Verifying Implementation Relations

Jonathan Burton and Maciej Koutny

Department of Computing Science, University of Newcastle,
Newcastle upon Tyne NE1 7RU, U.K. {j.i.burton, Maciej.Koutny}@ncl.ac.uk

Abstract. Implementation relations are a means to relate the behaviour of implementation and specification systems built of communicating processes in the event that respective implementation and specification processes have differing interfaces. In this paper we first present a graph-theoretic statement of such relations and then derive algorithms for their automatic verification.
Keywords: Behaviour abstraction, communicating sequential processes, compositionality, verification.

1 Introduction

The papers [1, 3, 4] introduced an implementation relation scheme aimed at formalising the notion that a communicating 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. Here we present a graph-theoretic restatement of those implementation relations, from which we directly derive algorithms for their automatic verification.

The implementation relation scheme developed in [1] (based on [3, 4] and reproduced in section 4) 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 conventional 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.

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 (substantial) subset 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

[1] In the treatment of NMR [4], this environment is made up of *disturbers*, feeding faulty but sufficiently redundant input, and *extractors*, which interpret the implementation's output.

labelled transition systems with which we represent CSP processes.[2] 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.

To relate behaviour on a set of channels in the implementation process with that on a channel in the specification process, we use the notion of an *extraction pattern*. A set of extraction patterns acts as a parameter to the implementation relation scheme.

The paper is organised as follows. The next section gives preliminary information. Section 3 describes the motivation behind our approach and defines the notion of extraction pattern. Section 4 presents the implementation relations themselves. Section 5 deals with computer representations of CSP processes and extraction patterns, in both cases employing a variant of a labelled transition system. In section 6 we make the necessary technical steps to relate the labelled transition systems representing processes and extraction patterns, and in sections 7 and 8 we show how the defining conditions for implementation relations can be verified algorithmically.

A preliminary version of this paper was presented at the "International Workshop on Verification and Computational Logic" held in London, July 2000. All proofs can be found in [1].

2 Preliminaries

We use the standard divergence model of CSP (Communicating Sequential Processes) [2, 5] 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^*. Moreover, $\tau P = \{t \mid (t, R) \in \phi P\}$ denotes the *traces* of P. We also define *maximal failures* as $max\phi P = \{(t, R) \in \phi P \mid (t, S) \in \phi P \ \wedge \ R \subseteq S \ \Rightarrow \ R = S\}$.

We use structured actions of the form $b!v$, where v is a *message* and b is a communication *channel*. For every channel b, μb is the *message set* of b - the set of all v such that $b!v$ is a valid action. We define $\alpha b = \{b!v \mid v \in \mu b\}$ to be the *alphabet* of channel b. It is assumed that μb is always finite and non-empty. For a set of channels B, $\alpha B = \bigcup_{b \in B} \alpha b$. χa gives the channel on which the event a occurred; for example, $\chi b!1 = b$. Moreover, $\chi A = \{\chi a \mid a \in A\}$. We will associate with P a set of channels, χP, and stipulate that the alphabet of P is that of χP. Moreover, χP is partitioned into *input* and *output* channels, respectively denoted by $in\,P$ and $out\,P$. The *composition* of two processes, P and Q, is defined as $P \otimes Q = (P \| Q) \backslash (\alpha P \cap \alpha Q)$ i.e., it is a parallel composition of P and Q with all the interprocess communication hidden.

