Asynchronous Communication in Dynamically Structured Systems

R.P. Hopkins and M. Koutny

Abstract

We propose a process formalism for systems of asynchronously communicating processes with dynamic connectivity. We give an equivalence for such processes, validated against a CCS-like observational equivalence for synchronous communication and fixed connectivity.


Series Editor: M.J. Elphick

©1989 University of Newcastle upon Tyne.
Printed and published by the University of Newcastle upon Tyne, Computing Laboratory, Claremont Tower, Claremont Road, Newcastle upon Tyne, NE1 7RU, England.
Bibliographical details

HOPKINS, Richard Pinder

Asynchronous communication in dynamically structured systems. [By] R.P. Hopkins and M. Koutny

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Laboratory, 1989.

(University of Newcastle upon Tyne, Computing Laboratory, Technical Report Series, no. 287)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE.
Computing Laboratory. Technical Report Series. 287

KOUTNY, Maciej

Abstract

We propose a process formalism for systems of asynchronously communicating processes, validated against a CCS-like observational equivalence for synchronous communication and fixed connectivity.

About the author

Dr. R.P. Hopkins has been a lecturer in the Computing Laboratory since October, 1983, prior to which he was a Research Associate.

Dr. M. Koutny has been at the Computing Laboratory as a Research Associate from March, 1985 until Nov. 1986 and as a Lecturer from December, 1986. Prior to this he was at Warsaw Technical University in Poland from 1982.

Suggested keywords

COMMUNICATION
CONCURRENCY
DYNAMICALLY STRUCTURED SYSTEMS
MESSAGE PASSING
OBSERVATIONAL EQUIVALENCE

Suggested classmarks (primary classmark underlined)
Dewey (18th): 001.64404
U.D.C. 519.687
ASYNC HRONOUS COMMUNICATION
IN
DYNAMICALLY STRUCTURED SYSTEMS†

Richard P. Hopkins and Maciej Koutny

Computing Laboratory
The University
Newcastle upon Tyne NE1 7RU, U.K.

We propose a process formalism for systems of asynchronously communicating processes with dynamic connectivity. We give an equivalence for such processes, validated against a CCS-like observational equivalence for synchronous communication and fixed connectivity.

1. INTRODUCTION
This paper addresses two important characteristics of many concurrent, particularly distributed, systems which are not dealt with directly by formalisms such as CCS [4,8] or CSP [2,5], namely: asynchronous ("buffered") communication; and dynamic structure of communication channels, as for example occurs in concurrent object-based systems (e.g. [1]) where a reference from one object to another is a communication channel on which can be sent a reference to a third object. In [6,7] we develop a process model and algebra (DASS - Dynamic (A)Synchronously communicating Systems) which accommodates dynamic communication structure and deals explicitly with both synchronous and asynchronous communication. This paper presents part of that work, namely the process representation, asynchronous communication and process equivalence.

Figure 1 - System Structure and Communication

Figure 1 illustrates the process model. The (i)-(v) are successive stages in the execution of a synchronously communicating system comprising three processes, A, B and C. A box such as A₀ represents a process state with which are associated a set of ports, \{a!\} for A₀ and \{a?, b!\} for B₀. Each port is an outport (!) or an inport (?). An outport may be linked to an inport, as shown by the arcs, such that a message sent on an outport is receivable on the linked inport, and each link is unsharable, i.e. from a single outport to a single inport. The a in a? and b in b! are referred to as their tags, and ports with the same tag are linked. In (i) → (ii) there is a send by A on port a! and a matching receive by B on the linked port a? (these being the transfer ports of the respective communications). In such a transmission the link used disappears and may be replaced by some new links. For each new link there is a new port in the sender (a-1! or a-2?) with a tag obtained by sufffixing an integer to the tag of the send's transfer port; and a new port in the receiver (a-1? or a-2!), with tag obtained by the same sufffixing of the tag of the receive's transfer port.

