusion properties may be easily tested for.
- Every node p in the new CTS such that $p_0 \stackrel{t}{\Longrightarrow} p$ is mapped to a set of stable nodes in the original CTS.

The algorithm used for this normalisation process is adapted from [15]. Given a finite $CTS = (V, C, D, A, v_0)$ such that $\Delta = \emptyset$, we form a labelled transition system CTS_{det} whose nodes $V_{CTS_{det}}$ are members of the powerset of V, as follows:

1. The initial node is the set $p_0 = \mathcal{T}(v_0)$ where, for any node $v \in V$, $\mathcal{T}(v) = \{w \in V \mid v \stackrel{\langle\rangle}{\Longrightarrow} w\}$ are the nodes reachable under some sequence of τ's from v.
2. For each node p generated, we determine the set of non-τ actions possible for $v \in p$, that is all actions in $\bigcup_{v \in p} en(v) - \{\tau\}$. For each such action a, we form a new node, $p' = \bigcup\{\mathcal{T}(w) \mid \exists v \in p : v \stackrel{a}{\longrightarrow} w\}$, the set of all nodes reachable after action a and any number of τ's from members of p. We also add a transition $p \stackrel{a}{\longrightarrow} p'$.

We also denote, for every node p of CTS_{det},

$$\kappa_{CTS}(p) = \{en(v) \mid v \in p \ \wedge \ \forall w \in p : en(w) \subseteq en(v) \implies en(w) = en(v)\} \ . \quad (8)$$

In other words, $\kappa_{CTS}(p)$ is the set of minimal sets of actions enabled at stable nodes corresponding to p. Such a set can then be used to calculate the maximal failures.

Proposition 12 *Let CTS be a communicating transition system such that $\Delta = \emptyset$, and CTS_{det} be the determinised version of CTS, with the initial state p_0. Then $(t, R) \in \phi P_{CTS}$ if and only if there exist $p_0 \stackrel{t}{\Longrightarrow} p$ and $A \in \kappa_{CTS}(p)$ such that $R \subseteq \alpha P_{CTS} - A$.*

Proof. Follows from proposition 11, $\Delta = \emptyset$, and the definitions of Φ and κ_{CTS}. □

Corollary 13 *$(t, R) \in max\phi P_{CTS}$ if and only if there exist $p_0 \stackrel{t}{\Longrightarrow} p$ and $A \in \kappa_{CTS}(p)$ such that $R = \alpha P - A$.*

4.3 Testing value independence

We now investigate how one may check whether the input channels of a process represented by a CTS are value independent. We first show that in the definition of value independence (2) it is necessary to consider only maximal failures.

Proposition 14 *The definition of a value independent channel (2) can be replaced by:*

$$\forall (t, R) \in max\phi P : \ c \in \chi R \implies \alpha c \subseteq R \ . \quad (9)$$

[3] We use only the first part of that normalisation process and do not deal with nodes which are bisimilar.

Proof. Let $(t, R') \in \phi P$ and $c \in \chi R'$. Then there is $(t, R) \in max\phi P$ such that $R' \subseteq R$ and $c \in \chi R$. The latter and (9) means that $\alpha c \subseteq R$. Hence, by CSP2, $(t, R' \cup \alpha c) \in \phi P$. □

The rendering of value independence in terms of a communicating transition system CTS is as follows. Let $c \in C \cup D$. We denote $c \in vind\, CTS$ if, for all $p \in V_{CTS_{det}}$ and $A \in \kappa_{CTS}(p)$, if $\alpha c - A \neq \emptyset$ then $\alpha c \cap A = \emptyset$.

Proposition 15 *$vind\, CTS = vind\, P_{CTS}$, provided that $\Delta = \emptyset$.*

Proof. By corollary 13, $(t, R) \in max\phi P_{CTS}$ if and only if there exist $p_0 \stackrel{t}{\Longrightarrow} p$ and $A \in \kappa_{CTS}(p)$ such that $R = \alpha P - A$. Moreover, for $c \in C \cup D$, $c \in \chi R \Leftrightarrow \alpha c - A \neq \emptyset$ and $\alpha c \subseteq R \Leftrightarrow \alpha c \cap A = \emptyset$. Thus the result follows from (9) and the definition of $vind\, CTS$. □

The above considerations give the following algorithm, used to test for the value independence of *input* channels.

Algorithm 1. Let $CTS = (V, C, D, A, v_0)$ be a communicating transition system such that $\Delta = \emptyset$. Moreover, let $C = \{1, \ldots, k\}$ and, for simplicity, $\mu c = \{1, 2, \ldots, l\}$, for every $c \in C$. The pseudo-code of the algorithm is given in figure 8. □

```
// To indicate whether a particular event was offered
bool array [1..k, 1..l] chanEvents

// To indicate the number of events refused on a channel
int array [1..k] counters

function valIndep()
   for every p ∈ V_CTS_det
      for every A ∈ κ_CTS(p)
         counters ← l
         chanEvents ← false
         for every c!e ∈ A
            if ¬chanEvents[c][e]
               then
                  chanEvents[c][e] ← true
                  counters[c] ← counters[c]−1
         for every c ∈ C
            if 0 ≠ counters[c] ≠ l then return failure
   return success
```

Fig. 8. Testing for value independence of input channels

4.4 Extraction graphs

We now turn to the representation of extraction patterns. An *extraction graph* is a tuple

$$EG = (B, b, V, A, v_0, \varrho, \delta, \iota)$$

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 family of proper subsets of αB; $\delta : V \to \{d, D\}$; and $\imath : \alpha b \to \alpha B^*$.
Intuitively, ϱ corresponds to *ref*, d indicates traces in *dom*, D indicates traces in $Dom - dom$, and $\imath$ corresponds to *inv* (see section 3.2). We will use the following notation:

EN1 If $(v, a, t, w) \in A$, we denote $v \xrightarrow{a:t} w$.
EN2 If $v_1 \xrightarrow{a_1:t_1} v_2 \xrightarrow{a_2:t_2} \cdots \xrightarrow{a_n:t_n} v_{n+1}$, we denote $v_1 \xRightarrow{\langle a_1,a_2,\ldots,a_n\rangle:t_1 \circ t_2 \circ \cdots \circ t_n} v_{n+1}$; moreover, $v \xRightarrow{\langle\rangle:\langle\rangle} v$, for every node v.
EN3 If $v \xRightarrow{u:t} w$, we denote $v \Longrightarrow w$.

We impose the following restrictions on an extraction graph EG, where $v \in V$:

EG0 If $R, R' \in \varrho(v)$ and $R \subseteq R'$, then $R = R'$; moreover, if $a \in \alpha B$ and there are no w and t such that $v \xrightarrow{a:t} w$, then $a \in R$.
EG1 $v_0 \Longrightarrow v$.
EG2 If $v \xrightarrow{a:t} w$ and $v \xrightarrow{a:t'} w'$ then $t = t'$ and $w = w'$.
EG3 If $\delta(v) = D$, then there is $w \in V$ such that $v \Longrightarrow w$ and $\delta(w) = d$.
EG4 If $t = \langle a_1, \ldots, a_k\rangle \in \alpha b^*$, then there is w such that $v_0 \xRightarrow{\imath(a_1)\circ\cdots\circ\imath(a_k):t} w$.

When representing an extraction pattern $ep = (B, b, dom, extr, ref, inv)$, it is straightforward to deal with its last component, *inv*, which can be represented by the mapping $\imath$, giving for every $a \in \alpha b$ the trace $inv(a)$. The representation of *dom*, *extr* and *ref* can be provided by other components of an extraction graph.

Let $EG = (B, b, V, A, v_0, \varrho, \delta, \imath)$ be an extraction graph. Then Dom_{EG} is a set of traces, and

$$ep_{EG} = (B, b, dom_{EG}, extr_{EG}, ref_{EG}, inv_{EG})$$

is a tuple, defined in the following way.

EG5 $Dom_{EG} = \{u \in \alpha B^* \mid \exists v, t : v_0 \xRightarrow{u:t} v\}$.
EG6 $dom_{EG} = \{u \in \alpha B^* \mid \exists v, t : v_0 \xRightarrow{u:t} v \wedge \delta(v) = d\}$.
EG7 For every $t = \langle a_1, \ldots, a_k\rangle \in \alpha b^*$, $inv_{EG}(t) = \imath(a_1) \circ \cdots \circ \imath(a_k)$.
EG8 By EG2, for every $u \in Dom_{EG}$, there are *unique* t and v such that $v_0 \xRightarrow{u:t} v$. We then define

$$extr_{EG}(u) = t \quad \text{and} \quad ref_{EG}(u) = \{R \mid \exists R' \in \varrho(v) : R \subseteq R'\} .$$

Proposition 16 *ep_{EG} is an extraction pattern.*

Proof. We take advantage of the intuition that each component of ep_{EG} corresponds to the similarly named component of a generic extraction pattern ep. We proceed to show that each condition EPi, for $0 \le i \le 4$, is met by the relevant component(s) of ep_{EG}.

EP0: Clearly, B is a non-empty finite set of channels and b is a channel.

EP1: By EG5, $dom_{EG} \subseteq \alpha B^*$. Moreover, $dom_{EG} \neq \emptyset$ and $Dom_{EG} = Pref(dom_{EG})$ follow from EG3.

