The ERT Model of Fault-Tolerant Computing and Its Application to a Formalisation of Coordinated Atomic Actions

Maciej Koutny
Department of Computing Science
Claremont Tower, Claremont Road
University of Newcastle
Newcastle upon Tyne NE1 7RU
U.K.

Giuseppe Pappalardo
Dipartimento di Matematica
Viale A. Doria 6
Università degli Studi di Catania
I-95125 Catania
Italy

February 25, 1998

Abstract

The Coordinated Atomic (CA) action concept is an approach to structuring complex concurrent activities in a distributed environment, aimed at supporting fault tolerance in the design of object-oriented systems. In this paper we investigate the issues involved in the formalisation of systems based on CA actions. For this investigation we have chosen a compositional model of system behaviour which enables one to relate the behaviour of (simple) base processes and their (complex) implementations, and state the correctness of the latter in terms of the former. The base processes can be thought of as specifications, or ideal processes operating in an error-free environment, while the implementations model their actual realisation which can exploit CA actions, possibly combined with a variety of fault-tolerant techniques to deliver reliable results.

Keywords: Distributed systems, CA actions, concurrency control, communicating sequential processes, fault tolerance, nondeterminism, partial and total correctness, verification, divergence, formal semantics.

1 Introduction

1.1 Coordinated Atomic actions

The Coordinated Atomic (CA) action concept [11, 15, 16] is an approach to structuring complex concurrent activities in a distributed environment, aimed at supporting fault tolerance in object-oriented systems. In its current presentation, the CA action model provides a conceptual framework in which fault tolerance is achieved by integrating the concepts of conversations and transactions. The former are used to control cooperative concurrency and coordinated error recovery, while the latter are used to maintain the consistency of shared resources. The following are some of the essential characteristics of the model [16]: (i) a CA action has roles which are activated by some external participants (processes or threads); (ii) a CA action starts when all the roles have been activated and finishes when each role has completed its execution; (iii) the execution of a CA action updates the system state (represented by a set of external objects) atomically; (iv) roles can access local objects as well as participate in nested CA actions; and (v) CA actions provide a basic framework for exception handling that can support a variety of fault tolerance mechanisms to tolerate both hardware and software faults.
1.2 Modelling fault tolerant systems with CA actions

When tackling the formalisation of some fundamental software engineering concept, such as that of a CA action, one can envisage different levels of abstraction at which such a formalisation could be developed. For example, one could specify a distributed (sub)system as a complex Petri net or a process algebra expression, and then manipulate it in order to show that it satisfies its specification, given in terms of another, simpler object, or a formula in a suitable logic. This is a well established method when dealing with systems operating in a fault-free external environment. However, if one adds the possibility of unpredictably faulty behaviour by the environment and the system, then it is no longer straightforward to say what is an 'acceptable' or 'adequate' behaviour of the latter. Indeed, due to the application of fault tolerant techniques, the system may exhibit some level of incorrectness yet without being inadequate. Thus, it is crucial to be able to say what external (or observable) behaviour of an object is considered acceptable in the environment in which it operates. If we adopt this view, then it becomes apparent that the property of being acceptable will depend on the fault tolerant mechanisms employed in the context in which the system operates; of course, in particular, in the absence of such mechanisms, the system has to be simply fault-free. We therefore propose that a first step on the way to produce a formal model of CA actions should be the development of a technique allowing one to state that a model of an actual design is an acceptable realisation of its specification. In the next (verification) step one would aim at actually proving that such a relationship does indeed hold for a given system, using suitable techniques based, for instance, on Petri nets or process algebras. In this paper, we shall address the former issue.

An important requirement for a formalisation of CA actions is that it should be simple, and preferably based on a well established model of behaviour. There are at least two reasons why one might want this to be the case. The first is that complex behavioural models can obscure or be incompatible with the inherent features of CA actions; the second is that a well established model can usually be adapted with little overhead to other system modelling frameworks, which would be a significant advantage in the second step of the development.

The treatment of a real life distributed fault-tolerant system is complicated by the rapid growth of the associated state space. Indeed one of the primary roles of CA actions is to control and manage this complexity by applying appropriate structuring to the system’s design. It therefore seems both unavoidable and desirable that the formalisation of CA actions should directly support modular and hierarchical development.

To summarise, we would aim at a model of behaviour allowing the correctness of CA actions based systems to be stated in a framework in which modular and hierarchical reasoning is supported, and a wide range of fault tolerant techniques is expressible.

1.3 The ERT model of behaviour

As a solution, this paper introduces a formal model of distributed behaviour which significantly refines and extends that proposed in [8]. We will present the model and some of its fundamental features, and then discuss how CA actions and their properties can be cast in this general framework. We will call the model ERT, for Extraction, Refusals and Traces, where the second and third terms come from the CSP model [1, 2, 3] on which it is based, and the first refers to a specific technique used to relate systems specified at different levels of abstraction.

The ERT model originates from previous work on distributed replicated systems [5, 10], which has led to the development of a formal model [6, 7, 8], based on an implementation relation capturing the notion that a replicated system is a correct implementation of another target or base system. The relation was defined independently of replication, in an abstract way, which enabled a uniform treatment
both of implementation techniques different from replication, and of different fault assumptions for replication (e.g. fail-stop faults [13], or byzantine faults [9]). Although originally introduced to model replicated processing, implementation relations can be seen as an effective technique for a broader range of distributed systems, with a potential of being able to deal with fault-tolerance paradigms other than replication, and various interface refinement schemes. Implementation relations are defined in terms of an extraction function; intuitively, when applied to an input/output sequence of an implementation, this function should yield an input/output sequence of the base system. Extraction functions are used to formalise the interface of the implementation and relate it to that of the base system.

There are two operational criteria for an implementation relation to be acceptable: realisability by extraction and compositionality. Realisability ensures that the implementation can be used instead of the base process in actual implementation. Compositionality requires that, if the systems in a set implement each a base system, their composition should implement the composition of the base systems. This meets the requirement of facilitating the modular and hierarchical development of distributed systems.

The work on ERT has been carried out in the formal setting provided by the CSP divergence-based model [2, 3], which incorporates not only the action sequences a process may engage in (traces), but also the action sets it may refuse (refusals) and the traces after which it may diverge (divergences). Traces alone provide a tool to express and prove the partial correctness of processes, while failures and divergences enable nondeterminism (including deadlock freedom) to be treated. In the view of [3], this amounts to total correctness.

The model of CSP leads to a natural formulation in terms of traces for the classes of input faults an implementation system may tolerate (the input fault assumptions). Moreover, it facilitates viewing a fault-tolerant system as an indeterminate member of a set of processes, rather than as a particular process; this is desirable, for the behaviour of a fault-tolerant system cannot of course be fully specified a priori but just constrained. An advantage of CSP is also its smooth treatment of process composition, which is crucial to the above compositionality property, and therefore to the modular design and analysis of distributed systems.

1.4 Outline of this paper

In this paper we generalise and substantially strengthen the model introduced in [8], by extending the notion of compositionality to extraction functions, and by allowing more general networks of communicating processes. The resulting ERT model is initially provided with two implementation relations: weak — for dealing with acyclic networks of base processes, and strong — for dealing with cyclic networks of base processes. In each case the implementation can be shown to be compositional and support realisability.

The paper is organised as follows. In the next section we introduce some basic notions used throughout the paper. In section 3 we introduce extraction patterns — a central notion to defining the interface of an implementation — and discuss examples of extraction patterns. Section 4 deals with acyclic networks of processes of a suitable class, which is followed in section 5 by the discussion of implementations in cyclic process networks. In section 6 we present an example in which the ERT model is used to model implementation of replication on a concrete architecture. In section 7 we discuss how the CA actions could be dealt with within the ERT based framework. The proofs of all the theorems stated in the paper, together with some auxiliary results, are included in the appendix.
2 Preliminaries

This section contains a brief introduction to that part of CSP which we will need to formulate our definitions and results.

2.1 Actions and traces

Communicating Sequential Processes (CSP) [1, 2, 3] 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, as in several other application-oriented papers, 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 $v$ to be exchanged between processes communicating over $b$. For every channel $b$, $\text{msg}_b$ is the message set of $b$, the set of all messages that can be exchanged on $b$. We define $\alpha b = \{b!v | v \in \text{msg}_b\}$ to be the alphabet of channel $b$. It is assumed that $\text{msg}_b$ 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 [3] (below $t, u$ are traces; $b, b', b''$ channels; $B_1, \ldots, B_n$ $B$ disjoint sets of channels; $A$ a set of actions; and $T, T'$ sets of traces):

- $\{\}$ is the empty trace;
- $t = \langle a_1, \ldots, a_n \rangle$ is the trace whose $i$-th element is $a_i$, and whose length, $|t|$, is $n$; we will denote $t[i] = a_i$ for $i \leq n$; if $n \geq 1$ then $\text{head}(t) = a_1$ and $\text{tail}(t) = \langle a_2, \ldots, a_n \rangle$;
- $t \circ u$ is the trace obtained by appending $u$ to $t$;
- $A^*$ is the set of all traces of actions from $A$;
- $T^*$ is the set of all traces $t = t_1 \circ \cdots \circ t_n$ ($n \geq 0$) such that $t_1, \ldots, t_n \in T$;
- $\text{Pref}(T)$ denotes the prefix-closure of $T$;
- $\leq$ denotes the prefix relation on traces;
- $t < u$ if $t \leq u$ and $t \neq u$;
- $t[b'/b]$ is a trace obtained from $t$ by replacing each action $b!v$ by $b'!v$;
- $t[B]$ is obtained by deleting from $t$ all the actions that do not occur on the channels in $B$; for example, $\langle b'^*3, b!1, b'^*2, b!3, b'^*6, b'^*2|\{b, b'\} = \langle b!1, b'^*3, b'^*6, b'^*2\rangle$;
- $t \downarrow b$ is the message sequence obtained by projecting $t$ on $b$ and then deleting the channel name; for example, $\langle b!1, b'^*2, b'^*3, b'^*6\rangle \downarrow b = \langle 1, 3 \rangle$;
- Let $T_i$, for $i \leq n$, be sets of traces over the channels in $B_i$; their shuffle, denoted by $T_1 T_2 \cdots T_n$, is the set of all traces $t$ over $B_1 \cup \cdots \cup B_n$ such that $t[B_i] \in T_i$, for all $i$;
- A mapping $f : T \to T'$ between non-empty sets of traces is monotonic if $t \leq u$ and $t, u \in T$ implies $f(t) \leq f(u)$ (for functions on vectors of traces, the prefix relation is defined component-wise); $f$ is strict if $f(\langle \rangle) = \langle \rangle$; 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 $\mathcal{Y}$ is subset-closed if $Y \subset X \in \mathcal{Y}$ implies $Y \in \mathcal{Y}$.  

4
2.2 Processes

We use the divergence model of CSP [2, 3] in which a process \( P \) is a triple \((\alpha P, \phi P, \delta P)\) where \( \alpha P \) — alphabet — is a non-empty finite set of actions, \( \phi P \) — failures — is a subset of \( \alpha P^* \times 2^{\alpha P} \), and \( \delta P \) — divergences — is a subset of \( \alpha P^* \). The conditions imposed on the three components are given below, where \( \tau P \) denotes the traces of \( P \), \( \tau P = \{ t \mid (t, R) \in \phi P \} \):

CSP1 \( \tau P \) is non-empty and prefix-closed.

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 \( a \in \alpha P \) is such that \( t \circ a \not\in \tau P \) then \((t, R \cup \{a\}) \in \phi P \).

CSP4 If \( t \in \delta P \) then \((t \circ a, R) \in \phi P \), for all \( a \in \alpha P^* \) 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. This facilitates a smooth definition of fixpoints used to give the semantics of recursively defined processes.

We will associate with \( P \) a set of channels, \( \text{chan} P \), and stipulate that the alphabet of \( P \) is that of \( \text{chan} P \). Thus, we shall be able to identify \( P \) with the triple \((\text{chan} P, \phi P, \delta P)\) 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 to the treatment of ERT are only the parallel composition of processes, hiding of the communication over a set of channels and renaming of channels. In the modelling of CA actions and examples we also use deterministic choice, \( P \parallel Q \), non-deterministic choice \( P \cap Q \), and prefixing, \( a \rightarrow P \). All the operators are formally defined in the appendix.

Parallel composition \( P \parallel Q \) 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 \( P_1 \parallel \cdots \parallel P_n \) 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 \( P \setminus B \) is a process that behaves like \( P \) with the actions occurring at the channels in \( B \) made invisible. Hiding is associative in that \((P \setminus B) \setminus B' = P \setminus (B \cup B')\).

Let \( P \) be a process with a channel \( b \in \text{chan} P \), and \( b' \) be a channel not in \( \text{chan} P \) such that \( \text{msg} b = \text{msg} b' \). Then \( P \parallel [b'/b] \) is a process that behaves like \( P \) except that each action \( bb'v \) is replaced by \( b'b'v \).

A crucial property [2] involving the parallel composition and hiding operators states that if \( P \) and \( Q \) are two processes and \( B \subseteq \text{chan} P \cap \text{chan} Q \) then \((P \setminus B) \parallel Q = (P \parallel Q) \setminus B \). 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 then define \( P_1 \odot \cdots \odot P_n \) to be the process obtained by taking the parallel composition of the processes and then hiding all interprocess communication, i.e. the process \((P_1 \parallel \cdots \parallel P_n) \setminus B \), where \( B \) is the set of channels shared by at least two different processes. We will call \( P_1 \odot \cdots \odot P_n \) a network.
Theorem 2.1 Let $P_1, \ldots, P_{k+l}$, where $k, l \geq 1$ be a network of processes. Then

$$P_1 \otimes \cdots \otimes P_k \otimes P_{k+1} \otimes \cdots \otimes P_{k+l} = (P_1 \otimes \cdots \otimes P_k) \otimes P_{k+1} \otimes \cdots \otimes P_{k+l}.$$ 

That is, a network can be obtained by first composing some of the processes into a subnetwork, and then composing the result with the remaining processes. In the failure model of CSP, where a process $P$ is identified with the pair $(\alpha P, \phi P)$, this property does not hold [2], whence the need for the more complicated divergence model.

We can partition the channels of a process $P$ into the input channels, $\text{in} P$, and output channels, $\text{out} P$. It is implicitly assumed that no two processes in a network have a common input channel or a common output channel. When composing processes, we will always assume that

$$\text{in} (P_1 \otimes \cdots \otimes P_n) = \bigcup_{i=1}^{n} \text{in} P_i - \bigcup_{i=1}^{n} \text{out} P_i \quad \text{and} \quad \text{out} (P_1 \otimes \cdots \otimes P_n) = \bigcup_{i=1}^{n} \text{out} P_i - \bigcup_{i=1}^{n} \text{in} P_i.$$ 

In the diagrams, an outgoing arrow labelled by $c$ indicates that $c$ is an output channel, and an incoming arrow labelled by $c$ indicates that $c$ is an input channel.