In (vi) we show an intermediate stage which would occur between (i) and (ii) in the case of asynchronous communication. The circle represents a “buffered message”, labelled a! to identify its source port a!. We consider a message to have ports linked with the destination port on which it is receivable (bold link), and
with the new ports introduced by the send. If in (vi) A were to send a second
message on a-1!, its destination port would be a port of message a! and it would
only be receivable after the receipt of a!. (The a, a-1 and then a-1-1 etc. act as a
multi-message channel. Taking single links rather than channels as the basis has
the benefit of inherent message uniqueness and ordering.)

The remainder of this example illustrates full dynamic structure. In (ii) → (iii)
there is a transmission from B to C on b!!b? which is replaced by two new links
b-1!!b-1? and b-2!!b-2?. In (iii) → (iv) process B changes the inter-connection struc-
ture by introducing a join, an inport/outport pair (b-2?, a-2!) represented as a
dotted arc. This has the effect that the port (b-2! of C) which was linked with the
inport of the join becomes instead linked with the port (a-2? of A) which was
linked with the outport of the join, thus giving direct communication between A
and C. In (vii) and (viii), we show two intermediate stages which for asynchronous
communication could occur instead of (iii). In (ii) to (viii) to (iv), we see B sending
a message to C with a "reference" to a third process which C uses in (iv)→(ix)→(v).
(The join primitive is more general than required for the main objective of
allowing such communication of references, but this is in fact a simplifying
generalisation. We accumulate in a process state all the joins introduced by that
process, since this simplifies the formal treatment.)

2. PROCESS REPRESENTATION

The behaviour of a process is represented as a labelled transition tree (similar to a
CCS synchronisation tree) where the nodes are states and each transition is
either: a communication labelled with a characterisation p of a message sent or
received; or a silent transition, all such having the same label τ. A state has a set
of ports, a portset, and a set of accumulated joins (port pairs), a joinset.

The structure of ports is defined using a tag-graph, an acyclic directed graph with
vertices being tags, and arcs being labelled with positive integers, such that: each
vertex λ has exactly one incoming arc; and one outgoing arc for each possible label
n, this arc leading to a tag denoted by λ-n.

A port a∈Ports is a pair (λ, p) such that λ is a tag and p = ? or p = !. We denote λ by
[a], and p by a°.

A set of ports A is a portset if for all a,b∈A, [a]=[b]n₁…nk (k≥0) implies a = b.
A set of port pairs $H$ is a joinset if for all $(a,b) \in H$, $a^\circ = ? \neq b^\circ$; and for all pairs $(c,d),(e,f) \in H$, if $(c,d) = (e,f)$ then $c = e$ and $d = f$. We also define $|H| = \{a,b : (a,b) \in H\}$.

A communication $\mu \in \text{Comm}$ is a triple $(a,F,B)$, denoted $a_{FB}$, such that $a$ is a port and $F$ and $B$ are non-negative integers. We call $a$ the transfer port of $\mu$, denoted $Tsf_\mu$; if $a^\circ = ?$ then $\mu$ is a receive communication, $\mu \in \text{Rcv}$; otherwise $\mu$ is a send communication, $\mu \in \text{Snd}$. The $F$ is the number of new links created in the “forward direction” from sender to receiver. Similarly $B$ is the number of new “backward” links. We define the ports acquired as a result of a send or receive $\mu = a_{FB}$ as $\text{New}_\mu = \{a_1, \ldots, a_F + B\}$, where $[a_m] = [a] \cdot m$, and $(a_m)^\circ = a^\circ$ iff $m \leq F$, for all $m$.