EP2: Strictness of $extr_{EG}$ follows from $v_0 \xRightarrow{\langle\rangle:\langle\rangle} v_0$ in EN2, and monotonicity follows from EN2 and EG8. If $extr_{EG}(u) = t$, then u is a trace in Dom_{EG} and t is a trace over b, by $A \subseteq V \times (\alpha B \times \alpha b^*) \times V$, EN1, EN2 and EG8.

EP3: ref_{EG} is defined for traces in Dom_{EG} by $A \subseteq V \times (\alpha B \times \alpha b^*) \times V$, EN1, EN2 and EG8. Let $u \in Dom_{EG}$. $ref_{EG}(u)$ is subset-closed by EG8. Moreover, $ref_{EG}(u)$ is non-empty and contains only proper subsets of αB, by definition of ϱ and EG8. The second part of EP3 follows from EG8 and the second part of EG0.

EP4: This follows from EG4 and EG7. □

Conversely, one can easily see that for every extraction pattern ep there is an extraction graph EG such that $ep = ep_{EG}$. From the point of view of practical implementation, however, we will be interested only in those extraction graphs which are finite, i.e., have a finite number of nodes V.

Proposition 17 *If V is finite then (V, A) is a finite graph.*

Proof. Follows from V being finite, αB being finite and EG2. □

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

$$|extr(t \circ \langle a \rangle)| \leq 1 + |extr(t)| \ . \tag{10}$$

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

4.5 Examples of extraction graphs

The identity extraction pattern id_c defined in section 3.2 for a channel c with $\mu c = \{0, 1\}$, can be represented by the *identity* extraction graph EG_c, shown in figure 9. The extraction pattern ep_{twice}, also defined in section 3.2, can be represented by the extraction graph EG_{twice}, shown in figure 10. It is easy to check that $ep_{EG_c} = id_c$ and $ep_{EG_{twice}} = ep_{twice}$.

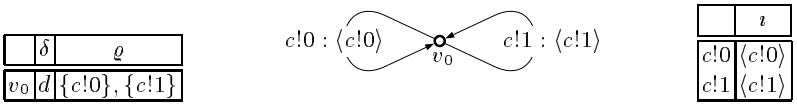
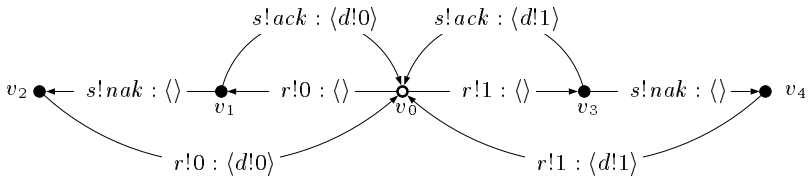


Fig. 9. Extraction graph EG_c

5 Unambiguous CTS

Extraction patterns and so extraction graphs are defined for a channel in a base process P and a channel or 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.

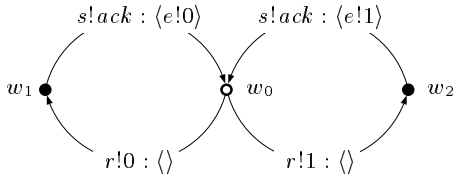


	δ	ϱ
v_0	d	$\{s!ack, s!nak, r!0\}, \{s!ack, s!nak, r!1\}$
v_1	D	$\{r!0, r!1\}$
v_2	D	$\{s!ack, s!nak, r!1\}$
v_3	D	$\{r!0, r!1\}$
v_4	D	$\{s!ack, s!nak, r!0\}$

	$\imath$
$d!0$	$\langle r!0, s!ack \rangle$
$d!1$	$\langle r!1, s!ack \rangle$

Fig. 10. Extraction graph EG_{twice}

We now discuss how to verify trace-based implementation conditions, such as IR1. Let us consider a base process *Buf* modelling a buffer of capacity one, with input channel d and output channel e, defined in section 3.1. Recall that it can be modelled by the communicating transition system CTS_{buf} shown in figure 6. We 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, $EG_1 = EG_d$, is the identity extraction graph defined similarly as in figure 9. The second one, over the sources $\{r, s\}$ and target e, is given in figure 11.



	δ	ϱ
w_0	d	$\{s!ack, r!0\}, \{s!ack, r!1\}$
w_1	D	$\{r!0, r!1\}$
w_2	D	$\{r!0, r!1\}$

	$\imath$
$e!0$	$\langle r!0, s!ack \rangle$
$e!1$	$\langle r!1, s!ack \rangle$

Fig. 11. Extraction graph EG_2

We would like to verify the implementation conditions, with respect to ep_1 and ep_2, for a process Q_0 such that $in\,Q_0 = \{d\}$ and $out\,Q_0 = \{r, s\}$, and whose behaviour is described by the communicating transition system CTS_Q shown in figure 12, i.e., $Q_0 = P_{CTS_Q}$. Although it is not difficult to see that $Q_0 \in Impl_3(Buf, ep_1, ep_2)$, it may not be clear what needs to be done to verify this using only the representations of Q_0, *Buf*, ep_1 and ep_2, given in the form of the appropriate communicating transition systems and extraction graphs. In particular, suppose that we want to verify that IR1(c) holds,

i.e.,

$$extr_{\{ep_1, ep_2\}}(\tau Q_0) \subseteq \tau Buf \ . \tag{11}$$

A possible attempt would be to replace each of the arc annotations in CTS_Q by the 'extracted' string 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_Q is an *ambiguous* representation of Q_0 given the extraction patterns ep_1 and ep_2. A solution we propose is to remove this ambiguity, by suitably modifying CTS_Q.

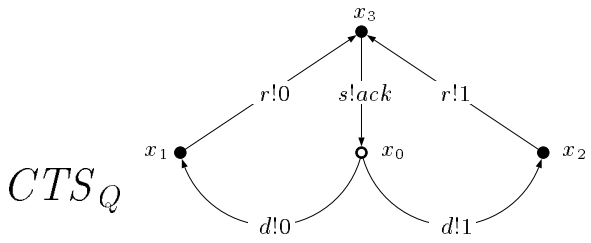


Fig. 12. An implementation of a buffer of capacity one

We split the node x_3 of CTS_Q and separate the two arcs incoming to it, obtaining CTS'_Q shown in figure 13(a). We can now unambiguously interpret each of the arc annotations, which leads to the graph G shown in figure 13(b). To verify that IR1(c) holds, it now suffices to check that the traces generated by G are also generated by CTS_{buf}.

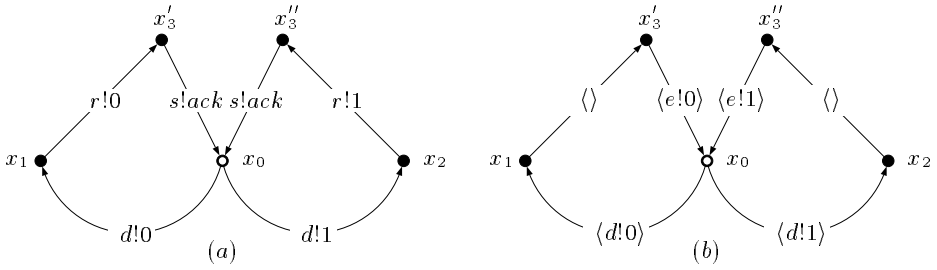


Fig. 13. Disambiguating CTS_Q

The following algorithm generates an equivalent unambiguous CTS, from a given CTS and a set of extraction graphs.

Algorithm 2. For $i = 1, \ldots, m+n$, let

$$EG_i = (B_i, b_i, V_i, A_i, v_{0i}, \varrho_i, \delta_i, \iota_i)$$

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

$$C = B_1 \cup \ldots \cup B_m \quad \text{and} \quad D = B_{m+1} \cup \ldots \cup B_{m+n} \ .$$

The algorithm generates a communicating transition system CTS^u, in two steps.

Step 1: We generate a labelled directed graph G, with the 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; 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 $q \xrightarrow{\tau} (w, v_1, \ldots, v_n)$.
2. $a \neq \tau$ and there is an arc $v_i \xrightarrow{a:t} w_i$ in EG_i, for some $i \geq 1$.[4]
   Then we add a transition $q \xrightarrow{a} (w, w_1, \ldots, w_n)$ where $w_j = v_i$, for all $j \neq i$. Moreover, we denote $extr(q, a) = \tau$ if $t = \langle\rangle$, and $extr(q, a) = b$ if $t = \langle b \rangle$. Note that $extr(q, a)$ is well defined by EG2.
3. $a \in \alpha C$ and there is no arc $v_i \xrightarrow{a:t} w_i$, for any $i \geq 1$. Then we do nothing.
4. $a \in \alpha D$ and there is no arc $v_i \xrightarrow{a:t} w_i$, for any $i \geq 1$. Then we mark permanently q as an *unfinished* node (all the nodes are assumed to be *finished* at the beginning).

Step 2: From the graph G we obtain a communicating transition system CTS^u with the same channels as CTS, by taking $q_0 = (v_0, v_{01}, \ldots, v_{0n})$ as the initial node, and then adding all the nodes reachable from q_0, together with all the interconnecting arcs. If any of the reachable nodes is marked as unfinished, we *reject* CTS^u (since this means that the traces generated by $Q = P_{CTS^u}$ do not satisfy the condition IR1(a), where each ep_i is generated by EG_i, see the proof of proposition 24). □

The above algorithm will be executed on the CTS representation of the *implementation* process Q. The result is denoted CTS^u_Q; its main characteristic is that the definition of the nodes allows the unambiguous interpretation of the arc labels through the extraction mappings (see proposition 19). In addition, $\tau P_{CTS^u_Q} = \tau Q \cap Dom_{ep \cup ep'}$ (see proposition 25).