For a process $P$, let $\psi P = \{t \mid (t, \text{out} P) \in \phi P\}$. Intuitively, $\psi P$ comprises those traces of $P$ for which the process produced all the outputs for a given set of inputs.

2.4 Processes class employed

To facilitate the discussion, we will refer to three processes, $P, K$ and $L$, such that $\text{in} P = \{p_1, \ldots, p_m\}$, $\text{out} P = \{q_1, \ldots, q_n\}$, $\text{in} K = C$, $\text{out} K = D \cup E$, $\text{in} L = D \cup F$ and $\text{out} L = G$ [see figure 1]. It is assumed that the channel sets $E \cup D$ and $G$ are non-empty, and $P$ has at least one output channel ($n \geq 1$). In the case that $P$ has no input channels, $m = 0$, it will be called output-only. In the proofs we will always assume that the processes involved are not output-only. All the proofs can be easily adapted (and simplified) for output-only processes.

Let $U$ and $V$ be two non-empty prefix-closed sets of traces over respectively $\text{in} P$ and $\text{out} P$. Then $P$ is input-guarded w.r.t. $U$ and $V$ if, for every set of traces $T \subseteq \tau P$ satisfying $T \mid \text{in} P \subseteq U$,

\[ \text{IG1} \ T \mid \text{out} P \subseteq V. \]

\[ \text{IG2} \quad \text{If } T \text{ is infinite then so is } T \mid \text{in} P. \]

We denote this by $P \in \text{IG}(U, V)$, and if $U$ and $V$ comprise all traces over $\text{in} P$ and $\text{out} P$ we simply write $P \in \text{IG}$. For an output-only $P$ the definition reduces to requiring that $\tau P$ should be a finite subset of $V$. Input guardedness is compositional in the sense that if $T_C, T_D, T_E, T_F$ and $T_G$ are
prefix-closed sets of traces over respectively the channels $C$, $D$, $E$, $F$ and $G$, then $K \in IG(T_C, T_D \parallel T_E)$ and $L \in IG(T_D \parallel T_F, T_G)$ implies $K \circ L \in IG(T_C \parallel T_F, T_E \parallel T_G)$.

A class of base processes we shall consider is that of general input/output processes \cite{8}. These comprise processes $P$ which are input-guarded (i.e., $P \in IG$) and never refuse any input (i.e., $R \cap \text{ain} P = \emptyset$, for all $(t, R) \in \phi P$). Both restrictions are rather mild and a wide range of processes are GIO, e.g., Merge (defined in the next section) which combines with a FIFO policy messages received on its input channels into a single output stream. An output-only process $P$ is GIO if $\tau P$ is finite. The GIO process class is non-diverging and compositional, i.e., if $P \in \text{GIO}$ then $\delta P = \emptyset$; and if $K \in \text{GIO}$ and $L \in \text{GIO}$ then $K \circ L \in \text{GIO}$.

We also consider a subclass of GIO comprising functional processes $P$ with buffered input and output, as illustrated in figure 2: $P$ receives messages on the input channels, which are then kept in $m$ input buffers. Messages are removed from the input buffers by the computational unit $CU$ which works out and deposits the results in the $n$ output buffers. Such a process $P$ can be characterised by a function over vectors of traces \cite{4}. Let

$$f : (\alpha p_1)^* \times \cdots \times (\alpha p_m)^* \rightarrow (\alpha q_1)^* \times \cdots \times (\alpha q_n)^*$$

be a monotonic function. (For $m = 0$, $f$ is a constant vector of traces.) Then $P$ is a deterministic input/output process implementing $f$ if

\begin{align*}
\text{DIO1} & \quad \tau P = \{ t \mid \forall u \leq t : (u[q_1, \ldots, u[q_n] \leq f(u[p_1, \ldots, u[p_m]) \}
\text{DIO2} & \quad (t, \{ a \}) \in \phi P \Rightarrow t \circ \{ a \} \notin \tau P.
\end{align*}

We denote this by $P \in \text{DIO}$ (note that $\text{DIO} \subset \text{GIO}$). If $P$ has exactly one input channel and one output channel, and $f(t) = t[q_1/p_1]$, then it is an unbounded deterministic buffer, denoted by $\text{buffer}_{p_1q_1}$. DIO processes are compositional, i.e., if $K \in \text{DIO}$ and $L \in \text{DIO}$ then $K \circ L \in \text{DIO}$.

3 Extraction Patterns

3.1 Motivating example

Consider two processes, $\text{Sgl}$ and $\text{Buffer}$, as shown in figure 3(a). The former generates a single binary pulse, i.e., one belonging to $B = \{0, 1\}$, on its output channel $p$, then terminates; the latter is a buffer process forwarding signals received on its input channel. $\text{Sgl}$ is a GIO process, $\text{Buffer} = \text{buffer}_{p_2q_1}$ is a DIO process.

---

\textsuperscript{1} See proposition A.4(3) in the appendix.

\textsuperscript{2} See theorem A.5.

\textsuperscript{3} See theorem A.6.
Suppose that the signal transmission between the two processes has been implemented using two channels, \( r \) and \( s \), as shown in figure 3(b). The signal itself is now duplicated and the two copies sent along \( r \) and \( s \).

That is, \( Sgl_0 \) sends the duplicated signal, while \( Buffer_0 \) accepts a single copy and passes it on (possibly after a delay) ignoring the other one. The scheme clearly works as we have \( Sgl \odot Buffer = Sgl_0 \odot Buffer_0 \). Suppose now that the transmission of signals is imperfect and two types of faulty behaviour can occur:

\[
Sgl_1 = Sgl_0 \cap \text{stop} \quad \text{and} \quad Sgl_2 = Sgl_0 \cap \bigcap_{v \in B} r!v \to \text{stop}.
\]

\( Sgl_1 \) can break down completely, refusing to output any signals, while \( Sgl_2 \) can fail in such a way that although channel \( s \) is blocked \( r \) can still transmit the signal. \( Sgl_2 \) could be used to model the following realistic situation: in order to improve performance, a ‘slow’ channel \( p \) is replaced by two channels, a high-speed yet unreliable channel \( s \) and a slow but reliable backup channel \( r \). Since \( Sgl \odot Buffer = Sgl_2 \odot Buffer_0 \) but \( Sgl \odot Buffer \not= Sgl_1 \odot Buffer_0 \) it follows that \( Sgl_2 \) is much ‘better’ an implementation of the \( Sgl \) process than \( Sgl_1 \). We will now analyse 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 \( Sgl_2 \) can be thought of as adhering to the following two rules:
The transmissions over \( r \) and \( s \) are consistent.

Transmission over \( r \) is reliable, but there is no such guarantee for \( s \).

The output produced by \( Sgl_1 \) 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 \( Sgl \) and \( Sgl_2 \) we will employ an (extraction) mapping \( extr \) which for traces over \( r \) and \( s \) returns corresponding traces over \( p \). For example,

\[
\begin{align*}
(\epsilon) & \mapsto (\epsilon) \\
(r!0) & \mapsto (p!0) \\
(s!0) & \mapsto (p!0) \\
(s!1, r!1) & \mapsto (p!1) \\
(r!1, s!1) & \mapsto (p!1)
\end{align*}
\]

Note that the extraction mapping needs 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 \( Sgl_1 \) in the presence of faults since \( \tau Sgl = extr(\tau Sgl_1) = extr(\tau Sgl_2) \). What one also needs is an ability to relate the refusals of \( Sgl_1 \) with the possible refusals of the base process \( Sgl \). This, however, is much harder than relating traces. For suppose that we attempted to ‘translate’ the refusals of \( Sgl_2 \) using the extraction mapping. Then, we would have had

\[
(\epsilon, \{s!0\}) \in \partial Sgl_2 \quad \text{and} \quad extr((\epsilon, \{s!0\})) = ((\epsilon, \{p!0\}) \not\in \partial Sgl.
\]

This indicates that the crude extraction of refusals is not going to work. What we need is a more sophisticated device which in our case comes in the form of another mapping, \( ref \), constraining the possible refusals a process can exhibit after a given trace. This will help preventing, e.g., 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 \( e \) should not be blocked before the sender, \( Sgl_2 \), has sent a signal it \( Sgl_1 \) fails to satisfy a similar condition. For example, we will stipulate that \( ref((s!0)) \) must not comprise a refusal containing both \( r!0 \) and \( r!1 \). We will denoted by \( dom \) traces conveying information that can at least in principle be regarded as complete. According to R2, \( \langle s!0 \rangle \) and \( \langle s!1 \rangle \) will not belong to \( dom \).

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 can be extracted from the traces of its implementations.

### 3.2 Formal definition

An *extraction pattern* is defined for two non-empty sets of channels, \( B \) and \( B' \), respectively called *sources* and *targets*. It is a tuple \( ep = (dom, extr, ref, inv) \) satisfying the following:

**EP1** \( dom \) is a 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 targets.

**EP3** \( ref \) is a mapping defined for traces in \( Dom \); for every \( t \), \( ref(t) \) is a non-empty subset-closed family of proper subsets of \( aB \).
The extraction mapping is monotonic as receiving more information cannot decrease the current knowledge about the transmission. \( aB \not\in \text{ref}(t) \) means that for the unfinished communication \( t \) we do not allow the sender to refuse all possible transmission. From EP4 it follows that \( \text{dom} \) has at least the same cardinality as the traces over targets. Since \( \text{inv} \) is a trace homomorphism, it suffices to define it for single actions over the targets only.

The different components of the extraction patterns can be subscripted or primed to avoid ambiguity. Note that we do not mention explicitly the source and target channels. We assume these can always be retrieved from the domains of \( \text{extr} \) and \( \text{inv} \). Unless explicitly stated, different extraction patterns have disjoint sources and disjoint targets. A basic extraction pattern is one with a singleton target channel (basic extraction patterns were discussed in [8]).

Let \( \text{ep}_1 \) and \( \text{ep}_2 \) be two extraction patterns with the sources (targets) respectively \( B_1 \) and \( B_2 \) \((B'_1\) and \( B'_2\)). Then \( \text{ep} = \text{ep}_1 \oplus \text{ep}_2 \) is an extraction pattern with the sources \( B = B_1 \cup B_2 \) and targets \( B' = B'_1 \cup B'_2 \) such that the following are satisfied:

\[
\text{dom} = \text{dom}_1 \parallel \text{dom}_2
\]

\[
\text{extr} = \begin{cases}
\text{extr}(\langle \rangle) & = \langle \rangle \\
\text{ref}(t) & = \{ R \in aB | R \cap \alpha B_1 \in \text{ref}_1(t|B_1) \lor R \cap \alpha B_2 \in \text{ref}_2(t|B_2) \}
\end{cases}
\]

\[
\text{extr}(t \circ \langle a \rangle) = \begin{cases}
\text{extr}(t) \circ u_1 & \text{if } a \in \alpha B_1 \land \text{extr}_1(w|B_1) = \text{extr}_1(w|B_1) \circ u_1 \\
\text{extr}(t) \circ u_2 & \text{if } a \in \alpha B_2 \land \text{extr}_2(w|B_2) = \text{extr}_2(w|B_2) \circ u_2
\end{cases}
\]

\[
\text{inv}(a) = \begin{cases}
\text{inv}_1(a) & \text{if } a \in \alpha B'_1 \\
\text{inv}_2(a) & \text{if } a \in \alpha B'_2
\end{cases}
\]

It is not difficult to see that \( \text{ep} \) is well-defined, i.e., it is indeed an extraction pattern, such that \( \text{Dom} = \text{Dom}_1 \parallel \text{Dom}_2 \). For the proof it is essential that extraction mappings are strict and monotonic. \( \parallel \) is both associative and commutative. It provides an important way of building complex extraction patterns from simpler ones, in particular the basic extraction patterns.

### 3.3 Examples of extraction patterns

We have already discussed a kind of ‘fail-stop’ extraction pattern for the example in figure 3. In general, to define a fail-stop extraction pattern, we assume that the sources are channels \( B \) and the target is a singleton channel \( b \) (the message sets of all the channels involved are the same). Moreover, \( \text{NF} \subseteq B \) is a non-empty set of reliable channels. Those in \( B - \text{NF} \) may be unreliable, but even in the worst case may only fail by doing nothing [12]. These assumptions are captured by the extraction pattern denoted by \( \text{fs}(b, B, \text{NF}) \) and defined in the following way (below \( p \) is a fixed channel in \( \text{NF} \)).

\[
\text{dom} = \{ t | \forall q \in B \forall r \in \text{NF} : t \downarrow q \leq t \downarrow r \}
\]

\[
\text{ref}(t) = \{ R | R \not\in \text{NF} \not\subseteq R \}
\]

\[
\text{extr}(t) = \max \{ \langle t[b]/q \rangle | q \in B \}
\]

\[
\text{inv}(b|v) = \langle plv \rangle.
\]

\[\text{EP4} \quad \text{inv} \text{ is a trace homomorphism from traces over the targets to traces in } \text{Dom}; \text{ for every trace } w \text{ over the targets, } \text{extr}(\text{inv}(w)) = w.\]
Note that $\text{Dom} = \{ t \mid \forall q, r \in C : t \downarrow q \leq t \downarrow r \ \vee \ t \downarrow r \leq t \downarrow q \}$.

The second basic extraction pattern is one which can be used to model systems employing majority voting. Here we have the same sources and targets as before, with the reliable source channels being in majority, $|NF| > \frac{1}{2} |B|$. Channels not in $NF$ are in no way constrained. We define the majority voting extraction pattern $\text{mv}(b, B, NF)$ in the following way (below $p_1, \ldots, p_{|NF|}$ is a fixed enumeration of the channels in $NF$).

\[
\begin{align*}
\text{dom} &= \left\{ t \mid \forall q, r \in NF : t \downarrow q = t \downarrow r \right\} \\
\text{ref}(t) &= \left\{ R \mid aNF \not\subseteq R \right\} \\
\text{extr}(t) &= \max \left\{ u \mid \frac{1}{2} |B| < \left| \{ q \in B \mid a \downarrow b \leq t \downarrow q \} \right| \right\} \\
\text{inv}(qv) &= \langle p_1qv, \ldots, p_{|NF|}qv \rangle.
\end{align*}
\]

Note that $\text{Dom} = \{ t \mid \forall q, r \in NF : t \downarrow q \leq t \downarrow r \ \vee \ t \downarrow r \leq t \downarrow q \}$.

In the formal treatment we shall need a special simple extraction pattern which relates source to target communication in a one-to-one (identity) manner. An identity extraction pattern for a channel $b$, $\text{id}_b$, is one for which $B = B' = \{ b \}$, $\text{dom} = \text{Dom} = \alpha b \ast$, $\text{extr}(t) = \text{inv}(t) = t$ and $\text{ref}(t) = \{ R \mid \alpha b \not\subseteq R \}$. For a set of channels $B'' = \{ b_1, \ldots, b_k \}$, $\text{id}_{B''} = \text{id}_{b_1} + \cdots + \text{id}_{b_k}$. (Note that $\text{id}_b = \text{fs}(b, \{ b \}, \{ b \}) = \text{mv}(b, \{ b \}, \{ b \})$.)

