A Majority Vote, Modelled by Asynchronous Readers and Asynchronous Writers

Antoon H. Boode\textsuperscript{a,b,1}, and Jan F. Broenink\textsuperscript{a}

\textsuperscript{a} Robotics and Mechatronics, Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, the Netherlands

\textsuperscript{b} InHolland University of Applied Science, the Netherlands

Abstract. Reading and writing is modelled in CSP using actions containing the symbols $?\text{ and }!$. These reading actions and writing actions are synchronous, and there is a one-to-one relationship between occurrences of pairs of these actions. In the CPA conference 2017, we introduced the extended half-synchronous alphabetised parallel operator $X \parallel Y$, which disconnects the writing to and reading from a channel in time; the reading processes are divided into sets which are set-wise asynchronous, but intra-set-wise synchronous, giving full flexibility to the reads.

In this paper, we allow multiple writers to write to the same channel set-wise asynchronously, but intra-set-wise synchronously and we study the impact on our (Extended) Vertex Removing Synchronised Product. The advantages we accomplish are that the extension of $X \parallel Y$ gives more flexibility by indexing the writing actions and the reading actions, leading to a straightforward majority vote design. Furthermore, the extension of $X \parallel Y$ preserves the advantages of the $X \parallel Y$ operator.

Keywords. CSP, Half-Synchronous Alphabetised Parallel Operator, Asynchronous and Synchronous Write Actions, Asynchronous and Synchronous Read Actions, (Extended) Vertex Removing Synchronised Product

Introduction

Periodic Hard Real-Time Control Systems (PHRCSs) modelled using process algebras comprise many short processes, which leads to fine-grained concurrency. Because of this fine-grained concurrency and the many short processes we introduced in [1] a software architecture which controls the behaviour of the PHRCSs by directed graphs\textsuperscript{2}.

To let the PHRCS perform its task as required by the specification, the processes synchronise over actions, asserting a certain order of the actions of the processes. A special case of this synchronisation over actions is the notion of writing to and reading from a channel, which was introduced in [4]. This writing to and reading from a channel is synchronous.

\textsuperscript{1}Corresponding Author: Ton Boode, Robotics and Mechatronics, CTIT Institute, Faculty EEMCS, University of Twente, P.O. Box 217 7500 AE Enschede, The Netherlands. Tel.: +31 631 006 734; E-mail: a.h.boode@utwente.nl.

\textsuperscript{2}Due to this architecture described in [2], we do not need models like failures models and traces models such as given by [3].
A.H. Boode, J.F. Broenink / A Majority Vote

In [5] and [6] we have introduced the (extended) half-synchronous alphabetised parallel operator, which disconnects the writing to and reading from a channel. The writing to a channel by more than one process was inhibited in [5], but relaxed to synchronous writing in [6]. In this paper, we relax these restrictions further; the writing processes are divided into sets which are set-wise asynchronous, but intra-set-wise synchronous, giving full flexibility to the asynchronous writing and reading actions. In this manner we are able to model a majority vote using this enhanced operator, together with the indexed writing and reading actions. Note that we still require our graphs to be pairwise consistent (defined in Definition 1) to ensure that pathological cases like a deadlock are avoided.

The idea is quite simple. Let the processes $P_1$, $P_2$ and $P_3$ contain a writing action $c_1 x : T$. Then by indexing this writing action for process $P_1$ with the indices 1, 2, for process $P_2$ with the indices 2, 3 and for process $P_3$ with the indices 1, 3, we have the following writing actions, e.g., for process $P_1$ the writing action $c_{1,2} x : T$, for process $P_2$ the writing action $c_{1,2} x : T$ and for process $P_3$ the writing action $c_{1,3} x : T$. In fact, $c_{1,2} x : T$ is a kind of short hand shown in Listing 1, where $P_1$ is strongly bisimilar to $P_2$.

\[
P_1 = c_{1,2} x : T \rightarrow \cdots \rightarrow \text{SKIP}
\]

\[
P_2 = c_{1,2} x : T \rightarrow \cdots \rightarrow \text{SKIP}
\]

Listing 1: Strongly bisimilar processes $P_1$ and $P_2$ with respect to indexed writing actions.