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. Later, for a node $q = (w_0, w_1, \ldots, w_n)$ of CTS^u and $0 \leq i \leq m + n$, we will denote $q^{(i)} = w_i$.

The graph G for the example in figure 11 is shown in figure 14(a), where the $*$ indicates an unfinished node. After restricting ourselves only to the relevant subgraph (comprising nodes reachable from the initial one), we obtain the graph, shown in figure 14(b), which is isomorphic to CTS'_Q obtained informally before. Note that $extr((x_3, v_0, w_1), s!ack) = e!0$ and $extr((x_1, v_0, w_0), r!0) = \langle\rangle$.

Let $ep_i = (B_i, b_i, dom_i, extr_i, ref_i, inv_i)$ be the extraction pattern generated by EG_i; i.e., $ep_i = ep_{EG_i}$. Moreover, $ep = \{ep_1, \ldots, ep_m\}$ and $ep' = \{ep_{m+1}, \ldots, ep_{m+n}\}$. We now provide some basic properties of CTS^u.

Proposition 18 *If* $q_0 \stackrel{t}{\Longrightarrow} q$ *then* $v_{0i} \stackrel{t\lceil B_i : extr_i(t\lceil B_i)}{\Longrightarrow} q^{(i)}$, *for every* $i \geq 1$.

Proof. We proceed by induction on the length of a path along which t is generated. In the base case, we have

$$q_0 \stackrel{\langle\rangle}{\Longrightarrow} q = q_0 \quad \text{and} \quad v_{0i} \stackrel{\langle\rangle : \langle\rangle}{\Longrightarrow} q^{(i)} = v_{0i} \ .$$

Hence the result holds since $extr_i$ is strict. In the induction step, suppose that

$$q_0 \stackrel{t}{\Longrightarrow} q \xrightarrow{a} q' \quad \text{and} \quad v_{0i} \stackrel{t\lceil B_i : extr_i(t\lceil B_i)}{\Longrightarrow} q^{(i)} \ .$$

[4] There can only be one such EG_i since the B_i's are mutually disjoint.

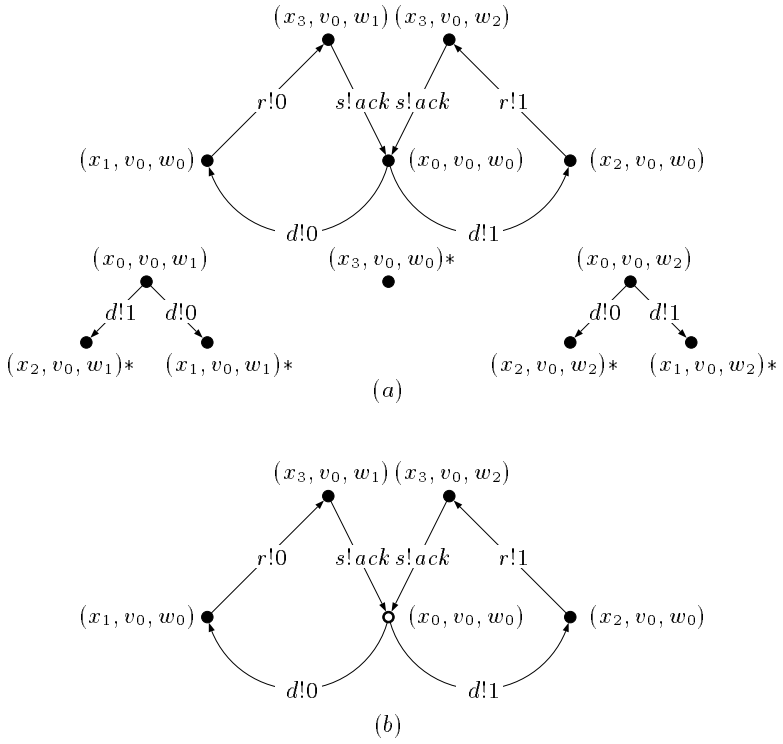


Fig. 14. Applying disambiguating algorithm

We then consider two cases.

Case 1: $a \notin \alpha B_i$. Then, by algorithm 2(1,2), $q'^{(i)} = q^{(i)}$. Moreover, $\langle a \rangle \lceil B_i = \langle\rangle$, and so we have $v_{0i} \stackrel{(t \circ \langle a \rangle) \lceil B_i : extr_i((t \circ \langle a \rangle) \lceil B_i)}{\Longrightarrow} q'^{(i)}$.

Case 2: $a \in \alpha B_i$. Then, by algorithm 2(2), $q^{(i)} \stackrel{a:t'}{\longrightarrow} q'^{(i)}$ and $\langle a \rangle \lceil B_i = \langle a \rangle$. Hence $v_{0i} \stackrel{(t \circ \langle a \rangle) \lceil B_i : extr_i((t \circ \langle a \rangle) \lceil B_i)}{\Longrightarrow} q'^{(i)}$, by $extr_i((t \circ \langle a \rangle) \lceil B_i) = extr_i(t \lceil B_i \circ \langle a \rangle) = extr_i(t \lceil B_i) \circ t'$. □

We may now state precisely why CTS^u can be regarded as an *unambiguous* transition system.

Proposition 19 *Let* $q_0 \stackrel{a_1}{\longrightarrow} q_1 \stackrel{a_2}{\longrightarrow} \cdots \stackrel{a_k}{\longrightarrow} q_k$ *in* CTS^u *and* $t = \langle a_1 \rangle \circ \cdots \circ \langle a_k \rangle$. *Then* $t \in Dom_{ep \cup ep'}$ *and* $extr_{ep \cup ep'}(t) = u_1 \circ \cdots \circ u_k$ *where* $u_i = \langle\rangle$ *if* $a_i = \tau$ *and* $u_i = \langle extr(q_{i-1}, a_i) \rangle$ *if* $a_i \neq \tau$, *for every* $i \leq k$.

Proof. We proceed by induction on k. In the base case, $k = 0$, the result follows immediately. In the induction step, assume that the result holds for k and $q_k \stackrel{a_{k+1}}{\longrightarrow} q_{k+1}$. We then consider two cases.

Case 1: $a_{k+1} = \tau$. Then $t \circ \langle a_{k+1} \rangle = t$ and so the result still holds.

Case 2: $a_{k+1} \in \alpha B_i$. Then, by algorithm 2(2), there is an arc $q_k^{(i)} \stackrel{a_{k+1}:u}{\longrightarrow} q_{k+1}^{(i)}$ in EG_i. By proposition 18, $v_{0i} \stackrel{t \lceil B_i : extr(t \lceil B_i)}{\Longrightarrow} q_k^{(i)}$ and so, by EG5 and EP6, we have $t \circ \langle a_{k+1} \rangle \in Dom_{ep \cup ep'}$. Moreover, $\langle extr(q_k, a_{k+1}) \rangle = u$ and so the second part also holds. □

We can also directly relate various paths in CTS and CTS^u.

Proposition 20 *If* $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} q_k$ *in* CTS^u *then* $q_0^{(0)} \xrightarrow{a_1} q_1^{(0)} \xrightarrow{a_2} \cdots \xrightarrow{a_k} q_k^{(0)}$ *in* CTS.

Proof. Follows by induction on k, directly from algorithm 2(1,2). □

Proposition 21 *If* $v_0 \xrightarrow{a_1} v_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} v_k$ *in* CTS *and* $\langle a_1 \rangle \circ \cdots \circ \langle a_k \rangle \in Dom_{ep \cup ep'}$, *then there is exactly one derivation* $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} q_k$ *in* CTS^u *such that* $q_i^{(0)} = v_i$, *for every* $i \leq k$.

Proof. We proceed by induction on k. In the base case $k = 0$, and the result follows immediately. In the inductive step, assume that the result holds for k, and that $v_k \xrightarrow{a_{k+1}} v_{k+1}$ is such that $\langle a_1 \rangle \circ \cdots \circ \langle a_k \rangle \circ \langle a_{k+1} \rangle \in Dom_{ep \cup ep'}$. We consider two cases.

Case 1: $a_{k+1} = \tau$. Then, by algorithm 2(1), $q_k \xrightarrow{\tau} q_{k+1}$ where $q_{k+1}^{(0)} = v_{k+1}$, for exactly one q_{k+1} (note that $q_{k+1}^{(i)} = q_k^{(i)}$, for $i \geq 1$).

Case 2: $a_{k+1} \in \alpha B_i$. From $\langle a_1 \rangle \circ \cdots \circ \langle a_k \rangle \circ \langle a_{k+1} \rangle \in Dom_{ep \cup ep'}$ and proposition 18, it follows that we can apply algorithm 2(2). Then we proceed similarly as in Case 1. □

Proposition 22 *If* q *is a node in* CTS^u *then*

$$en(q) = en(q^{(0)}) - \{a \in \alpha\, CTS \mid extr(q, a) \text{ is undefined}\}.$$

Moreover, CTS^u *is rejected if and only if there exists a node* q *in* CTS^u *such that* $(en(q^{(0)}) - en(q)) \cap \alpha D \neq \emptyset$.