Our previous examples might give an impression that extraction patterns can only model implementations lending itself to unidirectional interpretation. This is, in fact, not the case. Consider again the example of figure 3(a) and the following two processes (see figure 4(a)):

\[
\begin{align*}
\text{Sgl}_3 &= \Box_{v \in B} r!v \rightarrow (s!ack \rightarrow \text{stop} \Box s!nak \rightarrow r!v \rightarrow \text{stop}) \\
\text{Buffer}_1 &= B_1^{(1)} \\
B_1^{(t)} &= \begin{cases}
\Box_{v \in B} r!v \rightarrow (s!ack \rightarrow B_1^{(q!v)}) & \text{if } t = \langle \rangle \\
\text{head}(t) \rightarrow B_1^{\text{ack}(t)} \\
\Box_{v \in B} r!v \rightarrow (s!ack \rightarrow B_1^{(q!v)}) & \text{if } t \neq \langle \rangle 
\end{cases}
\end{align*}
\]

We have $\text{Sgl}_3 \oplus \text{Buffer}_1 = \text{Sgl} \oplus \text{Buffer}$. The processes employ a simple bi-directional protocol: a signal sent over channel $r$ is acknowledged by the receiver as successful (using $s!ack$) or failed (using $s!nak$) in which case the sender will retransmit. This second transmission always succeeds. These assumptions can be captured by the extraction pattern twice defined in the following way:

\[
\begin{align*}
\text{dom} &= \left\{ \langle r!0, s!ack \rangle, \langle r!0, s!nak, r!0 \rangle, \langle r!1, s!ack \rangle, \langle r!1, s!nak, r!1 \rangle \right\} \\
\text{ref}(t \circ u) &= \begin{cases}
\{ R \mid u \not\subseteq R \} & \text{if } t \in \text{dom} \land u \in \{ \langle \rangle, \langle r!0, s!0 \rangle, \langle r!1, s!0 \rangle \} \\
2^{|w|} & \text{if } t \in \text{dom} \land u \in \{ \langle r!0 \rangle, \langle r!1 \rangle \} \\
\langle \rangle & \text{if } t = \langle \rangle 
\end{cases} \\
\text{extr}(t) &= \text{extr}(w) \circ \langle q!v \rangle \text{ if } t = w \circ \langle r!v, s!ack \rangle \in \text{dom} \lor t = w \circ \langle r!v, s!nak, r!v \rangle \in \text{dom} \\
\text{inv}(q!v) &= \{ \langle r!v, s!ack \rangle \}.
\end{align*}
\]
Notice that not all interesting extraction patterns can be factored into a set of basic extraction patterns through the $\oplus$ operation. Consider the following pair of base processes:

$$\begin{align*}
\text{Sgls} &= \Box_{v \in B} p!v \rightarrow \text{stop} \parallel \Box_{v \in B} r!v \rightarrow \text{stop} \\
\text{Merge} &= M^{(1)} \\
M^{t'} &= \begin{cases} \\
\text{head}(t) \rightarrow M^{\text{tail}(t)} & \text{if } t = () \\
\bigcup \{ i \in \{p, r\}, v \in B \} \Box_{v} b!v \rightarrow M^{(q!v)}(t) & \text{if } t \neq ()
\end{cases}
\end{align*}$$

The Sgls process generates two binary signals on its two output channel which are then combined in the FIFO manner by the Merge process (see figure 4(b)). Suppose that in the actual implementation an internal error can occur and the channel $r$ can fail and its signal is then appended to that originally sent on channel $p$ (below the two channels are renamed as $p_0$ and $r_0$). The receiving process has to take this into account:

$$\begin{align*}
\text{Sgls}_0 &= \bigcup_{v \in B} p_0!v \rightarrow \text{stop} \parallel \bigcup_{v \in B} r_0!v \rightarrow \text{stop} \\
\text{Merge}_0 &= M_0^{(1)} \\
M_0^{t'} &= \begin{cases} \\
\text{head}(t) \rightarrow M_0^{\text{tail}(t)} & \text{if } t = () \\
\bigcup \{ i \in \{p_0, r_0\}, v \in B \} \Box_{v} b!v \rightarrow M_0^{(q!v)}(t) & \text{if } t \neq ()
\end{cases}
\end{align*}$$

We have $\text{Sgls} \odot \text{Merge} = \text{Sgls}_0 \odot \text{Merge}_0$ and the communication between Sgls and Merge can be described by the extraction pattern **double** defined in the following way:

$$\begin{align*}
\text{dom} &= (a_{p_0} \cup a_{r_0})^* \\
\text{ref}(t) &= \{ R | a_{p_0} \cup a_{r_0} \not\subseteq R \} \\
\text{extr}(t \circ \langle a \rangle) &= \begin{cases} \\
\text{extr}(t) \circ \langle p!v \rangle & \text{if } a = p_0!v \\
\text{extr}(t) \circ \langle r!v \rangle & \text{if } a = r_0!v \\
\text{extr}(t) \circ \langle p_0!v, r_0!v \rangle & \text{if } a = p_0_0!v_0!v \\
\langle p_0!v \rangle & \text{if } a = p!v \\
\langle r_0!v \rangle & \text{if } a = r!v
\end{cases} \\
\text{inv}(a) &= \{ \}
\end{align*}$$

Notice that double cannot be decomposed onto two extraction patterns. For there is a trace over $p_0$ generated by Sgls, which can be used to extract information about the intended communication on
both \( p \) and \( r \). This, however, means that the targets of one of the extraction patterns would have to include both \( p \) and \( r \), which is not possible (the targets of composed extraction patterns are non-empty and disjoint).

4 Extraction in Acyclic Networks

Suppose that we implemented the base \( \text{GI} \Theta \) process \( P \) using another process \( Q \). The correctness of the implementation will be expressed in terms of two extraction patterns, \( \text{ep} \) and \( \text{ep}' \). The former (with sources \( \text{in } Q \) and targets \( \text{in } P \)) will be used to relate the communication on the input channels of \( P \) and \( Q \); the latter (with sources \( \text{out } Q \) and targets \( \text{out } P \)) will serve a similar purpose for the output channels. There are four main 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 \( \text{ep} \), then it should be possible to interpret the projection on the output channels by \( \text{ep}' \) (see \( \text{WI}1 \) and \( \text{IG}1 \)). Secondly, when connected to another process and supplied with a valid input (i.e. belonging to \( \text{Dom} \)), \( Q \) should not introduce divergence (this rules out an infinite uninterrupted communication on the output channels — see \( \text{WI}1 \) and \( \text{IG}2 \)) and should not refuse ‘proper’ input (this rules out refusals which might lead to a deadlock with another process that provides input to \( Q \) and whose refusals are constrained by \( \text{ep} \) — see \( \text{WI}2 \)). Thirdly, if \( Q \) is to receive input from process \( S \) and send its output to process \( W \) and after \( S \) has communicated all the input to \( Q \), \( Q \) cannot cause a deadlock in the communication with \( W \) until all the results produced have been sent to \( W \) (\( \text{WI}3 \)). Finally, we ensure that the purely functional behaviour of \( P \) (i.e. that in terms of traces) can be realised by \( Q \) (\( \text{WI}4 \)). Formally, \( Q \) is a weak implementation of \( P \) if the following hold.

\( \text{WI}1 \) \( Q \in \text{IG}(\text{Dom}, \text{Dom}') \).

\( \text{WI}2 \) If \( (t, R) \in \phi Q \) is such that \( t[\text{in } Q \in \text{Dom} \text{ then } \text{a } \text{in } Q - R \not\in \text{ref } (t[\text{in } Q) \).

\( \text{WI}3 \) If \( (t, R) \in \phi Q \) is such that \( t[\text{in } Q \in \text{dom} \text{ and } \text{a } \text{out } Q \cap R \not\in \text{ref}'(t[\text{out } Q) \text{ then } t[\text{out } Q \in \text{dom}' \text{ and extr}_{\text{ep}, \text{ep}'}(t) \in \psi P \).

\( \text{WI}4 \) \( \text{inv}_{\text{ep}, \text{ep}'}(\tau P) \subseteq \tau Q \).

We denote this by \( Q \in \text{WI}(P, \text{ep}, \text{ep}') \). It is not difficult to see that \( \text{WI}(P, \text{id}_{\text{in } P}, \text{id}_{\text{out } P}) \subseteq \text{GI} \Theta \), and that the process \( P \) is its own weak implementation based on identity extraction patterns, \( P \in \text{WI}(P, \text{id}_{\text{in } P}, \text{id}_{\text{out } P}) \). Note also that \( Q \in \text{WI}(P, \text{id}_{\text{in } P}, \text{id}_{\text{out } P}) \) does not necessarily imply \( Q = P \) even if \( P \in \text{DI} \Theta \). For an output-only \( P \) we denote \( Q \in \text{WI}(P, \text{ep}') \) if \( \text{WI}4 \) holds and the following two conditions are satisfied:

\( \text{WI}1' \) \( \tau Q \) is finite and included in \( \text{Dom}' \).

\( \text{WI}3' \) If \( (t, R) \in \phi Q \) is such that \( \text{a } \text{out } Q \cap R \not\in \text{ref}'(t) \) then \( t \in \text{dom}' \) and \( \text{extr}_{\text{ep}'}(t) \in \psi P \).

4.1 Examples revisited

It can be checked that the following hold (below \( \text{fs} = \text{fs}(p, \{r, s\}, \{r\}) \)):

- \( \text{Sgl}_1 \not\in \text{WI}(\text{Sgl}, \text{fs}) \)
- \( \text{Sgl}_1 \in \text{WI}(\text{Sgl}, \text{twice}) \)
- \( \text{Buffer}_0 \in \text{WI}(\text{Buffer}, \text{fs}, \text{id}_0) \)
- \( \text{Merge}_0 \in \text{WI}(\text{Merge}, \text{double}, \text{id}_1) \)

13
Process $\text{Sgl}_1$ is not a weak implementation of $\text{Sgl}$ since it fails to satisfy $\text{WI}'$. In fact, $\text{Sgl}_1 \not\in \text{WI}(\text{Sgl}, \text{ep}')$ for any extraction pattern $\text{ep}'$ since $(\cdot), \text{ar} \cup \text{as} \in \text{Sgl}_1$ and $\text{ar} \cup \text{as} \not\in \text{ref}'(\cdot)$ (see EP3). On the other hand, $(\text{ext}'(\cdot), \text{out}\text{Sgl}) = (\cdot, \text{ap}) \not\in \phi \text{Sgl}$, contradicting $\text{WI}'$. Consider two more examples, Buffer$_2$ and Buffer$_3$:

$$\begin{align*}
\text{Buffer}_2 &= \text{Buffer}_0 \ | \ r!0 \rightarrow s!1 \rightarrow b \\
b &= q!0 \rightarrow b \\
\text{Buffer}_3 &= \text{Buffer}_3 \ | \ \text{stop}_s
\end{align*}$$

We have $\text{Buffer}_2, \text{Buffer}_3 \in \text{WI}(\text{Buffer}, \text{fs}, \text{id}_s)$. Although Buffer$_2$ can produce infinite uninterupted output, this does not matter as it can only happen for the input of the form $(r!0, s!1)$ which is not in the domain of $\text{fs}$. Buffer$_3$ is also acceptable despite $s$ being blocked as channel $r$ is reliable.

### 4.2 Compositionality and realizability

To support modular treatment of networks of implementations we will use two properties, compositionality and realizability. The former means that the parallel composition of two weak implementations is a weak implementation of the parallel composition of the base processes, and is formulated as theorem 4.1. The latter amounts to saying that it is possible to use implementations in place of the base processes.

**Theorem 4.1** Let $K$ and $L$ be two GIÖ processes as in figure 1, and let $c, d, e, f$ and $g$ be extraction patterns whose targets are respectively the channel sets $C, D, E, F$ and $G$.\footnote{If any of the sets is empty, the corresponding extraction pattern is simply left out.} If $M \in \text{WI}(K, c, d \oplus e)$ and $N \in \text{WI}(L, d \oplus f, g)$ then $M \otimes N \in \text{WI}(K \otimes L, c \oplus f, e \oplus g)$. \hfill $\square$

Realisability is captured in the next theorem. In what follows, for two GIÖ processes $P$ and $Q$, we will denote $Q \preceq P$ if in $P = \text{in} Q$, $\text{out} P = \text{out} Q$, $\tau Q = \tau P$ and $\psi Q \subseteq \psi P$.

**Theorem 4.2** If $Q \in \text{WI}(P, \text{id}_p, \text{id}_{\text{out} p})$ then $Q \preceq P$. \hfill $\square$

Why can this be regarded as a result expressing an adequate notion of realizability? First, note that given $Q' \in \text{WI}(P, \text{ep}, \text{ep}')$, where $\text{ep}$ and $\text{ep}'$ are general extraction patterns, compositionality in many situations allows $Q \in \text{WI}(P, \text{id}_p, \text{id}_{\text{out} p})$ to be constructed, e.g., using the extractors and disturbers discussed in the next section (for a GIÖ process $P$ this is always possible). As for the $\preceq$ relation, think of an environment for processes $Q$ and $P$ which is represented by an arbitrary GIÖ process $\text{Rcv}$ which is to receive the results produced by $Q$ and $P$, as shown in figure 5. It turns out that $Q \preceq P$ implies:\footnote{See proposition A.8.}

$$\begin{align*}
\tau(Q \otimes \text{Rcv}) &= \tau(P \otimes \text{Rcv}) \\
\phi(Q \otimes \text{Rcv}) &\subseteq \phi(P \otimes \text{Rcv}).
\end{align*}$$

The former property means that as far as functionality is concerned, $Q \otimes \text{Rcv}$ and $P \otimes \text{Rcv}$ are equivalent processes. The latter property means that $Q \otimes \text{Rcv}$ is more deterministic a process than $P \otimes \text{Rcv}$ in the sense of [3]. This makes $Q \otimes \text{Rcv}$ as good as $P \otimes \text{Rcv}$ (and possibly much better) process to be used in the actual implementation. Note that the underlying philosophy of [3] is to allow as non-deterministic processes as possible for specification, but as deterministic as possible ones for actual implementation; our approach is thus in line with that advocated there. Hence theorem 4.2 captures an adequate notion of realizability.
It is worth observing that the above realisability result can be strengthened if we assume that \( t \cdot \langle a \rangle \not\in \tau P \), for all \( t \in \psi P \) and \( a \in \text{out } P \). This means that \( P \) can refuse to generate any output only if its functional specification in terms of traces rules this out. We denote \( P \in \text{GIO}' \) in such a case. Note that \( \text{DIO} \subseteq \text{GIO}' \subseteq \text{GIO} \) and \( \text{Merge} \in \text{GIO}' \); however, \( \text{GIO}' \) processes are not compositional. It then follows\(^7\) that if \( P \in \text{GIO}' \) then \( Q \odot \text{Rcv} = P \odot \text{Rcv} \).