Let $X_i$ be the alphabet of process $P_i$, $i = 1, \ldots, 4$. Listing 2 shows the definition of this example, where the enhanced extended half-synchronous parallel operator is denoted as $\parallel$. The $wfnp$ action stands for the $wait\forall NextPeriod$ action [7], which is necessary to make sure that the processes in Listing 2 are pairwise consistent.

\[
P_1 = \cdots \rightarrow (c_{1,2} x : T \rightarrow wfnp \rightarrow \text{SKIP})
\]

\[
P_2 = \cdots \rightarrow (c_{1,3} x : T \rightarrow wfnp \rightarrow \text{SKIP})
\]

\[
P_3 = \cdots \rightarrow (c_{1,3} x : T \rightarrow wfnp \rightarrow \text{SKIP})
\]

\[
P_4 = c_{1,3} x : T \rightarrow \cdots \rightarrow wfnp \rightarrow \text{SKIP}
\]

\[
P_{1,2,3,4} = ((P_{X_1 \parallel X_2} P_2)_{X_1 \cup X_2 \parallel X_3} P_3)_{X_1 \cup X_2 \cup X_3 \parallel X_4} P_4
\]

Listing 2: The majority vote over a channel $c$.

Assume that the processes $P_1$ through $P_3$ read the following values into $x$ over channel $c_1$ through $c_3$: for $c_1$ $x$ is $v_1$, for $c_2$ $x$ is $v_2$, and for $c_3$ $x$ is $v_2$, respectively. Then, only the processes $P_2$ and $P_3$ can engage in the synchronisation over channel $c$, because they both belong to the group of processes with index 2.

In [1,8,9] we have developed a Vertex-Removing Synchronised Product (VRSP) that improves the performance of PHRCSs, where the PHRCS is designed using a
process algebra. With respect to asynchronous writing and reading we have adapted this VRSP into the Dot Vertex-Removing Synchronised Product (DVRSP) in [5] and the Extended Dot Vertex-Removing Synchronised Product (EVSP) [5]. In this paper we extend the EVSP into the Enhanced Extended Dot Vertex-Removing Synchronised Product (EEVSP), while maintaining full flexibility for the asynchronous writing and asynchronous reading.

We start in Section 1 with the terminology used on graphs. In Section 2 we describe the relational semantics of the enhanced extended half-synchronous parallel operator. In Section 3 we give the adaptation of the Extended Vertex Removing Synchronised Product (EVSP). We finish with the conclusions in Section 4.

1. Terminology

The terminology in this section is a copy of the terminology given in [2] relevant for this paper. We use [10], [11] and [12] for terminology and notations on graphs not defined here.

1.1. Graph Basics

Throughout this paper all graphs we consider are finite, deterministic, directed, acyclic, labelled multi-graphs, i.e., they may have multiple arcs. Such graphs consist of a vertex set $V$ (representing the states of a process), an arc set $A$ (representing the actions, i.e., transitions from one state to another), a set of labels $L$ (in our applications in fact a set of label pairs, each representing a type of action and the worst-case duration of its execution), and two mappings. The first mapping $\mu : A \rightarrow V \times V$ is an incidence function that identifies the tail and head of each arc $a \in A$. In particular, $\mu(a) = (u, v)$ means that the arc $a$ is directed from $u \in V$ to $v \in V$, where $\text{tail}(a) = u$ and $\text{head}(a) = v$. We also call $u$ and $v$ the ends of $a$. The second mapping $\lambda : A \rightarrow L$ assigns a label pair $\lambda(a) = (\ell(a), t(a))$ to each arc $a \in A$, where $\ell(a)$ is a string representing the (name of an) action and $t(a)$ is the weight of the arc $a$. This weight $t(a)$ is a real positive number representing the worst-case execution time of the action represented by $\ell(a)$.

Let $G$ denote a graph according to the above definition. An arc $a \in A(G)$ is called an in-arc of $v \in V(G)$ if $\text{head}(a) = v$, and an out-arc of $v$ if $\text{tail}(a) = v$. The in-degree of $v$, denoted by $d^-(v)$, is the number of in-arcs of $v$ in $G$; the out-degree of $v$, denoted by $d^+(v)$, is the number of out-arcs of $v$ in $G$. The subset of $V(G)$ consisting of vertices $v$ with $d^-(v) = 0$ is called the source of $G$, and is denoted by $S'(G)$. The subset of $V(G)$ consisting of vertices $v$ with $d^+(v) = 0$ is called the sink of $G$, and is denoted by $S''(G)$.