**Definition** A process is a quadruple $Q = (V, T, P, J)$, where:

1. $V$ is a non-empty set of states (distinct processes having disjoint states);
2. $T \subseteq V \times (\text{Comm} \cup \{?\}) \times V$ is the transition relation with $(p,x,q) \in T$ denoted $p \rightarrow x q$ or $p \rightarrow q$, and $\rightarrow^*$ being the reflexive transitive closure of $\rightarrow$;
3. $P : V \rightarrow 2^{\text{Ports}}$ is a mapping giving the ports of a state;
4. $J : V \rightarrow 2^{\text{Ports} \times \text{Ports}}$ gives the joins of a state;

such that

1. $(V, T)$ is a labelled tree with the root denoted by $\text{root}_Q$.
2. $P(\text{root}_Q)$ is a finite portset (these initial ports also denoted $P_Q$).
3. If $q \in V$ then $J(q)$ is a joinset satisfying $|J(q)| \subseteq P(q)$.
4. If $p \rightarrow q$ then $P(q) = P(p)$ and $J(p) \subseteq J(q)$.
5. If $p \rightarrow q$ then $Tsf_\mu \in P(p)$ and $P(q) = P(p) \cup \text{New}_\mu \setminus \{Tsf_\mu\}$ and $J(p) \subseteq J(q)$. □

Note that (5) prohibits communication on joined ports, e.g. $a.2!$ for $B_3$ of Figure 1.

Also, (2), (4) and (5) imply $P(q)$ is a portset, for all $q \in V$.

For every state $p$ we denote $\text{Hist}_p = \{\mu : (\exists t,u) t \rightarrow _\mu u \rightarrow p\}$, $\text{Snd}_v = \text{Snd} \cap \text{Hist}_p$ and $\text{Rcv}_v = \text{Rcv} \cap \text{Hist}_p$. A join-free process is one with $J(p) = \emptyset$ for all its states $p$.

### 3. Asynchronous Communication

Asynchronous communication can be captured using the notions of process expansion, to be defined here, and synchronous communication given by a straightforward adaption for join-free processes of CCS parallel composition. The expansion of any core process $Q$ is a join-free process $S$ which internally does all message buffering that can occur in $Q$'s asynchronous communication with any other process. This is illustrated in Figure 2 by constructing the system of
Figure 1 as the composition of the expansion, $S$, of $B$ with the composition, $AC$, of $A$ with $C$. We take Figure 1(iv) as giving the initial states shown in Figure 2(a), and show some transitions: (a) $\rightarrow$ (d) $\rightarrow$ (e) (corresponding to (iv) $\rightarrow$ (ix) $\rightarrow$ (v) of Figure 1); (a) $\rightarrow$ (b) $\rightarrow$ (f); and (a) $\rightarrow$ (c). The initial state of the expansion has the same ports as the initial state of the core. For each expansion state, e.g. $S_0$, there is an underlying state of the core process, $\theta(S_0)$ which is $B_3$, represented as a process-state box within that for the expansion state. Also within an expansion state box we show: any internally buffered messages, each labelled with the source port on which it was received by $S$ or sent by $B$, and having a link to its destination port, on which it can be received by $B$ or sent by $S$; and any potential message bufferings as dashed arcs from source to destination of potentially buffered messages.
There are three buffering situations. Input buffering occurs in (a) \( \rightarrow \) (b) where a receive by \( S \) on \( a\cdot1? \) allows a subsequent receive by \( B \) on its \( a\cdot1? \). The converse output buffering occurs in (a) \( \rightarrow \) (c). Thirdly, a join in \( B \), \( (b\cdot2?\cdot a\cdot2!) \), can give message forwarding. In (a) \( \rightarrow \) (d) a message is received by \( S \) on \( b\cdot2? \), and in (d) \( \rightarrow \) (e) is sent by \( S \) on \( a\cdot2! \), without involving the core process \( B \) at all. After this forwarding there are two further forwarding buffers with source/destination ports \( b\cdot2\cdot1?/a\cdot2\cdot1! \) and \( b\cdot2\cdot2?/a\cdot2\cdot2! \), matching Figure 1(v) where for example a message sent by \( C \) on \( b\cdot2\cdot1! \) can subsequently be received by \( A \) on \( a\cdot2\cdot1? \). It is worth noting that the introduction of a new join in the core process can re-direct an input buffered message so that it is forwardable, as in (b) \( \rightarrow \) (f) for new join \( (a\cdot1?, b\cdot1!) \).