### 4.3 Networks of processes

The compositionality result of theorem 4.1 is easily extended to process networks. Indeed, let \( P_1, \ldots, P_k \) be a network of base \( \text{GIO} \) processes, and let \( Q_i \) be a weak implementation of \( P_i \), for all \( i \). Using theorem 4.1 and induction it is straightforward to prove that the network \( Q_1, \ldots, Q_k \) represents a weak implementation of the network \( P_1, \ldots, P_k \). However, the possible topology of the latter is restricted by the hypothesis (illustrated in figure 1) of theorem 4.1. The base network must be ayclic in the sense that there should be no processes \( P_{k_1}, P_{k_2}, \ldots, P_{k_l} = P_{k_1} \) such that \( \text{out } P_{k_i} \cap \text{in } P_{k_{i+1}} \neq \emptyset \), for \( i < l \).

The notion of weak implementation cannot be used to deal with cyclic networks of base processes as it is possible to introduce an extra divergence which is not present in the base network. We illustrate this point on the following example.

Let \( \text{Buf} = \text{buffer}_{pq} | \text{stop}_p \) be a process with input channel \( p \) and output channels \( q \) and \( r \). Intuitively, \( \text{Buf} \) behaves like \( \text{buffer}_{pq} \) with a dormant output channel \( r \). Moreover, let

\[
\begin{align*}
\text{BUF} & = \text{Buf}_1 |\|\text{Buf}_2 |\|\text{Buf}_3 \\
\text{Buf}_i & = \text{Buf}[p_i/p][q_i/q][r_i/r] \quad (i = 1, 2, 3) \\
\text{buf} & = \text{buffer}_{q_1p_1} |\|\text{buffer}_{q_2p_2} |\|\text{buffer}' \\
\text{buffer}' & = q_3!0 \rightarrow \text{buffer}_{q_3p_3}
\end{align*}
\]

We also need three extraction patterns \( \text{mv}_b = \text{mv}(b, \{b_1, b_2, b_3\}, \{b_1, b_2\}) \), for \( b \in \{p, q, r\} \). It can be checked that \( \text{BUF} \) and \( \text{buf} \) are weak implementations of \( \text{Buf} \) and \( \text{buffer}_{qp} \):

\[
\begin{align*}
\text{BUF} & \in \text{WI}(\text{Buf}, \text{mv}_p, \text{mv}_q, \text{mv}_r) \\
\text{buf} & \in \text{WI}(\text{buffer}_{qp}, \text{mv}_q, \text{mv}_p).
\end{align*}
\]

Yet, when \( \text{BUF} \) and \( \text{buf} \) are combined together, the implementation relation is not preserved

\[
\text{BUF} \odot \text{buf} \not\in \text{WI}(\text{Buf} \odot \text{buffer}_{qp}, \text{mv}_r) = \text{WI}(\text{stop}_r, \text{mv}_r)
\]

\(^7\) See proposition A.9.
since $\text{BUF} \odot \text{buf}$ diverges at the very beginning (this follows from $\{(y_3,0,0,0)\}^* \subseteq \tau(\text{BUF}||\text{buf})$). To cope with cyclic networks a stronger notion of correct implementation is needed. A possible approach has been presented in [6], a more refined one is introduced in section 5.

### 4.4 DIO processes

For deterministic input/output processes the results from the previous section can be strengthened. To begin with, we can do without the last component of the extraction pattern, $\text{inv}$, and WI4. Moreover, the realisability result has a particularly pleasant form.

With the same assumptions as before, a process $Q$ is a semi-weak implementation of a DIO process $P$ if WI1, WI2 and WI3 are satisfied. We denote this by $Q \in \text{sWI}(P, \text{ep, ep'})$. The compositionality result for $\text{sWI}$ holds; its wording is exactly the same as that of theorem 4.1 with WI changed to $\text{sWI}$, and $K, L \in \text{DIO}$. The relationship between a DIO process $P$ and one of its semi-weak implementation $Q' \in \text{sWI}(P, \text{ep, ep'})$ can be characterised in the following way. First we observe that it is possible to place $Q'$ in an environment such that the resulting process is a semi-weak implementation of $P$ based on identity extraction patterns. This environment is built using two special processes called a disturber, Dist, and an extractor, Extr (see figure 6). Intuitively, the former provides noisy but sufficiently redundant input for the implementation, the latter can interpret the results it produces. Formally, Dist and Extr are processes satisfying the following:

\[
\begin{align*}
\text{Dist} & \in \text{sWI}(\text{buffer}_{p_1/p_1} \cdots \text{buffer}_{p_m/p_m}, \text{id}_{\text{Dist}, \text{ep}}) \\
\text{Extr} & \in \text{sWI}(\text{buffer}_{q_1/q_1} \cdots \text{buffer}_{q_n/q_n}, \text{ep}', \text{id}_{\text{Extr}}).
\end{align*}
\]

From theorem 2.1, the compositionality results for DIO processes, and the fact that DIO processes are insensitive to buffering their input and output channels [8], it follows that $Q'$ composed with the disturber and extractor is a semi-weak implementation of $P$:

\[
\text{Dist} \odot Q' \odot \text{Extr} \in \text{sWI}(P[p_1/p_1] \cdots [p_m/p_m][q_1/q_1] \cdots [q_n/q_n], \text{id}_{\text{Dist}, \text{id}_{\text{Extr}}}).
\]

From this we immediately obtain that $Q \in \text{sWI}(P, \text{id}_{\text{in}}, \text{id}_{\text{out}})$, where

\[
Q = (\text{Dist} \odot Q' \odot \text{Extr})[p_1/p_1] \cdots [p_m/p_m][q_1/q_1] \cdots [q_n/q_n].
\]

That is, provided that we know how to construct disturbers and extractors, $Q'$ can be transformed into a weak implementation of $P$ based on identity extraction patterns. It now follows that if we take the receiver process $\text{Rcv}$ as in figure 5, $\text{Rcv} \in \text{DIO}$, and connect it to both $P$ and $Q$ then $Q \odot \text{Rcv} = P \odot \text{Rcv}$. The resulting realisability result is therefore stronger than that for GIO processes.

---

*See corollary A.10.
*See proposition A.12.
and similar to that for GI\(\text{O}'\) processes. Moreover, if we take the receiver being a simple array of buffers, 
\(\text{Rcv} = \text{buffer}_{q_1/r_1} \sqcup \cdots \sqcup \text{buffer}_{q_n/r_n}\), then it is almost immediate to see that 
\((Q \otimes \text{Rcv})[q_1/r_1] \cdots [q_n/r_n] = P\). In this way we have shown that starting from an arbitrary semi-weak implementation of \(P \in \text{DIO}\) it is possible to extract process \(P\) exactly, provided that we know how to implement buffer processes.

5 Extraction in Cyclic Networks

To deal with cyclic networks we need another pair of base processes \(K\) and \(L\), as shown in figure 7(a).

The first problem we face is that \(K \neq L\) will not, in general, be a GI\(\text{O}\) process even though both \(K\) and \(L\) are. Our treatment is therefore restricted to well-behaved cases (i.e., those which do not lead to divergence). We say that the processes \(K\) and \(L\) are compatible if for every infinite set of traces \(T \subseteq \tau(K \sqcup L)\), the set \(T[\text{in}(K \otimes L)]\) is also infinite. It follows that if \(K\) and \(L\) are compatible GI\(\text{O}\) processes then \(K \otimes L \in \text{GI\(\text{O}\)}\).

With the same assumptions as those used in the definition of WI, \(Q\) is a strong implementation of \(P\) if the following hold.

\(\text{SI1}\) If \(T \subseteq \tau Q\) and \(T[\text{in} Q] \subseteq \text{Dom}\) then
(a) \(T[\text{out} Q] \subseteq \text{Dom}'\).
(b) If \(T\) is infinite then so is \(\text{extr}(T[\text{in} Q])\).
(c) \(\text{extr}_{\text{ep} \sqcup \text{ep}'}(T) \subseteq \tau P\).

\(\text{SI2}\) If \((t, R) \in \phi Q\) is such that \(t[\text{in} Q] \in \text{Dom}\) then \(\text{cin} Q - R \not\in \text{ref}(t[\text{in} Q])\).

\(\text{SI3}\) If \((t, R) \in \phi Q\) is such that \(t[\text{in} Q] \in \text{Dom}\) and \(\text{out} Q \cap R \not\in \text{ref}'(t[\text{out} Q])\) then
(a) \(t[\text{out} Q] \in \text{dom}'\)
(b) \(t \in \text{dom}_{\text{ep} \sqcup \text{ep}'} \Rightarrow \text{extr}_{\text{ep} \sqcup \text{ep}'}(t) \in \psi P\).

\(\text{SI4}\) \(\text{inv}_{\text{ep} \sqcup \text{ep}'}(\tau P) \subseteq \tau Q\).

We denote this by \(Q \in \text{SI}(P, \text{ep}, \text{ep}')\). The main difference between this and the previous definition is that IG2 in the definition of input-guardedness has been replaced by a stronger condition SI1(b) which can be interpreted as input-guardedness relative to extraction. Moreover, we have slightly strengthened WI3 and added SI1(c) to explicitly state that extracted valid traces of \(Q\) are traces of \(P\).

\(^{10}\) See proposition A.13.
For an output-only process the last definition reduces to $\text{WI}^{3'}$, $\text{SI}^4$ and the requirement that $\tau Q$ be a finite subset of $\text{Dom}'$ such that $\text{extr}_{ep}(\tau Q) \subseteq \tau P$. It is worth noting that $\text{extr}_{ep}(\tau Q) \subseteq \tau P$ can be derived from $\text{WI}^{3'}$ and $\text{ep}'$ being monotonic. Thus, for output-only base processes $P$, the notions of weak and strong implementations coincide. This is not true in general. For example,

$$\text{BUF} \not\in \text{SI}(\text{Buf, mv}_p, \text{mv}_q, \text{mv}_r) \quad \text{and} \quad \text{buf} \not\in \text{SI}(\text{buffer}_p, \text{mv}_q, \text{mv}_p)$$

where \text{BUF} and \text{buf} are processes used to show that \text{WI} is insufficient to deal with cyclic networks of processes. However, we do have an expected (and immediate) result that being strong implementation implies being also weak implementation. Consequently, the realizability result for strong implementation follows from theorem 4.2. Compositionality is given below.

**Theorem 5.1** Let $K$ and $L$ be two compatible \text{GIO} processes as in figure 7(a), and let $c, d, e, f, g$ and $h$ be extraction patterns whose targets are respectively the channel sets $C, D, E, F, G$ and $H$. If $M \in \text{Sl}(K, c \oplus h, d \oplus e)$ and $N \in \text{Sl}(L, d \oplus f, g \oplus h)$ then $M \oplus N \in \text{Sl}(K \oplus L, c \oplus f, e \oplus g)$. \hfill $\Box$

In this way we have shown that strong implementation is a suitable notion for dealing with cyclic networks of processes under the proviso that the base processes are compatible — more precisely, if we can decompose the base network down to single processes in such a way that at each stage compatibility is satisfied — an assumption which roughly amounts to saying that their parallel composition does not introduce any divergence.

### 5.1 Relationship between weak and strong implementability

The two notions of implementability we proposed are quite similar. And although \text{Sl} is in general stronger, there are several cases for which \text{Sl} is the same as \text{WI}. We have already mentioned that this holds for output-only base processes. In fact, a stronger result holds:

11 If $\text{ep}$ is an extraction pattern such that $\text{dom}$ is prefix-closed and $\text{extr}^{-1}(w)$ is always finite (note that identity extraction patterns satisfy these two conditions), then $\text{WI}(P, \text{ep, ep}') = \text{Sl}(P, \text{ep, ep}')$. It is also possible to obtain an interesting and useful (see section 6) result that two processes, a weak and a strong implementation can be composed together to form a strong implementation. The connectivity of the base processes, however, has to be restricted.

**Theorem 5.2** Let $K$ and $L$ be two \text{GIO} processes as in figure 7(b), and let $c, d, e$ and $g$ be extraction patterns whose targets are respectively the channel sets $C, D, E$ and $G$. If $M \in \text{Sl}(K, c, d \oplus e)$ and $N \in \text{Sl}(L, d, g)$ then $M \oplus N \in \text{Sl}(K \oplus L, c, e \oplus g)$. \hfill $\Box$

We also define **semi-strong implementation** for \text{DIO} process $P$, denoted $Q \in \text{sSl}(P, \text{ep, ep}')$, by assuming that \text{SI1-S13} hold. The realizability result for \text{sSl} follows immediately from that for \text{sWI}, as we clearly have $\text{sSl} \subset \text{sWI}$. Compositionality and mixed compositionality (in combination with \text{sWI}) are also satisfied.

---

11 See proposition A.15.
12 See corollaries A.14 and A.16.
5.2 Algebraic properties of implementation

This section provides a quick look at the process algebraic properties of implementation relations. Their relevance stems from the fact that they can be used in the future development of verification techniques for the ERT model.

Let $\text{impl}$ be any of the $\text{WI}, \text{sWI}, \text{SI}$ and $\text{sSI}$. Then, for any two base process $K$ and $L$ with disjoint channels the following holds (below we make the usual assumptions about the extraction patterns).

$$M \parallel N \in \text{impl}(K \parallel L, c \oplus f, e \oplus g) \iff M \in \text{impl}(K, c, e) \land N \in \text{impl}(L, f, g).$$

A similar conclusion can be drawn for processes combined using the non-deterministic choice operator: Let $\text{simp}$ be $\text{SI}$ or $\text{sWI}$, and $\text{in}Q = \text{in}Q'$ and $\text{out}Q = \text{out}Q'$. Then

$$Q \cap Q' \in \text{simp}(P, ep, ep') \iff Q \in \text{simp}(P, ep, ep') \land Q' \in \text{simp}(P, ep, ep') \land \text{inv}_{ep \oplus ep'}(\tau P) \in \tau Q \cup \tau Q'.$$

Although it is interesting to see whether other CSP operators can lead to a similar characterisation of implementation relations, from the point of view of dealing with fault tolerant systems, the results involving the non-deterministic choice operator $\cap$ are of special interest. For the possibility of the occurrence of an internal fault in process $P$ will often be modelled by a process $P_{\text{faulty}} = P \cap P_{\text{error}}$, where $P$ represents a correctly behaving system, and $P_{\text{error}}$ its abnormal execution. For example, we can simplify the proof that $\text{Sgl}_1$ fails to be a weak implementation of $\text{Sgl}$ by observing that $\text{Sgl}_1 = \text{Sgl} \cap \text{stop}$ and $\text{stop}$ cannot possibly a semi-weak implementation of $\text{Sgl}$ (since, by EP3, $\text{stop}$ can never satisfy $\text{WI2}$). Similarly, the correctness proof for $\text{Sgl}_2$ can be simplified by breaking it down onto two subproofs.