A graph $G$ is called weakly connected if all pairs of distinct vertices $u$ and $v$ of $G$ are connected through a sequence of distinct vertices $u = v_0v_1\ldots v_k = v$ and arcs $a_1a_2\ldots a_k$ of $G$ with $\mu(a_i) = (v_{i-1}, v_i)$ or $(v_i, v_{i-1})$ for $i = 1, 2, \ldots, k$. We are mainly interested in weakly connected graphs, or in the weakly connected components of a graph $G$. If $X \subseteq V(G)$, then the subgraph of $G$ induced by $X$, denoted as $G[X]$, is the graph on vertex set $X$ containing all the arcs of $G$ which have both their ends in $X$ (together with $L$, $\mu$ and $\lambda$ restricted to this subset of the arcs). If $X \subseteq V(G)$ induces a weakly connected subgraph of $G$, but there is no set $Y \subseteq V(G)$ such that $G[Y]$ is weakly connected and $X$ is a proper subset of $Y$, then $G[X]$ is called a weakly connected component of $G$. In the sequel, throughout we omit the words weakly connected, so a component should always be understood as a weakly connected component. In contrast to the notation in the textbook of [10], we use $\omega(G)$ to denote the number of components of a graph $G$. 
We denote the components of $G$ by $G_i$, where $i$ ranges from 1 to $\omega(G)$. In that case, we use $V_i$, $A_i$, and $L_i$ as shorthand notation for $V(G_i)$, $A(G_i)$ and $L(G_i)$, respectively. The mappings $\mu$ and $\lambda$ have natural counterparts restricted to the subsets $A_i \subset A(G)$ that we do not specify explicitly. We use $G = \sum_{i=1}^{\omega(G)} G_i$ to indicate that $G$ is the disjoint union of its components, implicitly defining its components as $G_1$ up to $G_{\omega(G)}$. In particular, $G = G_1$ if and only if $G$ is weakly connected itself.

A graph $G$ is called deterministic\(^3\) if its arcs have the following property. If $\lambda(a) = \lambda(b)$ for two arcs $a \in A$ and $b \in A$ with $\text{head}(a) \neq \text{head}(b)$, then $\text{tail}(a) \neq \text{tail}(b)$.

An arc $a$ with label pair $\lambda(a)$ in a graph $G$ is a synchronising arc with respect to another graph $H$, if and only if there exists an arc $b \in A(H)$ with label pair $\lambda(b)$ such that $\lambda(a) = \lambda(b)$.

We assume that two different arcs with the same head and tail have different labels; otherwise, we replace such multiple arcs by one arc with that label, because these arcs represent exactly the same action at the same stage of a process. Hence, we require that the following property holds for all the graphs we consider: any two distinct arcs $a \in A$ and $b \in A$ with $\mu(a) = \mu(b)$ have $\lambda(a) \neq \lambda(b)$.

For each pair $(v_i, v_j) \in V(G) \times V(G)$, we let $A(v_i, v_j) = \{a \in A(G) \mid \mu(a) = (v_i, v_j)\}$, and we let $t_m(v_i, v_j) = \max_{a \in A(v_i, v_j)} t(a)$.

A sequence of distinct vertices $v_0v_1 \ldots v_k$ and arcs $a_1a_2 \ldots a_k$ of $G$ is a (directed) path\(^4\) in $G$ if $\mu(a_i) = (v_{i-1}, v_i)$ for $i = 1, 2, \ldots, k$. We denote such a path by $P = v_0a_1v_1a_2 \ldots a_kv_k$, and we define its weight as $w(P) = \sum_{a_i \in A(P)} t(a_i)$.

A path from a vertex of the source of $G$ to a vertex of the sink of $G$ is called a full path (of $G$).

The path length of $G_i$, denoted by $\ell(G_i)$, is the maximum of $w(P)$ taken over all full paths $P$ of $G_i$.

The path length of a graph $G = \sum_{i=1}^{\omega(G)} G_i$, denoted by $\ell(G)$, is defined as $\ell(G) = \sum_{i=1}^{\omega(G)} \ell(G_i)$.

In the next subsection, we introduce a (directed labelled multigraph) analogue of the Cartesian product of two graphs and several other products we derive from it, resulting in the VRSP.

### 1.2. Graph Products

We start by introducing the next analogue of the Cartesian product.