Proof. The first part follows directly from Step 1 of algorithm 2. The second part follows from the rejection of CTS^u depends on marking at least one node as *unfinished*. The latter is equivalent, by algorithm 2(4), to the existence of a node q in CTS^u such that $(en(q^{(0)}) - en(q)) \cap \alpha D \neq \emptyset$. □

6 Graph representation of implementation relations

In this section, we will transfer the implementation conditions IR1–IR5 formulated in terms of the denotational semantics of CSP, into equivalent conditions expressed in terms of communicating transition systems and extraction graphs. The latter will provide, in section 7, suitable basis for verification algorithms. Below we list some general assumptions which will be used throughout this and the next section.

- P, Q, ep_i (for $i = 1, \ldots, m+n$), ep and ep' are as in section 3.3.
- 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 $i = 1, \ldots, m+n$, EG_i is an extraction graph with the initial node v_{0i} representing ep_i; i.e., $ep_{EG_i} = ep_i$.
- P_{det} is the normalised version of CTS_P. We will use κ_P to denote the mapping defined as in (8) for the nodes of P_{det}, and denote the initial state of P_{det} by p_0.
- CTS_Q^u is a disambiguated version of CTS_Q w.r.t. extraction graphs EG_i (see algorithm 2). We will denote the initial state of CTS_Q^u by q_0^u.
  The process generated by CTS_Q^u will be denoted by $\widehat{Q}$; i.e., $\widehat{Q} = P_{CTS_Q^u}$.
- Q_{det} is the normalised version of CTS_Q^u. We will use κ_Q to denote the mapping defined as in (8) for the nodes of Q_{det}, and denote the initial state of Q_{det} by q_0.
  It may be observed (see proposition 23 below) that if v is a node of Q_{det} and $q, r \in v$ then, for all $1 \leq i \leq m+n$, $q^{(i)} = r^{(i)}$. We can therefore use $v^{(i)}$ to denote $q^{(i)}$, and $extr(v, a) = extr(q, a)$ whenever the latter is defined.

Proposition 23 *If $v \in V_{Q_{det}}$ and $q, r \in v$ then, for all $1 \leq i \leq m+n$, $q^{(i)} = r^{(i)}$*
Note: Thus $extr(v,a)$ is well-defined with respect to Q_{det}.

Proof. To the contrary, suppose that $q^{(i)} \neq r^{(i)}$. Then, by proposition 18, for every derivation $q_0^u \stackrel{t}{\Longrightarrow} q$ (and there is at least one such derivation) it is the case that $v_{0i} \stackrel{t\lceil B_i : extr_i(t\lceil B_i)}{\Longrightarrow} q^{(i)}$ in EG_i. Moreover, again by proposition 18 and EG_i being deterministic and without τ-labels, there is no derivation $q_0^u \stackrel{t}{\Longrightarrow} r$. It then follows, from the definition of normalisation algorithm, that q and r will not be put in the same node of Q_{det}. □

We now proceed with a systematic re-evaluation of the implementation conditions IR1–IR5. We first obtain that testing for IR1(b) amounts to checking for the presence of τ-loops in the graph of CTS_Q^u, and if there is no such loop, then testing for IR1(a) is done while generating CTS_Q^u.

Proposition 24 *Q satisfies IR1(a,b) if and only if CTS_Q^u has been successfully generated and there are no nodes $v_1, \ldots, v_k$ ($k \geq 2$) in CTS_Q^u such that $v_1 \stackrel{\tau}{\longrightarrow} v_2 \stackrel{\tau}{\longrightarrow} \cdots \stackrel{\tau}{\longrightarrow} v_k = v_1$.*

Proof. By $Q = P_{CTS_Q}$ and proposition 11, and also by proceeding similarly as in the proof of proposition 11, one can show that Q satisfies IR1(a,b) if and only if, in CTS_Q, there is neither a derivation:

(i) $r_0 \stackrel{a_1}{\longrightarrow} r_1 \stackrel{a_2}{\longrightarrow} \cdots \stackrel{a_k}{\longrightarrow} r_k \stackrel{\tau}{\longrightarrow} r_{k+1} \stackrel{\tau}{\longrightarrow} \cdots \stackrel{\tau}{\longrightarrow} r_{k+l} = r_k$ $(l \geq 1)$

such that $\langle a_1 \rangle \circ \cdots \circ \langle a_k \rangle \in Dom_{ep \cup ep'}$, nor a derivation:

(ii) $s_0 \stackrel{b_1}{\longrightarrow} s_1 \stackrel{b_2}{\longrightarrow} \cdots \stackrel{b_h}{\longrightarrow} s_h \stackrel{b}{\longrightarrow} s$

such that $\langle b_1 \rangle \circ \cdots \circ \langle b_h \rangle \in Dom_{ep \cup ep'}$ and $\langle b_1 \rangle \circ \cdots \circ \langle b_h \rangle \circ \langle b \rangle \notin Dom_{ep \cup ep'}$ and $\chi b \in out\, Q$.

($\Longrightarrow$) The absence of τ-loops follows from the absence of derivations (i), and propositions 19 and 20. That CTS_Q^u is successfully generated follows from the absence of derivations (ii), and propositions 18, 19, 20 and 22, and finally from algorithm 2(4).

($\Longleftarrow$) By propositions 18, 21 and 22, the existence of a derivation (ii) means that CTS_Q^u is rejected in Step 2 of algorithm 2. Moreover, by propositions 21, the existence of a derivation (i) implies that CTS_Q^u contains a τ-loop. □

From now on, we will assume that CTS_Q^u has been successfully generated and does not contain any τ-loops, and so IR1(a,b) hold. In such a case, as the next result shows, CTS_Q^u generates a process which can be used to test for all implementation relations in place of Q.

Proposition 25 *For each verification condition IR1–IR5 (other than IR1(a,b)), it is the case that Q satisfies the condition if and only if the same is true of $\widehat{Q}$. Moreover, $\tau\widehat{Q} = \tau Q \cap Dom_{ep \cup ep'}$.*
Note: Thus, for $i = 1, 2, 3$, $Q \in Impl_i(P, ep, ep')$ if and only if $\widehat{Q} \in Impl_i(P, ep, ep')$.

Proof. Under the assumption that IR1(a,b) hold and by proposition 24, $\tau\widehat{Q} = \tau Q \cap Dom_{ep \cup ep'}$ follows from propositions 19, 20 and 21. In turn, $\tau\widehat{Q} = \tau Q \cap Dom_{ep \cup ep'}$ implies the result w.r.t. the conditions IR1(c), IR2 and IR4.

There are two other issues which need to be discussed. The first is the satisfaction of the condition related to an input channel b_i in the formulation of IR3. Let v be a node

in CTS_Q reachable by a trace $t \in Dom_{ep \cup ep'}$. By proposition 21, there is a node q in CTS^u_Q reachable by t such that $q^{(0)} = v$ and so, by proposition 22, $en(v) = en(q) \uplus Z$ (where $\uplus$ denotes disjoint union), for some $Z \subseteq \alpha in Q$. Moreover, $t \circ \langle a \rangle \notin Dom_{ep \cup ep'}$, for every $a \in Z$. Hence, by the second part of EP3, if $en(q) \cap \alpha B_i \in ref_i(t\lceil B_i)$, then $en(v) \cap \alpha B_i = (en(q) \cap \alpha B_i) \cup (Z \cap \alpha B_i) \in ref_i(t\lceil B_i)$ as well.

The second issue is related to the formulation IR5. Here, from proposition 21 (the uniqueness of a derivation is important) it follows that if two nodes reachable by a trace $t \in Dom_{ep \cup ep'}$, v in CTS_Q and q in CTS^u_Q, are such that $v = q^{(0)}$, then $a \in en(v) \Leftrightarrow a \in en(q)$, for every $a \in \alpha Q$ satisfying $t \circ \langle a \rangle \in Dom_{ep \cup ep'}$. □

As we have already seen, IR1(a,b) can be checked directly using CTS^u_Q. In dealing with the remaining implementation conditions, we assume that IR1(a,b) hold, and use Q_{det}, which is a normalised CTS derived from CTS^u_Q. Note that, by propositions 24 and 25, we may assume that $\widehat{Q}$ is the implementation process such that

$$\delta\widehat{Q} = \emptyset \quad \text{and} \quad \tau\widehat{Q} \subseteq Dom_{ep \cup ep'} \ . \tag{12}$$

Proposition 26 *$(t, R) \in \phi\widehat{Q}$ if and only if there exist $q_0 \stackrel{t}{\Longrightarrow} q$ and $A \in \kappa_Q(q)$ such that $R \subseteq \alpha Q - A$.*

Proof. Follows from proposition 11, (12), and the definitions of Φ and κ_Q. □

Corollary 27 *$(t, R) \in max\phi\widehat{Q}$ if and only if there exist $q_0 \stackrel{t}{\Longrightarrow} q$ and $A \in \kappa_Q(q)$ such that $R = \alpha Q - A$.*

A relation $sim_{extr} \subseteq V_{Q_{det}} \times V_{P_{det}}$ is an *extr-simulation* for Q_{det} and P_{det} if $(q_0, p_0) \in sim_{extr}$ and, for every $(q, p) \in sim_{extr}$,

$$q \stackrel{a}{\longrightarrow} q' \implies \exists (q', p') \in sim_{extr} : p \stackrel{\langle extr(q,a) \rangle}{\Longrightarrow} p' \ . \tag{13}$$

Proposition 28 *$\widehat{Q}$ (and so Q) satisfies IR1(c) if and only if there exists an extr-simulation for Q_{det} and P_{det}.*

Proof. ($\Longrightarrow$) By propositions 12 and 26, IR1(c) amounts to saying that for every derivation $q_0 \stackrel{t}{\Longrightarrow} q$ in Q_{det} there exists a derivation $p_0 \stackrel{extr_{ep \cup ep'}(t)}{\Longrightarrow} p$ in P_{det}. Let $tr_{incl} \subseteq V_{Q_{det}} \times V_{P_{det}}$ be a relation such that $(q, p) \in tr_{incl}$ if $q_0 \stackrel{t}{\Longrightarrow} q$ and $p_0 \stackrel{extr_{ep \cup ep'}(t)}{\Longrightarrow} p$, for some t. We may now write IR1(c) as: $\forall q \in V_{Q_{det}} \ \exists p \ : \ (q, p) \in tr_{incl}$.