6 Implementing Replication

We now address the problem of proving the correctness of a standard replication scheme. Here an important issue is the interplay between the fault assumptions and the extraction strategy. We will present the treatment for majority voting and fail-stop processes.

The architecture of the base process $P$ we consider is shown in figure 8. It is composed of two subprocesses, $\text{AU} \in \text{GIO}$ and $\text{FU} \in \text{DIO}$. The former represents the arbiter unit which receives messages on input channels $p_1, \ldots, p_m$ and then forwards them, along channels $r_1, \ldots, r_i$, to the functional unit, $\text{FU}$. We independently replicate the arbiter unit, and then assemble it together with $N$ copies of the functional unit (some of them possibly faulty), as shown in figure 8.

The arbiter unit is implemented by process $\text{au}$. The communication on the input channels of $\text{au}$ and $\text{AU}$ is related using an arbitrary extraction pattern $\text{ep}$. The output produced by $\text{au}$ is more constrained. It is sent along $i-N$ channels, $r_{ij}$, $\text{msg} r_i = \text{msg} r_{ij}$, which are intended to carry the replicated output feeding the $N$ copies of the functional unit, $\text{fu} = \text{fu}_1 \parallel \cdots \parallel \text{fun}_N$.

The replicated arbiter process, $\text{au}$, produces output which can be used for the majority voting. Without loss of generality, we assume that the first $M$ processes, $\frac{1}{2}N < M$ and the associated channels, are correct. Moreover,
We obtain a similar result as before. Again, it follows from the compositionality theorems and $fu \in WI(FU, MF, MV')$, the proof of which is straightforward. That is, we have the following (below Impl is any of WI, sWI and sSL).

$$au \in Impl(AU, ep, MV) \Rightarrow au \otimes fu \in Impl(P, ep, MV')$$
$$au \in SL(AU, ep, MV) \Rightarrow au \otimes fu \in sSL(P, ep, MV')$$

For the fail-stop extraction patterns we make the same assumptions as before, except for $M > \frac{1}{2}N$, and with the following additions:

- $Fs = fs_1 \oplus \cdots \oplus fs_i$ where $fs_i = fs(r_i, \{r_1, \ldots, r_iN\}, \{d_{i1}, \ldots, r_{iM}\})$.
- $Fs' = fs'_1 \oplus \cdots \oplus fs'_n$ where $fs'_j = fs(q_j, \{q_1, \ldots, q_jN\}, \{q_{j1}, \ldots, q_{jM}\})$.
- For $i > M$, $fu_i$ is any process such that $\tau fu_i \subseteq \tau FU[r_i/r_1] \cdots [r_i/r_i][q_i/q_1] \cdots [q_i/q_n]$.

We obtain a similar result as before. Again, it follows from the compositionality theorems and $fu \in WI(FU, Fs, Fs')$ (below Impl is any of WI, sWI and sSL).

$$au \in Impl(AU, ep, Fs) \Rightarrow au \otimes fu \in Impl(P, ep, Fs')$$
$$au \in SL(AU, ep, Fs) \Rightarrow au \otimes fu \in sSL(P, ep, Fs')$$

We have presented the ERT model and illustrated the ideas behind its design by means of several examples. The model is general in that allows one to relate the behaviour of two systems which
may have different interfaces and/or different patterns of information exchange flowing through these interfaces. ERT also supports a natural and expressive way of specifying fault assumptions in process behaviours. As a result, we envisage that one should be able to apply ERT to a variety of distributed system’s architectures and, in particular, different fault tolerant techniques, provided that the base processes providing high level specifications adhere to the GIO rules. In essence, this means that they communicate by asynchronous message passing.

7 Modelling CA Actions

We have shown that ERT can be used to capture the correctness of a fault tolerant architecture based on N-modular redundancy. In this section, we will discuss what could be a possible ERT based approach to the modelling of systems built using the CA action structuring mechanism. Before describing our approach, we briefly review the main features of ERT and the extent to which they match those of CA actions.

- **Interaction between processes is through message-passing with point-to-point communication.** The former fits well with the object-oriented framework within which CA actions are defined. The latter might cause some problems if, for instance, message broadcast was to be modelled; in such a case one could add special processes multiplying outgoing messages and forwarding them to several receivers.

- **Base processes are GIO.** The definition of a GIO process means that ERT can handle networks of implementations whose abstract specifications communicate by means of asynchronous message passing (synchronous message passing could be implemented, if necessary, on top of an underlying asynchronous message passing mechanism). It is expected that a substantial part of CA actions based systems would fall into this category.

- **Realisability by extraction and compositionality.** This crucial property of the ERT model formalises a fundamental characteristics of fault tolerant systems, namely what it does mean for an implementation to be acceptable in the presence of faults. The parameterisation by general extraction patterns of the various notions of implementation relation makes ERT suitable for a wide range of concrete fault tolerant techniques. This is complemented by compositionality, which allows each of the processes in a network to be considered separately and proved correct with respect to an abstract specification.

We therefore conclude that the main features of the ERT model are compatible with those expected of many potential applications of CA actions. What still remains to be discussed is the atomicity of operations performed on external (shared) objects. At this point we do not see a direct need to introduce such a requirement explicitly into the formalisation of CA actions. This corresponds to the view that atomicity is a feature of a communication protocol between CA actions and external objects, and not the property of either of these two objects alone. Thus dealing with it could be done at the level of the network of base processes where appropriate formal specification of communication between CA actions and shared objects could be designed and proved correct.

In the rest of this section, we first present an abstract CA action specification which conforms to the existing examples, such as those specified in [16] using an informal programming notation, as well as general rules specified elsewhere. We then propose two ways in which the CA action structuring technique could be supported by the model presented earlier in this paper. One of the crucial issues which has emerged from our investigation of ERT modelling of CA actions is whether the roles should
be modelled as computational parts of systems modelling CA actions, or as computational parts of threads calling CA actions. We shall discuss these two alternatives separately in sections 7.2 and 7.3, respectively. Finally, we discuss in more detail part of the CA action design presented in [16].

7.1 A formalisation of CA action

In its basic form, a CA action can be specified as a process, $\text{CA}_{\text{spec}}$, shown in figure 9. The behaviour of $\text{CA}_{\text{spec}}$ can be described as follows. Channels $\text{call}_1, \ldots, \text{call}_m$ are intended to carry messages with some input data (parameters) requesting an access to $\text{CA}_{\text{spec}}$, each such message corresponding to starting of a role within the CA action. Intuitively, $\text{CA}_{\text{spec}}$ waits for a set of $m$ messages, each such message arriving along a different channel $\text{call}_i$, before accepting them for further processing. It is, however, generally not the case that any message set will be accepted as a valid call; more precisely, $\text{CA}_{\text{spec}}$ will only accept some of these potential input sets. It is therefore assumed that there is a non-empty set of valid inputs, $\text{Valid}$. Each $V \in \text{Valid}$ is a set of $m$ messages, $\{\text{call}_1!v_1, \ldots, \text{call}_m!v_m\} \in \text{ocall}_1 \cup \cdots \cup \text{ocall}_m$. It is also assumed that for such a $V$ there is a non-empty set $\text{return}(V)$ which comprises all possible outcomes of the processing of messages in $V$. Each element $U$ of $\text{return}(V)$ is a set of $m$ messages over the channels $\text{return}_i$, i.e., $U = \{\text{return}_1!v_1, \ldots, \text{return}_m!v_m\} \in \text{oreturn}_1 \cup \cdots \cup \text{oreturn}_m$. Note that it is implicitly assumed that an execution of the CA action produces a single result for each of the participating threads. This is a simplifying assumption which could easily be relaxed. Note also that the modelling of an aborted execution of the CA action can be modelled by $U = \{\text{return}_1!\text{abort}, \ldots, \text{return}_m!\text{abort}\} \in \text{return}(V)$.

Since not all combinations of messages on channels $\text{call}_i$ are admissible, the CA action has $m$ additional channels $\text{acc}_i$ which are used to inform each of the threads attempting to enter $\text{CA}_{\text{spec}}$ whether or not the message has been accepted. $\text{CA}_{\text{spec}}$ keeps rejecting the calls from the threads until a message is received whose value is acceptable; the policy employed is one which accepts a message if it is not inconsistent with those already accepted. All messages received on $\text{call}_i$ are explicitly accepted or rejected. Note that if all possible combinations of messages arriving on channels $\text{call}_i$ were acceptable, then the channels $\text{acc}_i$ can be removed, thus simplifying the specification of $\text{CA}_{\text{spec}}$ (as in the example discussed at the end of this section).

After assembling a set of messages $V$ in $\text{Valid}$, the CA action produces (non-deterministically, as $\text{CA}_{\text{spec}}$ is a specification rather than an implementation, see our earlier discussion on realisability) one of the possible valid outcomes in $\text{return}(V)$.

The specification of $\text{CA}_{\text{spec}}$ is given by $\text{CA}_{\text{spec}} = \text{CA}_1^{V \vdash t_1} \cdots \text{CA}_m^{V \vdash t_m}$. In the definition, we use parameterised process symbols $\text{CA}^{V \vdash t_m}_i$ where $V$ is the current set of accepted messages, and each $t_i$ is the current sequence of messages still to be sent along channel $\text{acc}_i$. 

**Figure 9**: A base CA action process
Figure 10: CA action $CA^I_{\text{spec}}$ with internal roles

$$CA^I_{V^I_{\text{spec}}} = \begin{cases} 
\prod_{t \in \text{return}(V)} \text{call}^{t_1 \cdots t_m} & \text{if } |V| = m \\
\bigwedge_{i \leq m, v \in \text{msg call}} \text{call}^{t_i} \to CA' & \text{if } |V| < m \land t_1 \cdots t_m = \langle \rangle \\
\bigwedge_{i \neq j} \text{head}(t_i) \to CA_{\text{inr}}^{t_1 \cdots t_i (j) \cdots t_m} & \text{if } |V| < m \land t_1 \cdots t_m \neq \langle \rangle \\
\bigwedge_{i \leq m, v \in \text{msg call}} \text{call}^{t_i} \to CA' & \text{otherwise}
\end{cases}$$

where $CA'$ is given by the following formula:

$$CA' = \begin{cases} 
CA_{V \cup \{\text{call}^i \}}^{t_1 \cdots t_m} & \text{if } V \cap \text{acc}^i = \emptyset \land \exists V' \in \text{Valid} : V \cup \{\text{call}^i \} \subseteq V' \\
CA_{V^I_{\text{spec}}}^{t_1 \cdots t_m} & \text{otherwise}
\end{cases}$$

and process $\text{call}^{t_1 \cdots t_m}$ is defined thus:

$$\text{call}^{t_1 \cdots t_m} = \begin{cases} 
\text{call}^{t_1 \cdots t_m} & \text{if } V = \emptyset \\
\bigwedge_{a \in V} a \to \text{ca}^{t_1 \cdots t_m} & \text{if } V \neq \emptyset \land t_1 \cdots t_m = \langle \rangle \\
\bigwedge_{i \neq j} \text{head}(t_i) \to \text{ca}^{t_1 \cdots t_i (j) \cdots t_m} & \text{if } V \neq \emptyset \land t_1 \cdots t_m \neq \langle \rangle \\
\bigwedge_{i \leq m, v \in \text{msg call}} \text{call}^{t_i} \to \text{ca}^{t_1 \cdots t_i (acc) \cdots t_m} & \text{otherwise}
\end{cases}$$

One can check that $CA_{\text{spec}}$ is a GIO process, and thus can be used as a base process. Note that $CA_{\text{spec}}$ does not have any channels to communicate with external shared objects. This is a simplifying assumption which could be removed. To do so, one would assume that $CA_{\text{spec}}$ has a set of input channels, and a set of output channels to connect to the external objects. The handling of the communication on these channels would depend on the particular protocol used by the application considered. In general, no message to the external objects would be output before a $V$ set is accepted, or after $CA_{\text{spec}}$ starts producing the results in $U \in \text{return}(V)$. Moreover, the choice of $U$ would depend on the sequence of messages exchanged between $CA_{\text{spec}}$ and the external objects. The modelling of these features is straightforward, yet strongly dependent on the application studied.

### 7.2 CA action with internal roles

The first architectural model of a CA action as a base process $CA^I_{\text{spec}}$ is shown in figure 10. The external connectivity of $CA^I_{\text{spec}}$ is the same as that of $CA_{\text{spec}}$. Internally, $CA^I_{\text{spec}}$ comprises processes $\text{In}$, $\text{Out}$, $\text{role}_1$, ..., $\text{role}_m$, and possibly other processes $P_1$, ..., $P_n$ representing nested CA actions and
internal objects (not shown in figure 10). There can be channels linking together the role processes \( \text{role}_1, \ldots, \text{role}_m \) as well as channels linking \( P_1, \ldots, P_n \) with \( \text{role}_1, \ldots, \text{role}_m \).