Throughout the paper we use notations similar to those of [2]. In addition to that, an infinite sequence of traces $t_1, t_2, \ldots$ is *ω-monotonic* if $t_1 \leq t_2 \leq \ldots$ and $\lim_{i \to \infty} |t_i| = \infty$. A trace $t[c/b]$ is obtained from trace t by replacing each action $b!v$ by $c!v$, and $t\lceil B$ is obtained by deleting from t all the actions that do not occur on the channels in B. A

[2] Communicating processes are represented formally in the FD model of CSP[5]. Moreover, we assume that the specification processes with which we are dealing are *IO processes* as defined in section 2.

mapping from a set of traces to a set of traces $f : T \to T'$ is *monotonic* if $t, u \in T$ and $t \leq u$ implies $f(t) \leq f(u)$; and *strict* if $\langle\,\rangle \in T$ and $f(\langle\,\rangle) = \langle\,\rangle$. Moreover, f is a *homomorphism* if $t, u, t \circ u \in T$ implies $f(t \circ u) = f(t) \circ f(u)$.

The class of base processes comprises all those processes whose input channels cannot refuse a message purely on the basis of its content. Formally, a channel c of a process P is *value independent*, denoted $c \in \mathit{vind}\, P$, if for all $(t, R) \in \phi P$, $c \in \chi R$ implies $(t, R \cup \alpha c) \in \phi P$. 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 $P \in \mathit{IO}$.

3 Extraction patterns

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

$$\mathit{Snd} = \Box_{i \in \{0,1\}}\, c!i \to \mathit{Snd}_i \quad \text{and} \quad \mathit{Buf} = \Box_{i \in \{0,1\}}\, d!i \to B_i$$

where $\mathit{Snd}_i = d!i \to \mathit{Snd}_i$ and $B_i = e!i \to \mathit{Buf}$, for $i = 0, 1$.

Suppose that the signal transmission between the two processes has been implemented using two channels, r and s, as shown in figure 1(b). The transmissions on d are now duplicated and the two copies sent along r and s. That is, Snd' sends the duplicated signal, while Buf' accepts a single copy and passes it on (possibly after a delay), ignoring the other one. The scheme clearly works as we have $\mathit{Snd} \otimes \mathit{Buf} = \mathit{Snd}' \otimes \mathit{Buf}'$. Suppose now that the transmission of signals is imperfect and two types of faulty behaviour can occur: $\widetilde{\mathit{Snd}} = \mathit{Snd}' \sqcap \mathit{stop}$ and $\widehat{\mathit{Snd}} = \mathit{Snd}' \sqcap \overline{\mathit{Snd}}$, where $\overline{\mathit{Snd}}$ is Snd' with all the communication on channel s being blocked. In other words, $\widetilde{\mathit{Snd}}$ can break down completely, refusing to output any signals, while $\widehat{\mathit{Snd}}$ can fail in such a way that although channel s is blocked r can still transmit the signals. $\widehat{\mathit{Snd}}$ could be used to model the following situation: in order to improve performance, a 'slow' channel d is replaced by two channels, a high-speed yet unreliable channel s and a slow but reliable backup channel r. Since $\mathit{Snd} \otimes \mathit{Buf} = \widehat{\mathit{Snd}} \otimes \mathit{Buf}'$ and $\mathit{Snd} \otimes \mathit{Buf} \neq \widetilde{\mathit{Snd}} \otimes \mathit{Buf}'$ it follows that $\widehat{\mathit{Snd}}$ is a much 'better' implementation of the Snd process than $\widetilde{\mathit{Snd}}$.

We will now point at the differences between the two processes and at the same time introduce informally some basic concepts which are subsequently used. We start by observing that the output of $\widehat{\mathit{Snd}}$ can be thought of as adhering to two rules:

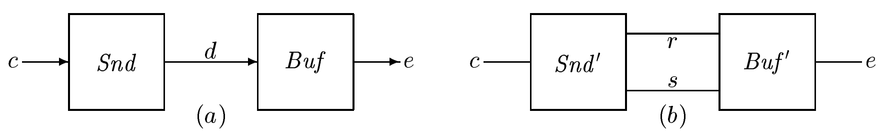

Fig. 1. Two base IO processes and their implementations.

R1 The transmissions over r and s are consistent w.r.t. message contents.
R2 Transmission over r is reliable, but there is no such guarantee for s.