The forwarding buffers for a process state \( v \), with joins \( J(v) \), are captured within the set \( J_v \) of all pairs \((a,b)\) of ports, \( a^2=?=b^3 \), such that \([a]=[c]\cdot n_1 \ldots n_k\) and \([b]=[d]\cdot n_1 \ldots n_k\) for some \((c,d)\in J(v)\) or \((d,c)\in J(v)\).

For example in (a) or (e), since \( J(B_3) = \{(b\cdot2?, a\cdot2!)\} \), \( J_B \) includes \( (b\cdot2?, a\cdot2!) \) which gives the forwarding buffer between ports of \( S_0 \); and \( (b\cdot2\cdot1?, a\cdot2\cdot1!) \) and \( (b\cdot2\cdot2?, a\cdot2\cdot2!) \) which give the forwarding buffers between the ports of \( S_4 \). Given that a particular receive on the input of such a buffer has occurred in reaching a state of \( S \), e.g. \( b\cdot2\cdot20 \) of buffer \( (b\cdot2?, a\cdot2!) \) for \( S_3 \), and the output of that buffer is a port of that state, then the corresponding forwarded send, \( a\cdot2\cdot20 \) is possible for that state. For an expansion process state which has had receives \( C \) and has \( v \) as its underlying core state, we define the set of possible sends by the expansion state, \( \text{Snd}_v(C) \), as the set of these possible forwarded sends plus those messages such as \( b\cdot1! \) of (c) which have been sent by the core,

\[
\text{Snd}_v(C) = \{h_{FB} : a_{FB} \in C \land (a,b) \in J_v\} \cup \text{Snd}_v.
\]

The conditions governing the transitions for a state of the expanded process \( S \) are:

1. Any of the above identified sends can occur if the state has the required output; Any receive can occur on any of its inports, e.g. \( a\cdot1?_{10} \) in (a) \( \rightarrow \) (b) or \( b\cdot2\cdot20 \) in (a) \( \rightarrow \) (d), thus ensuring that a send by a composed process, \( AC \), can always succeed; and in either case there is no change for the core process.

2. Any transition for the underlying core process state is possible, with a \( \tau \) transition for the expansion, e.g. (a) \( \rightarrow \) (c) or (b) \( \rightarrow \) (f), except that a receive by the core process is only possible if that receive has already occurred in the expansion process. The only possibility here is \( a\cdot1?_{10} \) for \( B_3 \) in (b).
DEFINITION An expansion of a process $Q$ is a join-free process $S$, denoted $Q^\alpha$, for which $P_S=P_Q$ (same initial ports) and there is $\theta : V_S \to V_Q$ such that if $v \in V_S$ and $q = \theta(v)$ then all transitions for $v$ are specified by

1. If $\mu \in Rcvu \cup Snd_q(Rcv_v)$ and $Tsf_{\mu} \in P(v)$ then for exactly one $t$, $v \rightarrow_{\mu} t$ and $\theta(t) = q$.
2. If $q \rightarrow r$ and $Rcv_r \subseteq Rcv_v$ then for exactly one $t$, $v \rightarrow_{r} t$ and $\theta(t) = r$.

It can be shown that $Q^\alpha$ exists and is unique up to isomorphism.

4. PROCESS EQUIVALENCES

We now define bisimulations ($E$ and $F$) for synchronous and asynchronous equivalence, with the intended interpretations that two synchronously equivalent processes appear the same to an observer interacting synchronously with one or the other, and analogously for asynchronous equivalence. For synchronous equivalence it suffices here to consider only join-free processes for which the definition is a direct analogue of CCS observational equivalence.

DEFINITION Two join-free processes $Q$ and $R$ are synchronously equivalent, denoted $Q \equiv R$, if $P_Q=P_R$ and there is an $E \subseteq V_Q \times V_R$ such that $(root_Q, root_R) \in E$ and if $(v, w) \in E$ then $Hist_v = Hist_w$ and

1. If $v \rightarrow t$ then for some $u$, $w \rightarrow^* u$ and $(t, u) \in E$.
2. If $w \rightarrow u$ then for some $t$, $v \rightarrow^* t$ and $(t, u) \in E$.