We will show that tr_{incl} is an extr-simulation for Q_{det} and P_{det}. We first note that $(q_0, p_0) \in tr_{incl}$ by the strictness of $extr$. If $q_0 \stackrel{t}{\Longrightarrow} q \stackrel{a}{\longrightarrow} q'$ and $(q, p) \in tr_{incl}$, then $(q', p') \in tr_{incl}$, where p' is such that $p_0 \stackrel{extr_{ep \cup ep'}(t)}{\Longrightarrow} p \stackrel{\langle extr(q,a) \rangle}{\Longrightarrow} p'$, by the monotonicity of $extr$, $extr_{ep \cup ep'}(t \circ \langle a \rangle) = extr_{ep \cup ep'}(t) \circ \langle extr(q, a) \rangle$ and the determinism of P_{det}.

($\Longleftarrow$) Let sim_{extr} be an extr-simulation for Q_{det} and P_{det}. We proceed by induction on the length of traces, showing that if $q_0 \stackrel{t}{\Longrightarrow} q$ then there is a derivation $p_0 \stackrel{extr_{ep \cup ep'}(t)}{\Longrightarrow} p$ such that $(q, p) \in sim_{extr}$.

In the base case $t = \langle\rangle$, so $q_0 \stackrel{\langle\rangle}{\Longrightarrow} q_0$ and $p_0 \stackrel{extr_{ep \cup ep'}(\langle\rangle)}{\Longrightarrow} p_0$ and $(q_0, p_0) \in sim_{extr}$. In the inductive step, we assume $q_0 \stackrel{t}{\Longrightarrow} q \stackrel{a}{\longrightarrow} q'$ and $p_0 \stackrel{extr_{ep \cup ep'}(t)}{\Longrightarrow} p$ and $(q, p) \in sim_{extr}$. Then there is $(q', p') \in sim_{extr}$ such that $p \stackrel{\langle extr(q,a) \rangle}{\Longrightarrow} p'$, and so we have $q_0 \stackrel{t \circ \langle a \rangle}{\Longrightarrow} q'$ and $p_0 \stackrel{extr_{ep \cup ep'}(t \circ \langle a \rangle)}{\Longrightarrow} p'$ by EN2 and the definition of $extr(q, a)$. □

Note that, since P_{det} is deterministic and contains no τ-transitions, if there is at least one extr-simulation, then there exists the smallest one, sim^{min}_{extr}.

From now on, we will additionally assume that $\widehat{Q}$ (and so Q) satisfies IR1(c). Then, testing for the next implementation condition, IR2, amounts to checking for extracted τ-loops in the graph of Q_{det}.

Proposition 29 *$\widehat{Q}$ (and so Q) satisfies IR2 if and only if there are no nodes $v_1, \ldots, v_k$ ($k \geq 2$) in Q_{det} such that $v_1 \stackrel{a_1}{\longrightarrow} v_2 \stackrel{a_2}{\longrightarrow} \cdots \stackrel{a_{k-1}}{\longrightarrow} v_k = v_1$ and $extr(v_i, a_i) = \tau$, for all $i \leq k$.*

Proof. We first observe that, by (12) and the finiteness of CTS^u_Q, we may state that IR2 is met if and only if there are no nodes $w_1, \ldots, w_l$ ($l \geq 2$) in CTS^u_Q such that $w_1 \stackrel{b_1}{\longrightarrow} w_2 \stackrel{b_2}{\longrightarrow} \cdots \stackrel{b_{l-1}}{\longrightarrow} w_k = w_1$ and $extr(w_i, b_i) = \tau$, for all $i < l$. Thus IR2 is met if and only if for every derivation $w_1 \stackrel{b_1}{\longrightarrow} v_2 \stackrel{b_2}{\longrightarrow} \cdots \stackrel{b_{l-1}}{\longrightarrow} w_k = w_1$ ($l \geq 2$) in CTS^u_Q, there exists $b_i \neq \tau$ such that $extr(w_i, b_i) \neq \tau$. The proof then follows from the fact that CTS^u_Q contains no τ-loops, and that the process of normalisation returns a finite result from a finite input, preserves traces, and finally from the finiteness of CTS^u_Q (note that if a finite CTS generates an infinite trace $t \circ w \circ w \circ \cdots$ then there is a cycle in CTS generating a trace of the form $w \circ \cdots w$). □

To prepare the ground for testing of IR3, we re-phrase it in terms of maximal failures of $\widehat{Q}$. For every $(t, R) \in \phi\widehat{Q}$, we denote

$$\mathcal{C}_{t,R} = \{b_i \in in\,P \mid \alpha B_i - R \in ref_i(t\lceil B_i)\} \;\cup\; \{b_i \in out\,P \mid \alpha B_i \cap R \notin ref_i(t\lceil B_i)\}\ .$$

We also denote $\mathcal{C}_t = \{\mathcal{C}_{t,R} \mid (t, R) \in max\phi\widehat{Q}\}$ and $\overline{\mathcal{C}}_t = \bigcup\{B \mid B \in \mathcal{C}_t\}$.

Proposition 30 *$\widehat{Q}$ (and so Q) satisfies IR3 if and only if, for every $t \in \tau\widehat{Q}$, the following hold.*

1. *If $b_i \in \overline{\mathcal{C}}_t$ then $t\lceil B_i \in dom_i$.*
2. *If $t \in dom_{ep \cup ep'}$ then for every $B \in \mathcal{C}_t$ there exists R such that $(extr_{ep \cup ep'}(t), R) \in max\phi P$ and $\alpha B \subseteq R$.*

Proof. ($\Longleftarrow$) We first observe that, by EP3, if $R' \subseteq R$ then $\mathcal{C}_{t,R'} \subseteq \mathcal{C}_{t,R}$. Hence IR3(a) follows from (1) above and EP5. Moreover, IR3(b) follows from (2) above and CSP2 for P.

($\Longrightarrow$) The proof follows by EP5 and the fact that (1) and (2) above deal with a special case of the original condition. □

We now introduce notions corresponding to $\mathcal{C}_{t,R}$, $\mathcal{C}_t$ and $\overline{\mathcal{C}}_t$ in the domain of communicating transition systems and extraction graphs. For all $q \in V_{Q_{det}}$ and $A \in \kappa_Q(q)$, we denote

$$\begin{aligned}\mathcal{C}_{q,A} = \{b_i \in in\,P \;\; &\mid \exists R' \in \varrho_i(q^{(i)}) : \alpha B_i \cap A \subseteq R'\} \cup \\ \{b_i \in out\,P \mid &\nexists R' \in \varrho_i(q^{(i)}) : \alpha B_i - A \subseteq R'\}\ .\end{aligned}$$

We also denote $\mathcal{C}_q = \{\mathcal{C}_{q,A} \mid A \in \kappa_Q(q)\}$ and $\overline{\mathcal{C}}_q = \bigcup\{B \mid B \in \mathcal{C}_q\}$.

Proposition 31 *If $q_0 \stackrel{t}{\Longrightarrow} q$ then $\mathcal{C}_q = \mathcal{C}_t$. Moreover,*

1. *$\mathcal{C}_{q,A} = \mathcal{C}_{t,\alpha Q - A}$ for every $A \in \kappa_Q(q)$.*
2. *$\mathcal{C}_{t,R} = \mathcal{C}_{q,\alpha Q - R}$ for every $(t, R) \in max\phi\widehat{Q}$.*

Proof. The first part follows immediately from (1) and (2), so we only show these.

(1) Let $A \in \kappa_Q(q)$. If $R' \in \varrho_i(q^{(i)})$ is such that $\alpha B_i \cap A \subseteq R'$ then by EG8 and propositions 16 and 18, we have $\alpha B_i \cap A \in ref_i(t\lceil B_i)$. By corollary 27, $(t,R) \in max\phi\widehat{Q}$ where $R = \alpha Q - A$. For such an R, we have $\alpha B_i - R \in ref_i(t\lceil B_i)$.

By similar reasoning, we infer that if there does not exist $R' \in \varrho_i(q^{(i)})$ such that $\alpha B_i - A \subseteq R'$ then $\alpha B_i \cap R \in ref_i(t\lceil B_i)$, where $(t,R) \in max\phi\widehat{Q}$ and $R = \alpha Q - A$.