The output produced by $\widetilde{Snd}$ satisfies R1 but fails to satisfy R2. To express this formally we need to render the two conditions in some form of precise notation.

To capture the behavioural relationship that exists between Snd and $\widehat{Snd}$ we will employ an (extraction) mapping *extr* which for traces over r and s returns corresponding traces over d. For example, $\langle\,\rangle \mapsto \langle\,\rangle$, $\langle r!0\rangle \mapsto \langle d!0\rangle$, $\langle s!0\rangle \mapsto \langle d!0\rangle$, $\langle s!1, r!1\rangle \mapsto \langle d!1\rangle$ and $\langle r!1, r!0\rangle \mapsto \langle d!1, d!0\rangle$. Note that the extraction mapping need only be defined for traces satisfying R1.

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

$$(\langle\,\rangle, \{s!0\}) \in \phi\widehat{Snd} \quad \text{and} \quad extr(\langle\,\rangle, \{s!0\}) = (\langle\,\rangle, \{d!0\}) \notin \phi Snd.$$

This indicates that the crude extraction of refusals is not going to work. What we need is another device, which comes in the form of another mapping, *ref*, constraining the possible refusals that a process can exhibit after a given trace. This will help prevent, for example, the sender process from blocking if its transmission is yet incomplete. In the example at hand, this roughly amounts to requiring that the communication on channel r should not be blocked ($\widetilde{Snd}$ fails to satisfy a similar condition). For example, we will stipulate that $ref(\langle s!0\rangle)$ must not comprise a refusal containing both $r!0$ and $r!1$. We will also denote by *dom* traces conveying information that can be regarded as complete. According to R2, $\langle s!0\rangle$ will not belong to *dom* as it is guaranteed that the message $\langle r!0\rangle$ will sooner or later be transmitted.

The last notion we will need to establish the correspondence between processes is a partial inverse of the extraction mapping, *inv*. It will be used to ensure that all the traces of a base process (see section 2) can be extracted from the traces of its implementation. (Note that implementations are not forced to preserve the direction of transfer of messages: see example in [1].)

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 with $Snd[e/d]$ using, e.g., CSP laws [2]. 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 in [1], some of which are recalled later in this paper.

Formal definition The notion of extraction pattern relates behaviour on a set of channels in an implementation process to that on a channel in a target process. It has two main functions: that of interpretation of behaviour necessitated by interface difference and the encoding of some correctness requirements. Formally, an *extraction pattern* 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 is 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 a trace over a channel 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 in terms of extraction patterns of a condition imposed on CSP processes that impossible events are always refused. Note that since *inv* is a homomorphism, it suffices to define it for single actions over the target only.

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 $ep_i = (B_i, b_i, dom_i, extr_i, ref_i, inv_i)$ with distinct targets and disjoint sources; 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 $a \in \alpha b_i$.

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

Example extraction pattern We have already discussed a kind of 'fail-stop' [6] extraction pattern for the example in figure 1. It can be formalised as the extraction pattern denoted by ep_{fs}, where $B = \{r, s\}$, $b = d$ and

$$\begin{aligned} dom &= \{t \mid (t\lceil s)[r/s] \leq t\lceil r\} & \quad ref(t) &= \{R \mid \alpha r \not\subseteq R\} \\ extr(t) &= \max\{(t\lceil s)[d/s], (t\lceil r)[d/r]\} & \quad inv(d!v) &= \langle r!v \rangle. \end{aligned}$$

Note that $Dom = \{t \mid (t\lceil s)[r/s] \leq t\lceil r \ \vee \ t\lceil r \leq (t\lceil s)[r/s]\}$.

4 Implementation relations

Suppose that we implemented the base *IO* process P using another process Q (see figure 2). The correctness of the implementation will be expressed in terms of two sets of extraction patterns, ep and ep'. The former (with sources $in\, Q$ and targets $in\, P$) will be used to relate the communication on the input channels of P and Q; the latter (with sources $out\, Q$ and targets $out\, P$) will serve a similar purpose for the output channels. There are several properties that Q has to satisfy according to the definition given below.