It is assumed that \( CA_{\text{spec}}^I = CA_{\text{spec}} = \cap \cap \text{role}_1 \cdots \text{role}_m \cap P_1 \cdots P_n \cap \text{Out} \) and that each process in the network \( CA_{\text{spec}}^I \) is \( \text{GlO} \). The \( \text{ln} = \text{ln}_V^{t_1, \ldots, t_m} \) process can be defined thus:

\[
\text{ln}_V^{t_1, \ldots, t_m} = \begin{cases} 
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } |V| = m \\
\text{Reset}!\text{yes} \rightarrow \text{ln}_V^{t_1, \ldots, t_m} & \text{if } |V| < m \land t_1 \circ \cdots \circ t_m = \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } |V| < m \land t_1 \circ \cdots \circ t_m \neq \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } |V| = m \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } |V| < m \land t_1 \circ \cdots \circ t_m = \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } |V| < m \land t_1 \circ \cdots \circ t_m \neq \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } |V| = m \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } |V| < m \land t_1 \circ \cdots \circ t_m = \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } |V| < m \land t_1 \circ \cdots \circ t_m \neq \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } |V| = m \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } |V| < m \land t_1 \circ \cdots \circ t_m = \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } |V| < m \land t_1 \circ \cdots \circ t_m \neq \langle \rangle \\
\end{cases}
\]

where

\[
\text{ln}_V^{t_1, \ldots, t_m} = \begin{cases} 
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V = \emptyset \land t_1 \circ \cdots \circ t_m = \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V = \emptyset \land t_1 \circ \cdots \circ t_m \neq \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V = \emptyset \land t_1 \circ \cdots \circ t_m = \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V = \emptyset \land t_1 \circ \cdots \circ t_m \neq \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V = \emptyset \land t_1 \circ \cdots \circ t_m = \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V = \emptyset \land t_1 \circ \cdots \circ t_m \neq \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V = \emptyset \land t_1 \circ \cdots \circ t_m = \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V = \emptyset \land t_1 \circ \cdots \circ t_m \neq \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V = \emptyset \land t_1 \circ \cdots \circ t_m = \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V = \emptyset \land t_1 \circ \cdots \circ t_m \neq \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V = \emptyset \land t_1 \circ \cdots \circ t_m = \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V = \emptyset \land t_1 \circ \cdots \circ t_m \neq \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V = \emptyset \land t_1 \circ \cdots \circ t_m = \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V = \emptyset \land t_1 \circ \cdots \circ t_m \neq \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V = \emptyset \land t_1 \circ \cdots \circ t_m = \langle \rangle \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V = \emptyset \land t_1 \circ \cdots \circ t_m \neq \langle \rangle \\
\end{cases}
\]

In the above, \( \text{ln}' \) is given by:

\[
\text{ln}' = \begin{cases} 
\text{ln}_V^{t_1, \ldots, t_m} & \text{if } V \cap \text{aIn}_i = \emptyset \land \exists V' \in \text{Valid} : (V[\text{call}_1/\text{inr}_1] \cdots [\text{call}_m/\text{inr}_m]) \cup \{\text{call}_i\} \subseteq V' \\
\text{ln}_V^{t_1, \ldots, t_m} & \text{otherwise} \\
\end{cases}
\]

Note that process \( \text{ln} \) receives requests for accessing the CA action, and after assembling a set of \( m \) valid messages it forwards accepted requests to the role processes, \( \text{role}_i \), along the channels \( \text{inr}_i \). The \( m \) messages on channels \( \text{out}_i \) resulting from a single run of the CA action are collected by process \( \text{Out} \) which sends them along channels \( \text{return}_i \). The role of the \( \text{reset} \) channel is to inform \( \text{ln} \) that \( \text{Out} \) has completed the task of sending the current result set. The \( \text{Out} = \text{Out}_V \) process can be defined as follows:

\[
\text{Out}_V = \begin{cases} 
\text{out}_V & \text{if } |V| = m \\
\text{out}_V & \text{if } |V| < m \\
\text{Reset}!\text{yes} \rightarrow \text{Out}_V & \text{if } |V| = m \\
\text{out}_V & \text{if } |V| < m \\
\text{Reset}!\text{yes} \rightarrow \text{Out}_V & \text{if } |V| = m \\
\text{out}_V & \text{if } |V| < m \\
\end{cases}
\]

Note that \( \text{ln} \) and \( \text{Out} \) defined above are only examples of what might be appropriate specifications; given a particular application, their definition could be suitably modified. Having specified \( CA_{\text{spec}}^I \), we
can apply directly to it the general results obtained for the ERT model. In particular, we can choose specific extraction patterns and implementation relations corresponding to the actual implementation techniques, which may represent refinements as well as fault tolerance measures (e.g., the replication scheme discussed in the previous section). Recall that this may, and usually will, lead to the change of channels used by the processes.

### 7.3 CA action with external roles

The structuring of the CA action shown in figure 10 implicitly assumes that each role \( \text{role}_i \) is implemented as part of the programming construct modelling the CA action. However, as some of the existing examples of CA suggest, it may be the case that the roles are part of the specification of programming constructs modelling threads, i.e., processes that send messages on channels \( \text{call}_i \) and receive messages on channels \( \text{return}_i \). In such a case, the architecture of figure 10 is no longer appropriate. We then can use an alternative structuring of \( \text{CA}_{\text{spec}} \), as shown in figure 11 (note that the diagram depicts only one out of \( m \) role processes).

It is assumed \( \text{CA}_{\text{spec}} \) comprises internal CA actions as well as internal objects shared by the roles (all such processes are assumed to be \( \text{GIO} \)). Consistency with the basic CA action specification is ensured by requiring that \( \text{CA}_{\text{spec}} = \text{CA}_{\text{spec}}^\text{II} \odot \text{role}_1 \odot \cdots \odot \text{role}_m \). The subsequent application of the ERT approach would follow the same pattern as previously, i.e., one would choose concrete extraction patterns and implementation functions to establish the correctness of a low level implementation.

### 7.4 A Case Study Fragment

In this section we shall apply the modelling approach outlined in section 7.1 to a significant fragment of a real life CA action based application: the automated production cell described in [16]. The cell comprises several physical components, controlled by software structured in accordance with the CA action paradigm. The portion we shall concentrate on consists of a conveyor belt feeding a table with metal plates (in [16] these will later be forged in a press and forwarded to a deposit).

According to the architectural choices of [16], the software controlling each physical device is to be executed by two processes, serving the device proper and its sensors, respectively. Thus the system fragment we describe is essentially a network of four processes \( \text{FeedBelt}, \text{FeedBeltSensor}, \text{Table} \) and \( \text{TableSensor} \) interacting through CA action \( \text{LoadTable} \). Interaction with the environment takes place through CA actions \( \text{LoadPlate} \) and \( \text{UnloadTable} \). Figure 12 provides a simplified description of the system architecture in that, unlike Figure 9, it does not show the channels \( \text{call}_i, \text{return}_i \) and \( \text{acc}_i \) through which a generic process \( i \) would interact with a CA action. In fact, channel \( \text{acc}_i \) will not be needed here since the case study [16] simply abstracts from exception handling issues. In the interest of clarity, indexed channels \( \text{call}_i \) and \( \text{return}_i \) will be replaced by structured channel names like,
e.g., `call.LoadPlate.FB`, where the second component is the name of the CA action, and the third one conveniently identifies the participating process (in the example `FB` stands for `FeedBelt`).

The CSP code below employs some additional minor but useful conventions. The notation `b ? v → P(v)`, where `P(v)` is a process with state modelled explicitly by `v`, stands for the deterministic choice `∀ e ∈ msg  b? e v → P(v)`. Similarly, `b! v → P(v)` stands for the non-deterministic choice `∃ e ∈ msg  b! e v → P(v)`. Finally, in the formal descriptions we will omit the handling of unexpected incoming messages. This can be incorporated in a standard fashion as it has often been done earlier (cf. process `In` in Section 7.2).

At the abstract level, CA actions are viewed in [16] as ‘black boxes’ solely characterized by their external behaviour. In our ERT framework, this amounts to rendering a CA action as a base process only to be accessed through `call` and `ret` channels (cf. section 7.1). Base processes exhibit a simple cyclic behaviour.

Process `FeedBelt` acquires a plate through the CA action `LoadPlate`, passes it to the table by participating into `LoadTable` and restarts. Note that `LoadTable` is also passed the `feedBeltActuator` object through which it will effectively affect the motion of the feed belt.

```
FeedBelt(feedBeltActuator) =
    call.LoadPlate.FB! → ret.LoadPlate.FB, plate →
    call.LoadTable.FB!(plate, feedBeltActuator) → ret.LoadTable.FB! →
    FeedBelt(feedBeltActuator)
```

Process `FeedBeltSensor` provides CA action `LoadTable` with the object `photoElectricSensor`. This is intended to model the belt hardware sensor, and its state is accordingly specified, as in [16], to be internally chosen by the hardware.

```
FeedBeltSensor =
    call.LoadTable.FBS!(photoElectricSensor) → ret.LoadTable.FBS! →
    FeedBeltSensor
```

Likewise, process `TableSensor` supplies actions `LoadTable` and `UnloadTable` with objects corresponding to the three sensors that equip the table device.

```
TableSensor =
    call.LoadTable.TS!(bottomSwitchSensor, tableAngleSensor) → ret.LoadTable.TS! →
    call.UnloadTable.TS!(topSwitchSensor, tableAngleSensor) → ret.UnloadTable.TS! →
    TableSensor
```

![Figure 12: System architecture.](image-url)
Finally, process $\text{Table}$ participates to CA action $\text{LoadTable}$ by providing the $\text{tableActuator}$ object and receiving a plate, which is then passed on to the external environment through action $\text{UnloadTable}$.

$\text{Table}(\text{tableActuator}) =$

$\text{call}\cdot\text{LoadTable}.\text{T}!\text{tableActuator} \rightarrow \text{ret}\cdot\text{LoadTable}.\text{T}\cdot\text{plate} \rightarrow$

$\text{call}\cdot\text{UnloadTable}.\text{T}!(\text{plate, tableActuator}) \rightarrow \text{ret}\cdot\text{UnloadTable}.\text{T}! \rightarrow$

$\text{Table}(\text{tableActuator})$

At the implementation level, each CA action $\text{call}/\text{return}$ pair in base process definitions gets replaced by a CSP code block containing invocations of lower level CA actions and (non-atomic) operations on local objects such as, e.g., channels. This transformation (which could be easily defined formally by means of macro-expansion and CSP sequential composition [1]) yields the implementations for the device base processes. The implementation for the CA action $\text{LoadTable}$ base process is the composition of the newly introduced local objects and lower level (base) CA actions.

The implementation level CSP code refining $\text{call}/\text{return}$ pairs for $\text{LoadTable}$ is easy to write by simple inspection of the corresponding code in [16]. The extraction patterns under which the intended implementation relations hold are also straightforward to define. We shall simply give here the extraction functions, from which $\text{call}/\text{return}$ refinements and remaining extraction pattern components can be easily inferred.

\[
\begin{align*}
\langle \text{call} \cdot \text{LoadTable}.\text{FB!(plate, feedBeltActuator)} , \text{ret} \cdot \text{LoadTable}.\text{FB} \rangle & \mapsto \langle \text{call} \cdot \text{Wait.FB} , \text{ret} \cdot \text{Wait.FB} , \text{call} \cdot \text{MovePlate.FB!feedBeltActuator} , \\
& \quad \text{ret} \cdot \text{MovePlate.FB} , \text{channel}.\text{in}.\text{T!plate} \rangle \\
\langle \text{call} \cdot \text{MovePlate.FBS!photoElectricSensor} , \text{ret} \cdot \text{MovePlate.FBS} \rangle & \mapsto \langle \text{call} \cdot \text{LoadTable}.\text{FBS!photoElectricSensor} , \text{ret} \cdot \text{LoadTable}.\text{FBS} \rangle \\
\langle \text{call} \cdot \text{MoveTableDown.TS!bottomSwitchSensor} , \text{ret} \cdot \text{MoveTableDown.TS} , \\
& \quad \text{call} \cdot \text{RotateTable.TS!(tableAngleSensor, 0)} , \text{ret} \cdot \text{RotateTable}.\text{TS} \rangle & \mapsto \langle \text{call} \cdot \text{LoadTable}.\text{TS!(bottomSwitchSensor, tableAngleSensor)} , \text{ret} \cdot \text{LoadTable}.\text{TS} \rangle \\
\langle \text{call} \cdot \text{MoveTableDown.T!tableActuator} , \text{ret} \cdot \text{MoveTableDown.T} , \\
& \quad \text{call} \cdot \text{RotateTable.T!tableActuator} , \\
& \quad \text{call} \cdot \text{Wait.T} , \text{ret} \cdot \text{Wait.T} , \text{channel}.\text{plate}.\text{changeState('Table')} \rangle & \mapsto \langle \text{call} \cdot \text{LoadTable}.\text{T!tableActuator} , \text{ret} \cdot \text{LoadTable}.\text{T!plate} \rangle
\end{align*}
\]

Acknowledgements

The authors are indebted to Luigi Mancini for his contribution made in the course of work from which ERT has been derived. The work of the first author was supported by ESPRIT Long Term Research Project 20072 on “Design for Validation (DeVa)”. He would also like to thank the members of the DeVa team at Newcastle, Brian Randell, Alexander Romanovsky, Robert Stroud, Ian Welch and Avelino Zorzo, for several discussions on the formalisation of the semantics of CA actions. In particular, we would like to thank Jie Xu for his detailed comments on the draft version of this report. The work of the second author was supported by the Italian MURST.

References


A Appendix

A.1 Formal definitions of CSP selected operators

The operations on processes we use are defined in the following way:

$$\text{chan } (P || Q) = \text{chan } P \cup \text{chan } Q$$

$$\delta (P || Q) = \{ \tau \circ a \mid (t \text{[chan } P, t[\text{chan } Q]) \in (\tau P \times \delta Q) \cup (\delta P \times \tau Q) \}$$

$$\phi (P || Q) = \{(t, R \cup S) \mid (t \text{[chan } Q, S) \in \phi Q \} \cup \delta (P || Q) \times 2^{\phi (P || Q)}$$

$$\text{chan } (P \setminus B) = \text{chan } P - B$$

$$\delta (P \setminus B) = \{ (t, t[\text{chan } (P \setminus B)) \circ a \mid t \in \delta P \cup \exists a_1, a_2, \ldots \in \alpha B \forall n \geq 1 : t \circ \langle a_1, \ldots, a_n \rangle \in \tau P \}$$

$$\phi (P \setminus B) = \{(t \text{[chan } (P \setminus B), R) \mid (t, R \cup \alpha B) \in \phi P \} \cup \delta (P \setminus B) \times 2^{\phi (P \setminus B)}$$

$$\text{chan } P[b/b'] = \text{chan } P - \{b'\} \cup \{b\}$$

$$\delta P[b/b'] = \{ t[b/b'] \mid t \in \delta P \}$$

$$\phi P[b/b'] = \{(t[b/b'], R[b/b']) \mid (t, R) \in \phi P \}$$

$$\text{chan } (a \rightarrow P) = \text{chan } P$$

$$\delta (a \rightarrow P) = \{ \langle a \rangle \circ t \mid t \in \delta P \}$$

$$\phi (a \rightarrow P) = \{(\langle a \rangle \circ t, R) \mid (t, R) \in \phi P \} \times 2^{\phi P - \{a\}}$$

$$\text{chan } (P \parallel Q) = \text{chan } P$$

$$\delta (P \parallel Q) = \delta P \cup \delta Q$$

$$\phi (P \parallel Q) = \{(\{\}, R) \mid (\{\}, R) \in \phi P \cap \phi Q \} \cup \{(t, R) \mid t \neq \{\} \land (t, R) \in \phi P \cap \phi Q \}$$

$$\text{chan } (P \cap Q) = \text{chan } P$$

$$\delta (P \cap Q) = \delta P \cup \delta Q$$

$$\phi (P \cap Q) = \phi P \cup \phi Q.$$ 

In the above, $B$ is a proper subset of $\text{chan } P$; $b \not\sim \text{chan } P$ and $b' \in \text{chan } P$ are channels with the same message sets; $R[b/b']$ is $R$ with each $b'$ or changed to $b$; $a$ is an action in $aP$; in the last two definitions $\text{chan } P = \text{chan } Q$. Note that the divergences of $\delta (P \setminus B)$ could have been defined as in [3], where

$$\delta (P \setminus B) = \{ t[\text{chan } (P \setminus B) \circ a \mid t \in \delta P \cup \exists a_1, u_1, a_2, \ldots \in \alpha B^+ \forall n \geq 1 : t \circ \langle a_1, \ldots, a_n \rangle \in \tau P \}.$$ 

We also use $\text{stop}_B$ (or simply $\text{stop}$ if $B$ is clear from the context) to denote a deadlocked process with the channel set $B$, $\text{stop}_B = (B, 0, 0)$. 