(2) Let $(t,R) \in max\phi\widehat{Q}$. By corollary 27, there exists $A \in \kappa_Q(q)$ such that $R = \alpha Q - A$. Thus, by EG8 and proposition 18, $\alpha B_i - R \in ref_i(t\lceil B_i)$ implies $\alpha B_i \cap A \subseteq R'$ where $R' \in \varrho_i(q^{(i)})$. By similar reasoning, $\alpha B_i \cap R \notin ref_i(t\lceil B_i)$ implies that there does not exist an R' such that $R' \in \varrho_i(q^{(i)})$ and $\alpha B_i - A \subseteq R'$. □

Proposition 32 *$\widehat{Q}$ (and so Q) satisfies IR3 if and only if, for every $q \in V_{Q_{det}}$, the following hold.*

1. *If $b_i \in \overline{\mathcal{C}}_q$ then $\delta_i(q^{(i)}) = d$.*
2. *If $(q,p) \in sim^{min}_{extr}$ and $\delta_i(q^{(1)}) = \cdots = \delta_i(q^{(m+n)}) = d$ then, for every $B \in \mathcal{C}_q$, there is $A \in \kappa_P(p)$ satisfying $\alpha B \cap A = \emptyset$.*

Proof. ($\Longrightarrow$) Let $q \in V_{Q_{det}}$. By proposition 26, $q_0 \stackrel{t}{\Longrightarrow} q$, for some $t \in \tau\widehat{Q}$.

(1) By proposition 31, $\mathcal{C}_q = \mathcal{C}_t$ and so $\overline{\mathcal{C}}_q = \overline{\mathcal{C}}_t$. The proof follows by EG6 and propositions 18 and 30(1).

(2) Again by proposition 31, $\mathcal{C}_q = \mathcal{C}_t$. By propositions 16 and 18, EG6, and EP5, we have that $\delta_i(q^{(i)}) = d$ (for $1 \leq i \leq m+n$) implies $t \in dom_{ep \cup ep'}$. By proposition 28 and the fact that sim^{min}_{extr} is the smallest extr-simulation, $(q,p) \in sim^{min}_{extr}$ implies $p_0 \stackrel{extr_{ep\cup ep'}(t)}{\Longrightarrow} p$. Then we apply corollary 13 and proposition 30(2).

($\Longleftarrow$) It suffices to show that the two conditions in proposition 30 are satisfied. Let $t \in \tau\widehat{Q}$. By proposition 26, there exists a derivation $q_0 \stackrel{t}{\Longrightarrow} q$. Then proposition 30(1) follows from (1) above, proposition 18, $\overline{\mathcal{C}}_q = \overline{\mathcal{C}}_t$ (follows from proposition 31), and EG6.

Proposition 30(2) can be shown thus. By EP5, EG6 and proposition 18, we have that $t \in dom_{ep\cup ep'}$ implies $\delta_i(q^{(i)}) = d$ (for $1 \leq i \leq m+n$). By corollary 13 and proposition 28, there exist $p \in V_{Pdet}$ such that $p_0 \stackrel{extr_{ep\cup ep'}(t)}{\Longrightarrow} p$ and $(q,p) \in sim^{min}_{extr}$. Let $B \in \mathcal{C}_t$. By $\mathcal{C}_q = \mathcal{C}_t$, there is $A \in \kappa_P(p)$ such that $\alpha B \cap A = \emptyset$. Thus, by corollary 13, $(extr_{ep\cup ep'}(t), R) \in max\phi P$ and $\alpha B \subseteq R$, where $R = \alpha P - A$. □

We now turn to the two remaining implementation conditions. Since the inv_i's are homomorphisms, they can interpret the arc labels directly, without taking into account how a particular node has been reached. However, the situation is complicated by the fact that $inv_i(a)$ will usually be a non-singleton trace.

A relation $sim_{inv} \subseteq V_{P_{det}} \times V_{Q_{det}}$ is an *inv-simulation* for P_{det} and Q_{det} if $(p_0,q_0) \in sim_{inv}$ and, for every pair $(p,q) \in sim_{inv}$,

$$p \stackrel{a}{\longrightarrow} p' \implies \exists (p',q') \in sim_{inv} : q \stackrel{inv_{ep\cup ep'}(a)}{\Longrightarrow} q' \,. \tag{14}$$

Proposition 33 *$\widehat{Q}$ (and so Q) satisfies IR4 if and only if there exists an inv-simulation.*

Proof. IR4 holds if and only if $inv_{ep\cup ep'}(\tau P) \subseteq \tau\widehat{Q}$. Thus, by preservation of traces by the process of normalisation, we may work with Q_{det} and P_{det}.

($\Longrightarrow$) By propositions 12 and 26, we may write IR4 as $p_0 \stackrel{t}{\Longrightarrow} p$ implies $q_0 \stackrel{inv_{ep\cup ep'}(t)}{\Longrightarrow} q$, for some $q \in V_{Q_{det}}$. Let $tr_{incl} \subseteq V_{P_{det}} \times V_{Q_{det}}$ be a relation such that $(p,q) \in tr_{incl}$

if $p_0 \stackrel{t}{\Longrightarrow} p$ and $q_0 \stackrel{inv_{ep\cup ep'}(t)}{\Longrightarrow} q$, for some $t \in \tau P$. We will show that tr_{incl} is an inv-simulation.

We first note that $(p_0, q_0) \in tr_{incl}$ since $inv(\langle\rangle) = \langle\rangle$. Suppose now that $p_0 \stackrel{t}{\Longrightarrow} p \stackrel{a}{\longrightarrow} p'$ and $q_0 \stackrel{inv_{ep\cup ep'}(t)}{\Longrightarrow} q$ and $(p, q) \in tr_{incl}$. Then, since IR4 is met, inv is a homomorphism and by the determinism of Q_{det}, there is q' such that $q_0 \stackrel{inv_{ep\cup ep'}(t)}{\Longrightarrow} q \stackrel{inv(a)}{\Longrightarrow} q'$. Thus $(p', q') \in tr_{incl}$.

($\Longleftarrow$) Let sim_{inv} be an inv-simulation for P_{det} and Q_{det}. We proceed by induction on the length of traces, showing that if $p_0 \stackrel{t}{\Longrightarrow} p$ then there is a derivation $q_0 \stackrel{inv_{ep\cup ep'}(t)}{\Longrightarrow} q$ such that $(p, q) \in sim_{inv}$.

In the base case $t = \langle\rangle$, and so $p_0 \stackrel{\langle\rangle}{\Longrightarrow} p_0$ and $q_0 \stackrel{inv_{ep\cup ep'}(\langle\rangle)}{\Longrightarrow} q_0$ and $(p_0, q_0) \in sim_{inv}$. In the inductive step, we assume that $p_0 \stackrel{t}{\Longrightarrow} p \stackrel{a}{\longrightarrow} p'$ and $q_0 \stackrel{inv_{ep\cup ep'}(t)}{\Longrightarrow} q$ and $(p, q) \in sim_{inv}$. Then there is $(p', q') \in sim_{inv}$ such $q \stackrel{inv_{ep\cup ep'}(a)}{\Longrightarrow} q'$, and so we have $p_0 \stackrel{t\circ\langle a\rangle}{\Longrightarrow} p'$ and $q_0 \stackrel{inv_{ep\cup ep'}(t\circ\langle a\rangle)}{\Longrightarrow} q'$ since inv is a homomorphism. □

In the last part of this section dealing with IR5, we will assume that $\widehat{Q}$ (and so Q) satisfies IR4. This does not result in a loss of generality as IR4 is implied by IR5. Note that, since Q_{det} is deterministic and contains no τ-transitions, if there is at least one inv-simulation, then there exists the smallest one, sim^{min}_{inv}.

To test for IR5, we first observe that it can be equivalently expressed in terms of maximal failures. For every $(t, R) \in \phi P$, we denote $\mathcal{D}_{t,R} = \{b_i \in \chi P \mid \alpha b_i \subseteq R\}$; moreover, $\mathcal{D}_t = \{\mathcal{D}_{t,R} \mid (t, R) \in max\phi P\}$.

Proposition 34 *$\widehat{Q}$ (and so Q) satisfies IR5 if and only if, for every $t \in \tau P$ and $B \in \mathcal{D}_t$, there is $(inv_{ep\cup ep'}(t), R) \in max\phi\widehat{Q}$ such that $\bigcup_{b_i\in B} \alpha B_i \subseteq R$.*

Proof. ($\Longleftarrow$) Let $B' \subseteq \chi P$ and $(t, \alpha B') \in \phi P$. Then there is $(t, R') \in max\phi P$ such that $B' \subseteq B = \mathcal{D}_{t,R'}$. Hence there is $(inv_{ep\cup ep'}(t), R) \in max\phi\widehat{Q}$ such that $\bigcup_{b_i\in B} \alpha B_i \subseteq R$. Thus IR5 holds by $B' \subseteq B$ and CSP2 for $\widehat{Q}$.

($\Longrightarrow$) Let $t \in \tau P$ and $B \in \mathcal{D}_t$. Then, by IR5, we have that

$$(inv_{ep\cup ep'}(t), \{a \in \bigcup_{b_i\in B} \alpha B_i \mid inv_{ep\cup ep'}(t) \circ \langle a\rangle \in Dom_{ep\cup ep'}\}) \in \phi\widehat{Q} \ .$$

Thus, by CSP3 for $\widehat{Q}$ and (12), $(inv_{ep\cup ep'}(t), \bigcup_{b_i\in B} \alpha B_i) \in \phi\widehat{Q}$. Hence there is a maximal failure $(inv_{ep\cup ep'}(t), R) \in max\phi\widehat{Q}$ such that $\bigcup_{b_i\in B} \alpha B_i \subseteq R$. □

We now introduce notions corresponding to $\mathcal{D}_{t,R}$ and $\mathcal{D}_t$ in the domain of communicating transition systems. For every $p \in V_{P_{det}}$ and $A \in \kappa_P(p)$, we denote $\mathcal{D}_{p,A} = \{b_i \in \chi P \mid \alpha b_i \cap A = \emptyset\}$; moreover $\mathcal{D}_p = \{\mathcal{D}_{p,A} \mid A \in \kappa_P(p)\}$.