Firstly, if a trace t of Q projected on its input channels can be interpreted by ep, then it should be possible to interpret the projection on the output channels by ep' (see IR1(a,c) below). Moreover, such a trace should not lead to a divergence, which in CSP signifies an unacceptable behaviour (see IR1(b)). IR2 further ensures that the extracted traces do not lead to divergences in P. IR3 ensures that going outside the bounds of allowed refusals indicates that the communication on a given channel is a completed one, and, if this is true of all the channels, then the corresponding (sets of) refusals are present in the base process. IR4 simply states that the functionality in terms of traces present in P can also be found in Q and IR5 extends this to refused sets as well.

Formally, let P be a base *IO* process as in figure 2 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 2.

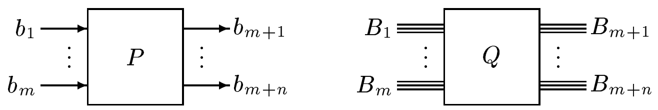

Fig. 2. Base *IO* process P and its implementation Q.

We are now ready to introduce three implementation relations which will provide a means to relate the base and implementation processes. 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 relations form a hierarchy: $Impl_3(P, ep, ep') \subseteq Impl_2(P, ep, ep') \subseteq Impl_1(P, ep, ep')$. Crucially, all three implementation relations are preserved by the process composition operation (details can be found in e.g., [1, 3]).

5 Representing CSP processes and extraction patterns

CSP Processes We shall represent a CSP process in terms of a *communicating transition system* (CTS). A CTS 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 $v \xrightarrow{a} w$.
- If $v_1 \xrightarrow{a_1} v_2 \xrightarrow{a_2} \cdots \xrightarrow{a_n} v_{n+1}$, we denote $v_1 \xRightarrow{\langle a_1\rangle\circ\cdots\circ\langle a_n\rangle} v_{n+1}$ where it is assumed that $\langle\tau\rangle = \langle\rangle$; moreover, $v \xRightarrow{\langle\rangle} v$, for every $v \in V$.
- If $v \xrightarrow{a} w$, 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 $v \xRightarrow{t} w$, we denote $v \Longrightarrow w$ or $v \xRightarrow{t}$.

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

The implementation relations which we want to verify algorithmically are all expressed in the denotational semantics of CSP [5], 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 $\alpha CTS = \alpha C \cup \alpha D$):

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

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

It turns out that P_{CTS} is a CSP process. Moreover, if $\Delta = \emptyset$, then $\tau P_{CTS} = \{t \in \alpha CTS^* \mid v_0 \xRightarrow{t}\}$.

Calculating failures Given a finite $CTS = (V, C, D, A, v_0)$ such that $\Delta = \emptyset$, we form a labelled transition system CTS_{det} (using a process of *normalisation* adapted from [5]) whose nodes $V_{CTS_{det}}$ are members of the powerset of V. We also denote, for every node p of CTS_{det},

$$\kappa_{CTS}(p) = \{en(v) \mid v \in p \ \wedge \ \forall w \in p : en(w) \subseteq en(v) \implies en(w) = en(v)\}\ . \quad (1)$$

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 maximal failures.

Extraction Patterns We now turn to the representation of extraction patterns. An *extraction graph* is a tuple $EG = (B, b, V, A, v_0, \varrho, \delta, \imath)$ 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*. We will use the following notation:

- If $(v, a, t, w) \in A$, we denote $v \xrightarrow{a:t} w$.
- If $v_1 \xrightarrow{a_1:t_1} v_2 \xrightarrow{a_2:t_2} \cdots \xrightarrow{a_n:t_n} v_{n+1}$, 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.
- If $v \xRightarrow{u:t} w$, we denote $v \Longrightarrow w$.