For asynchronous equivalence we give a fully general definition which is inevitably rather complex since it deals with message buffering and dynamic structure. However that definition is validated against the standard definition of synchronous equivalence and the above-justified and straightforward definition of expansion by the theorem that asynchronous equivalence of two processes is the same thing as synchronous equivalence of their expansions.

DEFINITION Two processes $Q$ and $R$ are asynchronously equivalent, $Q \equiv R$, if $P_Q=P_R$ and there is an $F \subseteq V_Q \times V_R$ such that $(root_Q, root_R) \in F$ and if $(v, t) \in F$ then

1. If $w = v$, or $v \rightarrow w$ and $\text{card}(Rcv_w \cup Rcv_t) = \text{card}(Tsf_{Rcv_w} \cup Rcv_t)$, then there is a $u$ such that
   a. $t \rightarrow^* u$ and $(w, u) \in F$.
   b. $Rcv_u \subseteq Rcv_w \cup Rcv_t$ and $Snd_u(Rcv_t) \subseteq Snd_u(Rcv_w)$.
2. If $(a, b) \in J_v$ and $a \notin Tsf_{Rcv_t}$ and $(a, b) \cap (P(v) \cup P(t)) = \emptyset$ then for all positive integers $F$ and $B$ there is a $u$ such that
ASYMPTOMATIC COMMUNICATION IN DYNAMICALLY STRUCTURED SYSTEMS

(a) \( t \rightarrow^* u \) and \((v, u) \in E\).
(b) \( Rcv_u \subseteq Rcv_v \cup Rcv_t \cup \{a_{FB}\} \) and \( b_{FB} \in Sn_d u(Rcv_v \cup \{a_{FB}\})\).

Furthermore, (1) and (2) have symmetric versions for \( t \).

**THEOREM** For all processes \( Q \) and \( R \), \( Q^x = R^x \) if and only if \( Q \sim R \).

One may show that \( \sim \) and \( \cong \) are equivalence relations. Furthermore, with definitions (excluded here for lack of space) of \( || \) for the join-free synchronous composition and \( ||| \) for a general asynchronous composition (having a quite complex definition), it can be shown that for all processes \( Q, R \) and \( S \) for which the relevant compositions are defined:

1. \( (Q||R)||S = Q||(R||S) \) and \( (Q|||R)|||S = Q||(R|||S) \) and \( Q^x|||R^x = (Q|||R)^x \).
2. \( R \sim S \Rightarrow Q||R = Q||S \) and \( R \cong S \Rightarrow Q|||R = Q|||S \).

5. CONCLUDING REMARKS

The DASS formalism, part of which was presented here, uses a process representation which is similar to that of CCS. The significant differences are inclusion of "joins" which re-direct communication channels; the structured port labels, which avoid dealing with an infinite number of potential communication ports as in the generalisation [3] of CCS for dynamic structure of synchronously communicating systems; and prohibition of the sharing of channels which has differing interpretations in CCS and CSP, and is complex for dynamic structure and asynchronous communication. For process composition and comparison there are additionally asynchronous composition and asynchronous (observational) equivalence, and extensions of synchronous composition and equivalence for processes having joins.

A dynamically structured asynchronously communicating system can be dealt with using synchronous composition and equivalence on the expansions of its component processes (or, as in CCS and CSP, by the composition of particular of those processes with explicit buffering processes). However the use of processes with joins and asynchronous composition more directly captures the essential structure and behaviour of such systems, abstracting out the details of message buffering and re-direction, and thus simplifying formal reasoning. Although the definition of asynchronous equivalence given is more complex than that of the synchronous equivalence, the actual proof of asynchronous equivalence between
two particular processes is likely to be more straightforward than proof of synchronous equivalence between their expansions (or between their respective compositions with buffering processes). Also dynamic structure of synchronously communicating systems cannot be dealt with without some additional mechanism such as joins.

REFERENCES