Proposition 35 *If $p_0 \stackrel{t}{\Longrightarrow} p$ then $\mathcal{D}_p = \mathcal{D}_t$. Moreover,*

1. *$\mathcal{D}_{p,A} = \mathcal{D}_{t,\alpha Q-A}$ for every $A \in \kappa_P(p)$.*
2. *$\mathcal{D}_{t,R} = \mathcal{D}_{p,\alpha Q-R}$ for every $(t, R) \in max\phi P$.*

Proof. The first part follows immediately from (1) and (2), so we only show these.

(1) Let $A \in \kappa_P(p)$. By corollary 13, $(t, R) \in max\phi P$ where $R = \alpha P - A$. Moreover, $\alpha b_i \cap A = \emptyset$ if and only if $\alpha b_i \subseteq R$.

(2) Let $(t, R) \in max\phi P$. By corollary 13, there exists $A \in \kappa_P(p)$ such that that $R = \alpha P - A$. Moreover, $\alpha b_i \subseteq R$ if and only if $\alpha b_i \cap A = \emptyset$. □

Proposition 36 *$\widehat{Q}$ (and so Q) satisfies IR5 if and only if for every $(p,q) \in sim^{min}_{inv}$, if $B \in \mathcal{D}_p$ then there is $A' \in \kappa_Q(q)$ such that $\bigcup_{b_i \in B} \alpha B_i \cap A' = \emptyset$.*

Proof. ($\Longrightarrow$) Let $(p,q) \in sim^{min}_{inv}$ and $B \in \mathcal{D}_p$. Then $p_0 \stackrel{t}{\Longrightarrow} p$ and $q_0 \stackrel{inv_{ep \cup ep'}(t)}{\Longrightarrow} q$, by proposition 33 and since sim^{min}_{inv} is the smallest inv-simulation. By propositions 12 and 35, $t \in \tau P$ and $\mathcal{D}_p = \mathcal{D}_t$. By proposition 34, there is $(inv_{ep \cup ep'}(t), R) \in max\phi\widehat{Q}$ such that $\bigcup_{b_i \in B} \alpha B_i \subseteq R$. Then, by corollary 27, $A' = \alpha Q - R \in \kappa_Q(q)$. And, for such an A', $\bigcup_{b_i \in B} \alpha B_i \cap A' = \emptyset$.

($\Longleftarrow$) Let $t \in \tau P$ and $B \in \mathcal{D}_t$. By propositions 12 and 35, there exists $p \in V_{Pdet}$ such that $p_0 \stackrel{t}{\Longrightarrow} p$ and $\mathcal{D}_t = \mathcal{D}_p$. By corollary 27, there exists $q \in V_{Q_{det}}$ such that $q_0 \stackrel{inv_{ep \cup ep'}(t)}{\Longrightarrow} q$ and so $(p,q) \in sim^{min}_{inv}$, by proposition 28. Hence there is $A' \in \kappa_Q(q)$ such that $\bigcup_{b_i \in B} \alpha B_i \cap A' = \emptyset$. Thus, by by corollary 27, $(inv_{ep \cup ep'}(t), R) \in max\phi\widehat{Q}$ where $R = \alpha Q - A'$. Moreover, for such an R, $\bigcup_{b_i \in B} \alpha B_i \subseteq R$. □

7 Algorithms

In this section, we outline algorithms for checking the implementation relations IR1–IR5 except for IR1(a) which is implicitly tested during the generation of CTS^u_Q provided that CTS^u_Q does not contain any τ-loop which is checked in order to establish that IR1(b) holds (see proposition 24).

To test for IR1(b), we use a modified version of the depth-first search algorithm given in [17] to test for strong connectivity in directed graphs. The (original) algorithm returns the nodes in each strongly connected component of the graph, thus indicating all those nodes on cycles. We make two modifications to the original algorithm. Firstly, we explore only τ-transitions and so return only those strongly connected components in which all nodes are mutually accessible by such transitions. Secondly, we ignore components consisting of only one node, unless they admit a self-τ-loop. Thus, due to proposition 24, IR1(b) is met if and only if the modified algorithm finds no τ-traversable strongly connected components.

Algorithm 3. The outline for the algorithm is shown in figure 15. The *visit* function executes the recursive depth-first search which searches for (τ-traversable) strongly connected components reachable (by τ-only transitions) from the node v for which it is originally called. If and when such a component is encountered, the nodes within it are recorded in the global variable *cyclicNodes*. The function *IR1b*() calls *visit* only for those nodes which have not already been considered and which have at least one τ-transition leaving them. □

The algorithm to test for IR1(c) is based on proposition 28. We aim to construct the minimal extr-simulation sim^{min}_{extr}, by traversing the product $V_{Q_{det}} \times V_{P_{det}}$. We first map the initial nodes to each other, $(q_0, p_0) \in sim_{extr}$. We then perform a depth-first search, beginning at (q_0, p_0). If the construction is successful, the set of all pairs of nodes reachable from (q_0, p_0) gives the minimal extr-simulation.

Algorithm 4. The pseudo-code of the algorithm is shown in figure 16. In the function *IR1c*() itself, the initial nodes in Q_{det} and P_{det} are paired. The *visit* function is then called for this initial pair of nodes. The data structure used to indicate if the paired nodes have been seen is:

- `array` $[1..|V_{Q_{det}}|][1..|V_{P_{det}}|]$ *jointNodes*

```
function IR1b()
   for every v ∈ V_{CTS^u_Q} such that τ ∈ en(v)
      if v has not already been seen then visit(v)
   if cyclicNodes ≠ ∅
      then return failure
      else return success
```

```
function IR2()
   for every q ∈ V_{Q_det}
      if q has not already been seen then visit(q)
   if cyclicNodes ≠ ∅
      then return failure
      else return success
```

Fig. 15. Testing for IR1(b) and IR2

The array represents $V_{Q_{det}} \times V_{P_{det}}$; if a combination in *jointNodes* has been *seen* by the execution of the depth-first search, then it constitutes a pair in sim^{min}_{extr} (this relation is then used in testing for IR3). □

```
function IR1c()
   jointNodes ← unseen; outcome ← success
   visit(q_0, p_0)
   return outcome

void visit(q, p)
   jointNodes[q][p] ← seen
   for every q --a--> q'
      if extr(q, a) = τ
         then
            if jointNodes[q'][p] = unseen then visit(q', p)
         else
            if extr(q, a) ∉ en(p)
               then outcome ← failure
               else if jointNodes[q'][p'] = unseen where p --extr(q,a)--> p' then visit(q', p')
   return
```

Fig. 16. Testing for IR1(c)

The algorithm to test for IR2 is based on proposition 29. We again use a modified version of the algorithm testing for strong connectivity. This time, however, we wish only to find those strongly connected components where all nodes are mutually accessible by τ-transitions after extraction. In addition, we wish to ignore components consisting of only one node q, unless $q \xrightarrow{a} q$ and $extr(q,a) = \tau$, as this does not constitute a cycle for our purposes.

Algorithm 5. The pseudo-code for the algorithm is shown in figure 15. The *visit* function executes the recursive depth-first search which searches for (τ-traversable, after extraction given by $extr(q,a)$) strongly connected components reachable (by empty traces, after applying extraction given by $extr(q,a)$) from the node q for which it is originally called. If and when such a component is encountered, the nodes within it are recorded in *cyclicNodes*. The function *IR2*() calls *visit* for every node $q \in V_{Q_{det}}$. □

The algorithm to test for IR3 is based on proposition 32 and uses the relation sim^{min}_{extr} calculated during the execution of algorithm 4.

Algorithm 6. The pseudo-code for the algorithm to test for IR3 is shown in figure 17. It uses three auxiliary functions:

- *IR3a*() to test for proposition 32(1) (which captures IR3(a)) for q and $\mathcal{C}_q$.
- *IR3b*() to test for proposition 32(2) (which captures IR3(b)) for q and $\mathcal{C}_q$.
- *getC*() to calculate the set $\mathcal{C}_q$ for a given q. □

To test for IR4 we use proposition 33, aiming to construct the minimal inv-simulation sim^{min}_{inv} by traversing the product $V_{P_{det}} \times V_{Q_{det}}$. We first map the initial nodes to each other, $(p_0, q_0) \in sim_{inv}$. We then perform a depth-first search, beginning at (p_0, q_0). If the construction is successful, the set of all pairs of nodes reachable from (p_0, q_0) gives the minimal inv-simulation.

If the depth-first search function has been called for $(p,q) \in sim^{min}_{inv}$, then we need to consider every transition a out of p, and generate $inv(a)$ before considering if the first event in $inv(a)$ is offered as a transition out of q.

Algorithm 7. The pseudo-code is shown in figure 18. The function *IR4*() calls *visit* for (p_0, q_0), as a result of which all pairs in sim^{min}_{inv} are reached. The algorithm uses the following global array:

- `array` $[1..|V_{P_{det}}|][1..|V_{Q_{det}}|]$*jointNodes*

The array represents $V_{P_{det}} \times V_{Q_{det}}$; if a combination in *jointNodes* has been *seen* by the execution of the depth-first search, then it constitutes a pair in sim^{min}_{inv} (this relation is then used in testing for IR5). □

The algorithm to test for IR5 is based on proposition 36, and uses the relation sim^{min}_{inv} calculated during the execution of algorithm 7.