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

- 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 $a \in R$.
- $v_0 \Longrightarrow v$.
- If $v \xrightarrow{a:t} w$ and $v \xrightarrow{a:t'} w'$ then $t = t'$ and $w = w'$. (*)
- If $\delta(v) = D$, then there is $w \in V$ such that $v \Longrightarrow w$ and $\delta(w) = d$.
- 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 be an extraction graph as above. 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.

- $Dom_{EG} = \{u \in \alpha B^* \mid \exists v, t : v_0 \xRightarrow{u:t} v\}$.
- $dom_{EG} = \{u \in \alpha B^* \mid \exists v, t : v_0 \xRightarrow{u:t} v \wedge \delta(v) = d\}$.
- For every $t = \langle a_1, \ldots, a_k\rangle \in \alpha b^*$, $inv_{EG}(t) = \imath(a_1) \circ \cdots \circ \imath(a_k)$.
- By (*) above, for every $u \in Dom_{EG}$, there are *unique* t and v such that $v_0 \xRightarrow{u:t} v$. We then define $extr_{EG}(u) = t$ and $ref_{EG}(u) = \{R \mid \exists R' \in \varrho(v) : R \subseteq R'\}$.

We have that ep_{EG} is an extraction pattern, and that for every extraction pattern ep there is an extraction graph EG such that $ep = ep_{EG}$. However, we will be interested only in those extraction graphs which are finite, i.e., have a finite number of nodes V. To simplify the presentation, we will assume that if $v \overset{a:t}{\Longrightarrow} w$ then $|t| \leq 1$.

6 Unambiguous CTS

It is possible that the CTS representing Q (i.e. the implementation process) will be ambiguous with respect to interpretation in terms of the EGs. For example, let $q_0, q \in V_{CTS_Q}$ be such that $q_0 \overset{t}{\Longrightarrow} q$, $q_0 \overset{u}{\Longrightarrow} q$ and $q \xrightarrow{a} q'$. Suppose that a extracts to a different sequence of events depending on whether it is an extension of trace t or an extension of trace u. In this case, it would be impossible to unambiguously interpret the single transition labelled by a which goes from q to q' in terms of a transition in any of the extraction graphs.

To tackle this problem we generate an equivalent, unambiguous CTS from a given CTS and a set of extraction graphs. Intuitively, the algorithm below splits nodes like the q above into several nodes corresponding to the different ways in which it can be reached from q_0 from the point of view of the extraction patterns.

Algorithm 1. For $i = 1, \ldots, m+n$, let $EG_i = (B_i, b_i, V_i, A_i, v_{0i}, \varrho_i, \delta_i, \imath_i)$ 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 $C = B_1 \cup \ldots \cup B_m andD = B_{m+1} \cup \ldots \cup B_{m+n}$. 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 $i \geq 1$.[3]
   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 condition (*) in section 5.
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). □

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

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. In addition, $\tau P_{CTS^u_Q} = \tau Q \cap Dom_{ep \cup ep'}$ (see proposition 2). 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 $q^{(i)} = w_i$.

7 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 8, a 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 4.
- CTS_P and CTS_Q are communicating transition systems representing P and Q respectively; i.e., $P = P_{CTS_P}$ and $Q = P_{CTS_Q}$.
- For $i = 1, \ldots, m+n$, EG_i is an extraction graph with initial node v_{0i} and 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 (1) for the nodes of P_{det}, and denote the initial state of P_{det} by p_0.
- CTS^u_Q is a disambiguated version of CTS_Q w.r.t. extraction graphs EG_i (see algorithm 1). We will denote the initial state of CTS^u_Q by q^u_0.
  The process generated by CTS^u_Q will be denoted by $\widehat{Q}$; i.e., $\widehat{Q} = P_{CTS^u_Q}$.
- Q_{det} is the normalised version of CTS^u_Q. We will use κ_Q to denote the mapping defined as in (1) for the nodes of Q_{det}, and denote the initial state of Q_{det} by q_0.
  It may be observed 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.