A.2 General properties of processes

**Proposition A.1** [8] Let $B$ and $B'$ be two non-empty finite sets of channels, $B \subseteq B'$. If $T$ is an infinite prefix-closed set of traces over $B'$ such that $\{ t[\{ (B' - B) \mid t \in T \}$ is a finite set then there is $t \in T$ and $a_1, a_2, \ldots \in \alpha B$ satisfying $t \circ \langle a_1, \ldots, a_k \rangle \in T$, for all $k \geq 1$. 

**Proposition A.2** Let $(t, R) \in \phi (P \setminus B)$ be such that $t \not\in \delta (P \setminus B)$. Then there is $(w, R \cup \alpha B) \in \phi P$ such that $w \not\in \delta P$ and $w \text{[chan } (P \setminus B) = t.$

**Proof.** Follows directly from the definition of $P \setminus B$. 

29
**Proposition A.3** Let $P \in 1G(U, V)$ and $t \in \delta P$. Then $t \in P \not\subset U$.

**Proof.** Suppose $t \in P \in U$. From CSP4 and $\text{out} P \neq \emptyset$ it follows that there are $a_1, a_2, \ldots \in \text{out} P$ such that $t_k = t \circ \langle a_1, \ldots, a_k \rangle \in \tau P$, for all $k \geq 1$. Define $T = \{t_k | k \geq 1\}$. Then $T \in P = \{t \in P\}$, contradicting $1G2$.

### A.3 Basic compositionality results

In the proofs that follow, we will use $X$ to denote the sources, and $X$ to denote the targets, of an extraction pattern $x$, for $x \in \{c, d, e, f, g, h\}$. We will also use $Z$ to denote the channel set of a process $Z$, for $Z \in \{I, J, K, L, M, N, O, S\}$, where $I = K\parallel L, J = K \otimes L, S = M\parallel N$ and $O = M \otimes N$. The union of sets of channels, say $C \cup D$, will be simply denoted as $CD$, and when denoting the composition of extraction patterns, such as $e \oplus f$, we will leave out the symbol $\oplus$.

The following diagram may be useful when reading the proofs of proposition A.4, and theorems A.5 and 4.1.

```
\begin{center}
\begin{tikzpicture}
  \node (C) at (0,0) {$C$};
  \node (K) at (2,0) {$K$};
  \node (D) at (4,0) {$D$};
  \node (L) at (6,0) {$L$};
  \node (G) at (8,0) {$G$};
  \draw[->] (C) -- (K);
  \draw[->] (K) -- (D);
  \draw[->] (D) -- (L);
  \draw[->] (L) -- (G);
  \draw[->] (C) -- (E);
  \draw[->] (C) -- (F);
  \end{tikzpicture}
\end{center}
```

**Proposition A.4** Let $T_C, T_D, T_E, T_F$ and $T_G$ be prefix-closed sets of traces over respectively the channels $C, D, E, F$ and $G$. If $K \in 1G(T_C, T_D, T_E)$ and $L \in 1G(T_D, T_F, T_G)$ are processes as in figure 1, then the following hold.

1. $t \in \tau(K\parallel L) \land t \内陆(C \cup F) \in T_C \parallel T_F \Rightarrow t \in T_C \parallel T_D \parallel T_F \parallel T_D \parallel T_G.$
2. $t \in \delta(K \otimes L) \land t \内陆(C \cup F) \not\subset T_C \parallel T_F.$

**Proof.** Let $V = \{t \in \alpha I^* | t[CF] \in T_C \parallel T_F\}$. Clearly, $V$ is prefix-closed. First, we prove that:

1. $t \in \tau I \cap V \Rightarrow t \not\subset \delta I$.

Suppose that $t \in \delta I \cap V$. Then there is $u \in \delta I$ such that $u \subseteq t$ and at least one of the following holds: (i) $u[K] \in \delta K$ and $u[L] \in \tau L$, or (ii) $u[K] \in \tau K$ and $u[L] \in \delta L$. Clearly, $u \in V$ since $V$ is prefix-closed. Then (ii) contradicts proposition A.3 since $(u[K])[C = u[C] \in T_C$, so (ii) must be satisfied. Hence, by $(u[K])[C \in T_C$ and $1G1$ for $K$, $(u[K])[D \in T_D$. Thus $(u[L])[DF = u[DF \in T_D \parallel T_F$. This and $u[L] \in \delta L$ produces a contradiction with proposition A.3. Hence (1) holds.

To prove A.4(1), suppose that $t \in \tau I \cap V$. By (1) and the definition of parallel composition, $t[K] \in \tau K$ and $t[L] \in \tau L$. Together with $1G1$ for $K$ and $L$, applied in this order, this yields $t[K] \in T_C \parallel T_E \parallel T_D$ and $t[L] \in T_D \parallel T_F \parallel T_G$. Hence A.4(1) holds.

To prove A.4(2), suppose that $t \in \delta I \cap V$. Then there is $u \in \tau I$ such that $u[\land \subseteq t$ and at least one of the following holds: (iii) $u \in \delta I$, or (iv) there are $a_1, a_2, \ldots \in \alpha I$ such that $u[k = u \circ \langle a_1, \ldots, a_k \rangle \in \tau I$, for all $k \geq 1$. Clearly, $u \in V$ as both $T_C$ and $T_F$ are prefix-closed. Hence (iii) contradicts (1). Moreover, (iv) leads to a contradiction in the following way. Since, for all $k \geq 1$, $a_k \in V$ (as $u[K] \parallel CF = u[CF$ we obtain, by (1), that $u_k \not\subset \delta I$, for all $k \geq 1$. Thus, by the definition of parallel composition,
Proposition A.5 If $P \in \text{GI}O$ then $\delta P = \emptyset$, and if $K, L \in \text{GI}O$ are as in figure 1, then $K \parallel L \in \text{GI}O$.

Proof. That $\delta P = \emptyset$ follows from proposition A.3. From proposition A.4(3) it follows that $J \in \text{IG}$. Suppose that $(t, R) \in \phi J$. By proposition A.4(2), $t \notin \delta I$. Hence from proposition A.2 it follows that there is $(w, R \cup aD) \in \phi I$ such that $w \notin \delta I$ and $w[J] = t$. Thus, by the definition of parallel composition, there are $R_1$ and $R_2$ such that $R \cup aD = R_1 \cup R_2$, $(w[K], R_1) \in \phi K$ and $(w[L], R_2) \in \phi L$. Hence, by the definition of GI processes, $R_1 \cap aC = \emptyset$ and $R_2 \cap aF = \emptyset$. Thus $R \cap aCF = \emptyset$. As a result, $J$ is a GI process.

Theorem A.6 [8] If $K, L \in \text{DI}O$ are as in figure 1, then $K \parallel L \in \text{DI}O$.

A.4 Properties of the relation $\preceq$}

Let $P, Q$ and Recv be GI processes with channels as indicated in figure 5. In the proofs of propositions A.8, A.9 and A.12, we will denote $U = Q \parallel \text{Recv}$ and $W = P \parallel \text{Recv}$. Moreover, the following two diagrams may be useful when reading these proofs.

Proposition A.7 If $\tau P \subseteq \tau Q$ and $\psi Q \subseteq \psi P$ then $\tau Q = \tau P$.

Proof. Let $t \in \tau Q$. Since $Q \in \text{IG}$ and $\text{out} Q \neq \emptyset$, by CSP3, there is $w \in \text{out} Q^*$ such that $t \circ w \in \psi Q$. Hence $t \circ w \in \psi P \subseteq \tau P$ which means, by CSP1, that $t \in \tau P$. \end{proof}
Proposition A.8 If \( Q \subseteq P \) then \( \delta(Q \circ \text{Rcv}) = \delta(P \circ \text{Rcv}) = \emptyset, \tau(Q \circ \text{Rcv}) = \tau(P \circ \text{Rcv}) \) and 
\[ \phi(Q \circ \text{Rcv}) \subseteq \phi(P \circ \text{Rcv}). \]

**Proof.** By theorem A.5, \( U, W \in \text{GIO} \) and \( \delta U = \delta W = \emptyset \). The latter and \( \tau Q = \tau P \) implies that \( \tau U = \tau W \).

Suppose \((t, R) \in \phi U \). Then, by \( \delta U = \emptyset \) and proposition A.2, there exists \((w, R \cup \text{out } Q) \in \phi(Q \| \text{Rcv}) \) such that \( t = w[\text{chan } U] \). Since \( Q, \text{Rcv} \in \text{GIO} \), we have \( w[\text{chan } Q] \in \psi Q \) and \( (w[\text{chan } \text{Rcv}, R) \in \phi \text{Rcv} \).

By \( \psi Q \subseteq \psi P \), \( (w, R \cup \text{out } P) \in \phi(P \| \text{Rcv}) \) yielding \((t, R) \in \phi W \). \( \square \)

Proposition A.9 If \( Q \subseteq P \) and \( P \in \text{GIO}' \) then \( Q \circ \text{Rcv} = P \circ \text{Rcv} \).

**Proof.** By proposition A.8, it suffices to show that \( \phi W \subseteq \phi U \).

Suppose \((t, R) \in \phi W \). By propositions A.2 and A.8, there is \((w, R \cup \text{out } P) \in \phi(P \| \text{Rcv}) \) such that \( t = w[\text{chan } W] \). Since \( P, \text{Rcv} \in \text{GIO} \), we have \( w[\text{chan } P] \in \psi P \) and \( (w[\text{chan } \text{Rcv}, R) \in \phi \text{Rcv} \). From \( P \in \text{GIO}' \) it follows that \( w[\text{chan } P] \circ \langle a \rangle \not\in \tau P \), for all \( a \in \text{out } P \). Thus, by \( \tau Q = \tau P \) and CSP3, \( w[\text{chan } Q] \in \psi Q \). Hence \((t, R) \in \phi U \). \( \square \)

### A.5 Compositionality in acyclic networks and realisability

**Proposition A.10** Let \( \text{ep}_1 \) and \( \text{ep}_2 \) be extraction patterns as in the definition of \( \text{ep}_1 \oplus \text{ep}_2 \).

1. \( \text{ep} = \text{ep}_1 \oplus \text{ep}_2 \) is an extraction pattern such that \( \text{Dom} = \text{Dom}_1 \| \text{Dom}_2 \).
2. For every \( t \in \text{Dom} \), \( \text{extr}(t)[B'_1 = \text{extr}_1(t[B_1]) \) and \( \text{extr}(t)[B'_2 = \text{extr}_2(t[B_2]). \)
3. For every \( t \in \text{Dom} \), \( R_1 \subseteq \alpha B_1 \) and \( R_2 \subseteq \alpha B_2 \), 
\[ R_1 \cup R_2 \not\in \text{ref}(t) \iff R_1 \not\in \text{ref}(t[B_1]) \land R_2 \not\in \text{ref}(t[B_1]). \]

**Proof.** Follows directly from the definition of \( \oplus \). \( \square \)

The following diagram may be useful when reading the next proof.

---

**Proof of theorem 4.1.** By proposition A.4(3), WI1 holds for \( O \). Suppose that \((t, R) \in \phi O \) and \( t[C]F \in \text{Dom}_j. \) From proposition A.4(2) we have that \( t \not\in \delta O \). Hence, by proposition A.2, there is \((w, R \cup \alpha D) \in \phi S \) such that \( w \not\in \delta S \) and \( t = w[O] \). Together with proposition A.4(1) and WI1 for \( M \) and \( N \), this implies that there are \( R_0 \) and \( R_1 \) such that

1. \( R_0 \cup R_1 = R \cup \alpha D \)
2. \( (w[M, R_0] \in \phi M, (w[N, R_1] \in \phi N \) and \( w \in \text{Dom}_{\text{ref},j}. \)

By (2) and WI2 for \( M \) and \( N \), \( \alpha C - R_0 \not\in \text{ref}_j(w[C] = \text{ref}_j(t[C]) \) and \( \alpha D F - R_1 \not\in \text{ref}_j(w[D F] = \text{ref}_j(t[D F]) \). Hence, by (1) and proposition A.10(3), \( \alpha C F - R \not\in \text{ref}_j(t[C] F) \) which means that WI2 holds. Moreover, from (1,2) and WI2 for \( N \) it follows that
(3) \(\alpha D \cap R_0 \not\in \ref_d(w[D]).\)

To show W13 we additionally assume \(t[C \in \dom_{\mathcal{E}} \text{ and } \alpha \mathcal{E} \cap R \not\in \ref_g(t[\mathcal{E}]).\) This, (1,2,3) and W13 for \(M\), implies that \(w[D] \in \dom_{a\mathcal{E}}\) and \(\text{extr}_{\mathcal{E}d}(w[M]) \in \psi K\). Furthermore, from W13 for \(N\), we obtain \(w[y] \in \dom_{\mathcal{E}}\) and \(\text{extr}_{\mathcal{E}d}(w[N]) \in \psi L\). Hence \(t[\mathcal{E}] = w[y] \in \dom_{\mathcal{E}}\) and \(\text{extr}_{\mathcal{E}d}(w), \alpha D\mathcal{E}\) \(\in \phi I\). Thus, by proposition A.10(2), \(\text{extr}_{\mathcal{E}d}(w) \in \psi J\).

To prove W14 we take \(t \in \tau J\). By proposition A.2 and theorem A.5, there is \(w \in \tau I\) such that \(w \not\in \delta I\) and \(t = w[J]\). Thus, since W14 holds for both \(K\) and \(L\), \(\text{inv}_{\mathcal{E}c}(w[K]) \in \tau M\) and \(\text{inv}_{\mathcal{E}d}(w[L]) \in \tau N\). Let \(x = \text{inv}_{\mathcal{E}c}(w).\) Clearly, \(x[\mathcal{M} = \text{inv}_{\mathcal{E}c}(w[K]) \in \tau M\) and \(x[N = \text{inv}_{\mathcal{E}d}(w[L]) \in \tau N\). Hence \(x \in \tau S\) and \(\text{inv}_{\mathcal{E}c}(w) = x[O \in \tau O]\).

\[\square\]

Corollary A.11 Let \(K\) and \(L\) be two DIO processes as in figure 1, and let \(c, d, e, f\) and \(g\) be extraction patterns whose targets are respectively the channel sets \(C, D, E, F\) and \(G\). If \(M \in \text{swl}(K, e, d \oplus e)\) and \(N \in \text{swl}(L, d \oplus f, g)\) then \(M \otimes N \in \text{swl}(K \otimes L, c \oplus f, e \oplus g)\).

\[\textbf{Proof.}\] In the proof of theorem 4.1, W14 was not needed to show that W11-W13 hold. \[\square\]

Proposition A.12 Let \(P, Q\) and \(\text{Rcv}\) be processes as in figure 5. Moreover, let \(P, \text{Rcv} \in \text{DIO}\) and \(Q \in \text{swl}(P, \text{id}_{in}P, \text{id}_{out}P)\). Then \(Q \otimes \text{Rcv} = P \otimes \text{Rcv}\).

\[\textbf{Proof.}\] Let \(U = Q \otimes \text{Rcv}\) and \(W = P \otimes \text{Rcv}\). From corollary A.11 and theorems 2.1 and 2.6, it follows that \(W \in \text{DIO}\) and \(W \in \text{swl}(W, \text{id}_{in}W, \text{id}_{out}W)\). Hence \(\delta U = \delta W = \emptyset\) and it suffices to show that \(\phi U = \phi W\).