Algorithm 8. The pseudo-code for the algorithm to test for IR5 is shown in figure 18. □

8 Concluding remarks

We have presented an implementation relation scheme which formalises the notion that a system built of communicating processes is an *implementation* of another base or target system in the event that the respective specification and implementation processes of which the systems are built have different interfaces. An important compositionality result was obtained which allows for automatic verifcation of the implementation relations in terms of constituent processes of the systems, so avoiding a major cause of the state explosion problem. We then presented algorithms for automatic verification of the implementation relations which take advantage of this compositionality result.

The algorithms presented here have been derived almost directly from the implementation relations themselves and little effort has yet been put into optimisation. Future work will explore possibilities for optimisation, as well as including a case study to evaluate the performance of the algorithms in practice and to establish in which context the implementation relations may prove to be most applicable.

```
function IR3()
   for every q ∈ V_{Q_det}
      C_q ← getC(q)
      if IR3a(q, C_q) = failure or IR3b(q, C_q) = failure then return failure
   return success

function IR3a(q, C_q)
   B ← ⋃_{B∈C_q} B
   for every b_i ∈ B
      if δ_i(q^(i)) = D then return failure
   return success

function IR3b(q, C_q)
   if δ_1(q^(1)) = ··· = δ_{m+n}(q^(m+n)) = d
   then
      for every B ∈ C_q
         for every p such that (q, p) ∈ sim^min_extr
           successful ← false
           for every A ∈ κ_P(p)
              if αB ∩ A = ∅ then successful ← true ; break
           if successful = false then return failure
   return success

function getC(q)
   C_q ← ∅
   for every A ∈ κ_Q(q)
      B ← ∅
      for every b_i ∈ in P
         if ∃R ∈ ϱ_i(q^(i)) : αB_i ∩ A ⊆ R then B ← B ∪ {b_i}
      for every b_i ∈ out P
         if ∄R ∈ ϱ_i(q^(i)) : αB_i − A ⊆ R then B ← B ∪ {b_i}
      C_q ← C_q ∪ {B}
   return C_q
```

Fig. 17. Testing for IR3

References

1. J. Baeten and W. P. Weijland: *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press (1990).
2. S. D. Brookes, C. A. R. Hoare and A. W. Roscoe: A Theory of Communicating Sequential Process. *Journal of ACM* 31 (1984) 560-599.
3. S. D. Brookes and A. W. Roscoe: An Improved Failures Model for Communicating Sequential Processes. Proc. of *Seminar on Concurrency*, S. D. Brookes, A. W. Roscoe and G. Winskel (Eds.). Springer-Verlag, Lecture Notes in Computer Science 197 (1985) 281-305.
4. J. Burton: *Verification of Implementations of Communicating Processes*. MSc Dissertation, University of Newcastle upon Tyne (1999).
5. R. De Nicola: Extensional Equivalences for Transition Systems. *Acta Informatica* 24 (1987) 211-237.
6. C. A. R. Hoare: *Communicating Sequential Processes*. Prentice Hall (1985).
7. G. Kahn and D. B. MacQueen: Coroutines and Networks of Parallel Processes. In: *Information Processing 77*, B. Gilchrist (Ed.). North-Holland (1977) 993-998.

```
function IR4()
   jointNodes ← unseen;  outcome ← success
   visit(p0, q0, ⟨⟩)
   return outcome

void visit(p, q, invEvents)
   if ⟨⟩ ≠ invEvents = ⟨a⟩ ∘ invEvents'
      then
         if a ∉ en(q)
            then outcome = failure
            else visit(p, q', invEvents') where q --a--> q'
      else
         if jointNodes[p][q] = unseen
            then
               jointNodes[p][q] ← seen
               for every p --a'--> p'
                  visit(p', q, inv_{ep ∪ ep'}(a'))
   return
```

```
function IR5()
  for every p ∈ V_{P_det}
    for every A ∈ κ_P(p)
      B ← χP − χA
      for every q such that (p, q) ∈ sim^{min}_{inv}
        matchFound ← false
        for every A' ∈ κ_Q(q)
          if ⋃_{b_i ∈ B} αB_i ⊆ χQ − χA' then matchFound = true ; break
        if matchFound = false then return failure
  return success
```

Fig. 18. Testing for IR4 and IR5

8. M. Koutny, L. Mancini and G. Pappalardo: Formalising Replicated Distributed Processing. Proc. of *10th Symposium on Reliable Distributed Systems*, (1991) 108-117.
9. 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.
10. M. Koutny and G. Pappalardo: The ERT Model of Fault-tolerant Computing and its Application to a Formalisation of Coordinated Atomic Actions. Technical Report 636, Department of Computing Science, University of Newcastle upon Tyne (1998).
11. M. Koutny and G. Pappalardo: A Model of Behaviour Abstraction for Communicating Processes. Proc. of *16th Symposium on Theoretical Aspects of Computer Science, STACS'99*, C. Meinel and S. Tison (Eds.). Springer-Verlag, Lecture Notes in Computer Science 1563 (1999) 313-322.
12. L. Mancini and G. Pappalardo: Towards a Theory of Replicated Processing. Proc. of *Formal Techniques in Real-Time and Fault-Tolerant Systems*, M. Joseph (Eds.). Springer-Verlag, Lecture Notes in Computer Science 331 (1988) 175-192.
13. R. Milner: *Communication and Concurrency*. Prentice Hall (1989).
14. G. Pappalardo: *Specification and Verification Issues in a Process Language*. PhD Thesis, University of Newcastle upon Tyne (1995).
15. A. W. Roscoe: *The Theory and practice of Concurrency*. Prentice-Hall (1998).
16. F. B. Schneider: Implementing Fault-tolerant Services Using the State Machine Approach: a Tutorial. *ACM Comput. Surveys* 22 (1990) 299-319.
17. R. Sedgewick: *Algorithms in C++*. Addison-Wesley (1992).
18. J. Xu, B. Randell, A. Romanovsky, C. Rubira, R. Stroud, and Z. Wu: Fault Tolerance in Concurrent Object-oriented Software Through Coordinated Error Recovery. Proc. of *25th International Symposium on Fault-Tolerant Computing*, IEEE Press (1995) 450-457.

A Appendix

$$
\begin{array}{ll}
\chi(P\|Q) & = \chi P \cup \chi Q \\
\delta(P\|Q) & = \{t \circ u \mid (t\lceil\chi P, t\lceil\chi Q) \in (\tau P \times \delta Q) \cup (\delta P \times \tau Q)\} \\
\phi(P\|Q) & = \{(t, R \cup S) \mid (t\lceil\chi P, R) \in \phi P \wedge (t\lceil\chi Q, S) \in \phi Q\} \cup \delta(P\|Q) \times 2^{\alpha(P\|Q)}
\end{array}
$$

$$
\begin{array}{ll}
\chi(P\backslash B) & = \chi P - B \\
\delta(P\backslash B) & = \{t\lceil\chi(P\backslash B) \circ u \mid t \in \delta P \ \vee \ \exists a_1, a_2, \ldots \in \alpha B \ \forall n \geq 1 : t \circ \langle a_1, \ldots, a_n\rangle \in \tau P\} \\
\phi(P\backslash B) & = \{(t\lceil\chi(P\backslash B), R) \mid (t, R \cup \alpha B) \in \phi P\} \cup \delta(P\backslash B) \times 2^{\alpha(P\backslash B)}
\end{array}
$$

$$
\begin{array}{ll}
\chi P[b/b'] & = \chi P - \{b'\} \cup \{b\} \\
\delta P[b/b'] & = \{t[b/b'] \mid t \in \delta P\} \\
\phi P[b/b'] & = \{(t[b/b'], R[b/b']) \mid (t, R) \in \phi P\}
\end{array}
$$

$$
\begin{array}{ll}
\chi(a \rightarrow P) & = \chi P \\
\delta(a \rightarrow P) & = \{\langle a\rangle \circ t \mid t \in \delta P\} \\
\phi(a \rightarrow P) & = \{(\langle a\rangle \circ t, R) \mid (t, R) \in \phi P\} \cup \{\langle\rangle\} \times 2^{\alpha P - \{a\}}
\end{array}
$$

$$
\begin{array}{ll}
\chi(P \,\Box\, Q) & = \chi P \\
\delta(P \,\Box\, Q) & = \delta P \cup \delta Q \\
\phi(P \,\Box\, Q) & = \{(\langle\rangle, R) \mid (\langle\rangle, R) \in \phi P \cap \phi Q\} \cup \{(t, R) \mid t \neq \langle\rangle \ \wedge \ (t, R) \in \phi P \cup \phi Q\}
\end{array}
$$

$$
\begin{array}{ll}
\chi(P \sqcap Q) & = \chi P \\
\delta(P \sqcap Q) & = \delta P \cup \delta Q \\
\phi(P \sqcap Q) & = \phi P \cup \phi Q \ .
\end{array}
$$

In the above, B is a proper subset of χP; $b \notin \chi P$ and $b' \in \chi P$ are channels with the same message sets; $R[b/b']$ is R with each $b'!v$ changed to $b!v$; a is an action in αP; in the last two definitions $\chi P = \chi Q$.