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

Proposition 1 *Q satisfies IR1(a,b) if and only if CTS^u_Q has been successfully generated and there are no nodes $v_1, \ldots, v_k$ $(k \geq 2)$ in CTS^u_Q such that $v_1 \xrightarrow{\tau} v_2 \cdots \xrightarrow{\tau} v_k = v_1$.*

From now on, we assume that CTS^u_Q has been successfully generated and does not contain any τ-loops, and so IR1(a,b) hold. Then CTS^u_Q generates a process which can be used to test for all implementation relations in place of Q.

Proposition 2 *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 $\widehat{Q} \in Impl_i(P, ep, ep')$.

As we have already seen, IR1(a,b) can be checked directly using CTS^u_Q. In dealing with the remaining conditions, we assume that IR1(a,b) hold, and use Q_{det}, which is the normalised CTS derived from CTS^u_Q. Note that, by propositions 1 and 2, we may assume that $\widehat{Q}$ is an implementation process such that $\delta\widehat{Q} = \emptyset$ and $trace\widehat{Q} \subseteq Dom_{ep\cup ep'}$.

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

$$q \xrightarrow{a} q' \implies \exists (q',p') \in sim_{extr} : p \xRightarrow{\langle extr(q,a)\rangle} p' \ . \qquad (2)$$

Proposition 3 *$\widehat{Q}$ satisfies IR1(c) if and only if there is an extr-simulation.*

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 $\widehat{Q}$ satisfies IR1(c). Then, testing for IR2 amounts to checking for extracted τ-loops in the graph of Q_{det}.

Proposition 4 *$\widehat{Q}$ satisfies IR2 if and only if there are no nodes $v_1, \ldots, v_k$ $(k \geq 2)$ in Q_{det} such that $v_1 \xrightarrow{a_1} v_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{k-1}} v_k = v_1$ and $extr(v_i, a_i) = \tau$, for all $i \leq k$.*

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 $\overline{\mathcal{C}}_t = \bigcup\{B \mid B \in \mathcal{C}_t\}$.

One can see that $\widehat{Q}$ satisfies IR3 if and only if, for every $t \in \tau\widehat{Q}$, the following hold: (i) if $b_i \in \overline{\mathcal{C}}_t$ then $t\lceil B_i \in dom_i$; and (ii) 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 $\alpha B \subseteq R$.

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 $\overline{\mathcal{C}}_q = \bigcup\{B \mid B \in \mathcal{C}_q\}$.

One can see that if $q_0 \xRightarrow{t} q$ then $\mathcal{C}_q = \mathcal{C}_t$. Moreover, $\mathcal{C}_{q,A} = \mathcal{C}_{t,\alpha Q - A}$ for every $A \in \kappa_Q(q)$ and $\mathcal{C}_{t,R} = \mathcal{C}_{q,\alpha Q - R}$ for every $(t,R) \in max\phi\widehat{Q}$.

Proposition 5 *$\widehat{Q}$ satisfies IR3 if and only if, for every $q \in V_{Q_{det}}$:*

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

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* if $(p_0, q_0) \in sim_{inv}$ and, for every $(p, q) \in sim_{inv}$,

$$p \xrightarrow{a} p' \implies \exists (p', q') \in sim_{inv} : q \xRightarrow{inv_{ep \cup ep'}(a)} q' \ . \qquad (3)$$

Proposition 6 *$\widehat{Q}$ satisfies IR4 if and only if there exists an inv-simulation.*

When dealing with IR5, we will assume that $\widehat{Q}$ 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 also exists the smallest one, sim_{inv}^{min}.