Let \((t, R) \in \phi U\). By W11 and W12 for \(U\), we have \(U \in \text{GIO}\). Hence there is \(w, R \cup \text{out}(Q) \in \phi(Q)[\text{Rcv}]\) such that \(t = w[\text{chan} U, w[\text{chan} Q] \in \psi Q\) and \(w[\text{chan} \text{Rcv}, R] \in \phi \text{Rcv}\). Thus, by W13 for \(Q\), \(w[\text{chan} P] \in \psi P\). Hence \((w, R \cup \text{out} P) \in \phi(P)[\text{Rcv}]\) which implies \((t, R) \in \phi W\). Hence, by \(\tau U = \tau W, (t, R \cup \{a\}) \in \phi U,\) a contradiction. Thus \(\phi U \subseteq \phi W\).

We now observe that \(\tau U = \tau W\). Otherwise we would have \(t \in \tau U \cap \tau W\) and \(t \circ \{a\} \in \tau W - \tau U\), for some \(t\) and \(a\). This, CSP3 for \(U\) and DIO2 for \(W\), implies that \((t, \{a\}) \in \phi U\) and \((t, \{a\}) \not\in \phi W\), contradicting \(\phi U \subseteq \phi W\).

Suppose \(\phi W \not\subseteq \phi U\). Then, by \(\tau U = \tau W\), we have \((t, R) \in \phi U \cap \phi W\) and \((t, R \cup \{a\}) \in \phi W - \phi U\), for some \(t, R\) and \(a\). From DIO2 for \(W\) it follows that \(t \circ \{a\} \not\in \tau W\). Hence, by \(\tau U = \tau W, (t, R \cup \{a\}) \in \phi U,\) a contradiction. Thus \(\phi W \subseteq \phi U\). \[\square\]

\[\textbf{Proof of theorem 4.2.}\] From W11-W14 and the definition of \(\text{id}\) it follows that \(Q \in \text{GIO}, (t, R) \in \phi Q \Rightarrow \text{ain}(Q) \cap R = \emptyset, \psi Q \subseteq \psi P\) and \(\tau P \subseteq \tau Q\). The first two properties imply \(Q \in \text{GIO}\). Hence, by the last two properties and proposition A.7, we obtain \(Q \preceq P\). \[\square\]

A.6 Compositionality in cyclic networks

The following diagram may be useful when reading the next two proofs.

![Diagam](image-url)

**Proposition A.13** If \(K\) and \(L\) are compatible GIO processes as in figure 7(a), then \(K \otimes L \in \text{GIO}\).
Proof. From $K$ and $L$ being compatible and deadlock-free (theorem A.5), it follows that $J$ is divergence-free and input-guarded. This, $K, L \in \mathcal{G}\mathcal{O}$ and the definition of the parallel composition and hiding operators, imply that $R \cap \alpha CF = \emptyset$, for every $(t, R) \in \phi J$.

The following diagram may be useful when reading the next proof.

Proof of theorem 5.1. Let $V = \{t \in \tau S \mid t[CF] \in \text{Dom}_f\}$ and $V' = \{t \in \tau O \mid t[CF] \in \text{Dom}_f\}$. Suppose that $V \cap \delta S \neq \emptyset$. Then, without loss of generality (note that, unlike in figure 1, the roles of $K$ and $L$ are now fully symmetric), there is $t \in V \cap \delta S$ such that $t[M] \in \delta M$ and $t[N] \in \tau N$. From $\text{SI}(a)$ for $M$ and $N$, and by induction on the length of the prefixes of $t$, it can be shown that $W \in \text{Dom}_{defgh}$, for all $w \leq t$. In particular, $t \in \text{Dom}_{defgh}$ which means that $t[CH] \in \text{Dom}_a$. Thus $t[M] \in \delta M$ and $t[CH] \in \text{Dom}_{eh}$. From CSP4 and $\mathcal{E} \mathcal{D} \neq \emptyset$ it follows that there are $a_1, a_2, \ldots \in \alpha \mathcal{E} \mathcal{D}$ such that $t_k = (t[M] \circ \langle a_1, a_2, \ldots \rangle) \in \tau M$, for all $k \geq 1$. Then, by taking $T = \{t_k \mid k \geq 1\}$, we obtain a contradiction with $\text{SI}(b)$ for $M$. Thus we have

1. $V \cap \delta S = \emptyset$.

We now observe that from (1) and $\text{SI}(a)$ for $M$ and $N$, and by induction on the length of the traces, it can be shown that

2. $V \subseteq \text{Dom}_{defgh}$.

Suppose $V' \cap \delta O \neq \emptyset$. Then, by (1), there is an infinite sequence of traces $t_1, t_2, \ldots \in \tau S \cap V$ such that $t_1 < t_2 < \ldots$ and $t_i[O] = t_i[O]$, for all $i \geq 1$. Hence $\{t_i[M] \mid i \geq 1\}$ is infinite. Consequently, by (2) and $\text{SI}(b)$ for $M$,

3. $\{\text{extr}_e(t_i[CH]) \mid i \geq 1\}$ is infinite.

For every $i \geq 1$, consider $w_i = \text{extr}_{defgh}(t_i)$ which is, by (2), a well-defined trace. We have the following: (i) $w_1, w_2, \ldots \in \tau I$ (follows from proposition A.10(2) and $\text{SI}(c)$ for $M$ and $N$); (ii) $\{w_i \mid i \geq 1\}$ is infinite (follows from (3)); and (iii) $w_i[\mathcal{F} = w_i[\mathcal{F}]$, for all $i \geq 1$ (follows from proposition A.10(2) and $t_i[O] = t_i[O]$, for all $i \geq 1$). But (i,ii,iii) contradict the assumed compatibility of $K$ and $L$. Hence $V' \cap \delta O = \emptyset$. This and the definition of the hiding operator means that


We now proceed with the case where $\text{SI}(b)$, we take an infinite $T \subseteq V'$. From (4) it follows that there is $W \subseteq V$ such that $T = W[O]$. By (1), $X = W[M] \subseteq \tau M$ and $Y = W[N] \subseteq \tau N$. From $T$ being infinite it follows that $X$ or $Y$ also is. Without loss of generality, we assume $X$ is infinite. Then, by $\text{SI}(b)$ for $M$, $\text{extr}_{ch}(X[CH])$ is infinite. If $\text{extr}(X[\mathcal{C}])$ is infinite then $\text{SI}(b)$ holds for $O$. Otherwise, $\text{extr}_{ch}(X[\mathcal{H}])$ is infinite and so is $Y[\mathcal{H}] = X[\mathcal{H}]$. Therefore, by $\text{SI}(b)$ for $N$, $\text{extr}_{df}(Y[\mathcal{D}])$ is infinite. If $\text{extr}_{df}(Y[\mathcal{D}])$ is infinite then $\text{SI}(b)$ holds for $O$. Otherwise, $\text{extr}_{df}(Y[\mathcal{D}])$ is infinite.

We are still left with the case when $\text{extr}(W[\mathcal{H}D])$ is infinite, while $\text{extr}_{df}(W[CF])$ is finite. Define $Z = \text{extr}_{defgh}(W)$. By (2) and $\text{SI}(c)$ for $M$ and $N$, $Z$ is well-defined and $Z \subseteq \tau I$. But we have that
Z[H \ D] is infinite and Z[C \ F] is finite, contradicting the compatibility of K and L. Hence S11(b) holds for O.

As for the remaining conditions, S11(c) and S12 for O follow directly from (1, 2, 4), S11(c) and S12 for M and N, and proposition A.10(2). Furthermore, from (1, 2, 4), S12 and S13(a) for M and N, and proposition A.10(2), it follows that S13(a) holds for O.

To show S13(b), suppose \( (t, R) \in \phi O, t \in V', \alpha \epsilon G \cap R \not\in \text{ref}_{eg}(t[E G]) \) and \( t \in \text{dom}_{eg} \). Then, by (1, 2, 4), we can find \( w \) and \( R_0, R_1 \) such that \( (w, R \cup \alpha \mathcal{H} D) \in \phi S, t = w|O, (w|M, R_0) \in \phi M, (w|\Lambda, R_1) \in \phi N \) and \( R \cup \alpha \mathcal{H} D = R_0 \cup R_1 \). Thus, by S13(a) for M and N, it follows that \( w \in \text{dom}_{eg} \). We then proceed, similarly as in the proof of WI3 in theorem 4.1, to show that \( \text{extr}_{eg}(t) \in \psi J \). Hence S13(b) holds.

Finally, S14 can be proved similarly as WI4 in theorem 4.1.

\[ \square \]

**Corollary A.14** Let K and L be two compatible \( \mathbb{G} \mathbb{O} \) processes as in figure 7(a), and let \( c, d, e, f, g \) and \( h \) be extraction patterns whose targets are respectively the channel sets \( C, D, E, F, G \) and \( H \). If \( M \in \text{sS1}(K, c \oplus h, d \oplus e) \) and \( N \in \text{sS1}(L, d \oplus f, g \oplus h) \) then \( M \oplus N \in \text{sS1}(K \ominus L, c \oplus f, e \oplus g) \).

**Proof.** In the proof of theorem 5.1, S14 was not needed to show that S11-S13 hold.

\[ \square \]

### A.7 Combining weak and strong implementability

**Proposition A.15** Let \( \text{ep} \) be an extraction pattern such that \( \text{dom} \) is prefix-closed and \( \text{extr}^{-1}(w) \) is always finite. Then \( \text{WI}(P, \text{ep}, \text{ep}') = S1(P, \text{ep}, \text{ep}') \).

**Proof.** We need to show that WI implies S1. The first assumption ensures that \( \mathbb{G} \mathbb{I} \) in WI1 and WI3 imply S13; the second one ensures that \( \mathbb{G} \mathbb{I} 2 \) in WI1 implies S11(b). To show S11(c) we observe that WI1 implies that for every \( t \in \text{Dom} \) there is \( w \in \psi Q \) such that \( t \leq w \) and \( w \in \text{inQ} = t|\text{inQ} \in \text{Dom} \). This, \( \text{dom} = \text{Dom} \) and WI3, implies \( \text{extr}_{\text{ep}, \text{ep}'}(w) \in \tau P \). Hence S11(c) holds by the monotonicity of extraction mappings.

The following two diagrams may prove useful when reading the next proof.

\[ \text{Proof of theorem 5.2.} \] By theorem 4.1, \( O \in \text{WI}(J, c, e \oplus g) \). This means S14 and S11(a) hold. Define \( V = \{ t \in \tau S \mid t[C \in \text{Dom}_{c} \} \) and \( V' = \{ t \in \tau O \mid t[C \in \text{Dom}_{c} \} \). From proposition A.4(2) and \( O \in \mathbb{G}(\text{Dom}_{c},\text{Dom}_{f}) \) it follows that \( V \cap \delta S = V' \cap \delta O = \emptyset \). Hence

1. \( V' = V[O, V[M \subseteq \tau M \text{ and } V[N \subseteq \tau N] \).

Suppose \( T \subseteq V' \) is infinite and \( \text{extr}(T[C]) \) is finite. By (1), there is \( W \subseteq V \) such that \( T = W[O \). From S11(a) for \( M, (1) \) and \( O \in \mathbb{G}(\text{Dom}_{c},\text{Dom}_{f}) \) and proposition A.10(2), it follows that \( W \subseteq \text{Dom}_{c} \). By S11(b) for \( M, W[M \) is finite. Hence \( W[N \) is infinite since \( T \) is. As a result, \( W[D \) is finite whereas \( W[\Lambda \) is infinite, contradicting \( O \in \mathbb{G}(\text{Dom}_{c},\text{Dom}_{f}) \). Hence S11(b) for \( O \) holds.

To show S11(c), suppose that \( t \in \tau O \) and \( t[C \in \text{Dom}_{c} \). From \( O \in \mathbb{G}(\text{Dom}_{c},\text{Dom}_{f}) \) it follows that there is \( w \in \tau O \) such that \( t \leq w, t[M = w[M \text{ and } w \in \psi O \). We will show that \( \text{extr}_{\text{eg}}(w) \in \tau J \) which will be sufficient due to the monotonicity of extraction functions.

From (1) it follows that there is \( z \in \tau S \) and \( R_0, R_1 \) such that
(2) \( R_0 \cup R_1 = \alpha D \cup \alpha D \varepsilon \mathcal{G} \)

(3) \( (z[M, R_0] \in \phi M, (z[N, R_1] \in \phi N \text{ and } w = z \mathcal{O}. \)

From WI2 for \( N \), (2,3) and SI3(a) for \( M \), it follows that \( z[D] \in \text{dom}_{d \varepsilon} \). This, (2,3) and WI3 for \( N \) implies that \( \text{extr}_{d \varepsilon} (z[N]) \in \tau I \). Together with SI1(c) for \( M \) this yields \( \text{extr}_{d \varepsilon} (z) \in \tau I \). Hence \( \text{extr}_{d \varepsilon} (w) \in \tau J \), by proposition A.10(2).

To show SI3(a) we take \( (w, R) \in \phi O \) satisfying \( w[C] \in \text{Dom} \) and \( \alpha \mathcal{E} \mathcal{G} \cap R \notin \text{ref}_{d \varepsilon}(w[\mathcal{G}]) \). By proceeding similarly as above we can find \( z, R_0, R_1 \) satisfying (3) and \( R_0 \cup R_1 = \alpha D \cup R \). We then have, as before, \( z[D] \in \text{dom}_{d \varepsilon} \). This, and WI3 for \( N \) implies \( w[\mathcal{G}] = z[\mathcal{G}] \in \text{dom}_{d \varepsilon} \).

SI3(b) for \( O \) follows from SI3(b) for \( M \) and WI3 for \( N \), the proof is similar to that for SI3(a); SI4 follows from SI4 for \( M \) and WI4 for \( N \).

**Corollary A.16** Let \( K \) and \( L \) be two GIO processes as in figure 7(b), and let \( c, d, e \) and \( g \) be extraction patterns whose targets are respectively the channel sets \( C, D, E \) and \( G \). If \( M \in \text{sSI}(K, c, d \oplus e) \) and \( N \in \text{sWI}(L, d, g) \) then \( M \sqcap N \in \text{SI}(K \sqcap L, c, e \oplus g) \).

**Proof.** In the proof of theorem 5.2, WI4 and SI4 were not needed to show SI1-SI3 that hold. \qed