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, $\mathcal{D}_t = \{\mathcal{D}_{t,R} \mid (t, R) \in max\phi P\}$. We have that $\widehat{Q}$ satisfies IR5 if and only if, for all $t \in \tau P$ and $B \in \mathcal{D}_t$, there is $(inv_{ep \cup ep'}(t), R) \in max\phi\widehat{Q}$ such that $\bigcup_{b_i \in B} \alpha B_i \subseteq R$.

We now introduce notions corresponding to $\mathcal{D}_{t,R}$ and $\mathcal{D}_t$ in the domain of communicating transition systems. For all $p \in V_{P_{det}}$ 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 $\mathcal{D}_p = \{\mathcal{D}_{p,A} \mid A \in \kappa_P(p)\}$. One can then see that if $p_0 \xRightarrow{t} p$ then $\mathcal{D}_p = \mathcal{D}_t$. Moreover, $\mathcal{D}_{p,A} = \mathcal{D}_{t,\alpha Q - A}$ for every $A \in \kappa_P(p)$ and $\mathcal{D}_{t,R} = \mathcal{D}_{p,\alpha Q - R}$ for every $(t, R) \in max\phi P$.

Proposition 7 *$\widehat{Q}$ satisfies IR5 if and only if for every $(p, q) \in sim_{inv}^{min}$, if $B \in \mathcal{D}_p$ then there is $A' \in \kappa_Q(q)$ such that $\bigcup_{b_i \in B} \alpha B_i \cap A' = \emptyset$.*

8 Algorithms

We now briefly outline algorithms for checking the implementation relations IR1–IR5, except for IR1(a), which is implicitly tested during the generation of CTS_Q^u provided that IR1(b) holds.

To test for IR1(b) and IR2 respectively we use modified versions of the depth-first search algorithm given in [7] to test for strongly connected components in directed graphs.

The algorithm to test for IR1(c) is based on proposition 3. We aim to construct the minimal extr-simulation sim_{extr}^{min}, 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.

The algorithm to test for IR3 is based on proposition 5 and uses the relation sim_{extr}^{min} calculated during the testing for IR1(c). The pseudo-code is shown in figure 3. It uses three auxiliary functions:

- *IR3a*() to test for proposition 5(1) (which captures IR3(a)) for q and $\mathcal{C}_q$.
- *IR3b*() to test for proposition 5(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 6, aiming to construct the minimal inv-simulation sim_{inv}^{min} 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 (stored in the hashtable *jointNodes*). The pseudo-code is shown in figure 4.

The algorithm to test for IR5 uses the relation sim_{inv}^{min} generated when testing for IR4 and is based on proposition 7. The pseudo-code is shown in figure 4.

9 Concluding remarks

The algorithms presented here assume a sequential model of behaviour and may be best suited to the verification of implementation processes which employ complicated (communication) protocols but whose behaviour is essentially sequential. However, some implementation processes may be highly concurrent — for example those that employ replication to provide fault tolerance — and future work will therefore include an extension of the treatment given here to deal in a more appropriate manner with such implementations, as well as a case study to investigate the performance in practice of the algorithms presented here.

References

1. J. Burton, M. Koutny and G. Pappalardo: Modelling and Verification of Communicating Processes in the Event of Interface Difference. Technical Report CS-TR-696, Department of Computing Science, University of Newcastle upon Tyne (2000).
2. C. A. R. Hoare: *Communicating Sequential Processes*. Prentice Hall (1985).
3. 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.
4. 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.
5. A. W. Roscoe: *The Theory and practice of Concurrency*. Prentice-Hall (1998).
6. F.B.Schneider: Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surveys 22 (1990) 299-319.
7. R. Sedgewick: *Algorithms in C++*. Addison-Wesley (1992).

```
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. 3. Testing for IR3.

```
function IR4()
    outcome ← success
    visit(p_0, q_0, ⟨⟩)
    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 (p,q) not in jointNodes
          then
            enter (p,q) in jointNodes
            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. 4. Testing for IR4 and IR5.