The Cartesian product $G_i \square G_j$ of $G_i$ and $G_j$ is defined as the graph on vertex set $V_{i,j} = V_i \times V_j$, and arc set $A_{i,j}$ consisting of two types of labelled arcs. For each arc $a \in A_i$ with $\mu(a) = (v_i, w_j)$, an arc of type $i$ is introduced between tail $(v_i, v_j) \in V_{i,j}$ and head $(w_i, w_j) \in V_{i,j}$ whenever $v_j = w_j$; such an arc receives the label $\lambda(a)$. This implicitly defines parts of the mappings $\mu$ and $\lambda$ for $G_i \square G_j$. Similarly, for each arc $a \in A_j$ with $\mu(a) = (v_j, w_j)$, an arc of type $j$ is introduced between tail $(v_i, v_j) \in V_{i,j}$ and head $(w_i, w_j) \in V_{i,j}$ whenever $v_i = w_i$; such an arc receives the label $\lambda(a)$. This completes

\(^3\)This is equivalent to determinism in the set of processes which is represented by the graph $G$.

\(^4\)There is a close relationship between a trace and a directed path; ‘a trace is a sequence of visible actions in the order they are observed.’ [3, page 29], a trace $b_1b_2 \ldots b_n$ of a process $Q$ is represented by a path $P = v_0a_1v_1 \ldots v_{n-1}a_nv_n$ in $G$, $\ell(a_i) = b_i, i = 1, 2, \ldots, n$ where the process $Q$ is represented by the graph $G$. 

the definition of $A_{i,j}$ and the mappings $\mu$ and $\lambda$ for $G_i \square G_j$. So, arcs of type $i$ and $j$
correspond to arcs of $G_i$ and $G_j$, respectively, and have the associated labels. For $k \geq 3$,
the Cartesian product $G_1 \square G_2 \square \cdots \square G_k$ is defined recursively as $((G_1 \square G_2) \square \cdots \square G_k$.
This Cartesian product is commutative and associative, as can be verified easily and is a
well-known fact for the undirected analogue.

Since we are particularly interested in synchronising arcs, we modify the Cartesian
product $G_i \square G_j$ according to the existence of synchronising arcs, i.e., pairs of arcs with
the same label pair, with one arc in $G_i$ and one arc in $G_j$.

The first step in this modification consists of ignoring (in fact deleting) the syn-
chronising arcs while forming arcs in the product, but additionally combining pairs of
synchronising arcs of $G_i$ and $G_j$ into one arc, yielding the intermediate product which
we denote by $G_i \boxtimes G_j$.

To be more precise, $G_i \boxtimes G_j$ is obtained from $G_i \square G_j$ by first ignoring all except
for the so-called asynchronous arcs, i.e., only by maintaining all arcs $a \in A_{i,j}$ for which
$\mu(a) = ((v_i, v_j), (w_i, w_j))$, whenever $v_j = w_j$ and $\lambda(a) \notin L_j$, as well as all arcs $a \in A_{i,j}$
for which $\mu(a) = ((v_i, v_j), (w_i, w_j))$, whenever $v_i = w_i$ and $\lambda(a) \notin L_i$. Additionally, we
add arcs that replace synchronising pairs $a_i \in A_i$ and $a_j \in A_j$ with $\lambda(a_i) = \lambda(a_j)$.
If $\mu(a_i) = (v_i, w_i)$ and $\mu(a_j) = (v_j, w_j)$, such a pair is replaced by an arc $a_{i,j}$ with
$\mu(a_{i,j}) = ((v_i, v_j), (w_i, w_j))$ and $\lambda(a_{i,j}) = \lambda(a_i)$. We call such arcs of $G_i \boxtimes G_j$ synchronous
arcs.

The second step in this modification consists of removing (from $G_i \boxtimes G_j$) the vertices
$(v_i, v_j) \in V_{i,j}$ and the arcs $a$ with $\text{tail}(a) = (v_i, v_j)$, in the case that $(v_i, v_j)$ has level $> 0$
in $G_i \square G_j$ but level $0$ in $G_i \boxtimes G_j$. This is then repeated in the newly obtained graph,
and so on, until there are no more vertices at level $0$ in the current graph that are at
level $> 0$ in $G_i \square G_j$. This finds its motivation in the fact that in our applications, the
states that are represented by such vertices can never be reached, so are irrelevant.

The resulting graph is called the Vertex-Removing Synchronised Product (VRSP
for short) of $G_i$ and $G_j$, and denoted as $G_i \boxtimes G_j$. For $k \geq 3$, the VRSP $G_1 \boxtimes G_2 \boxtimes \cdots \boxtimes G_k$
is defined recursively as $((G_1 \boxtimes G_2) \boxtimes \cdots \boxtimes G_k$.
The VRSP is commutative, but not associative in general, in contrast to the Cartesian product. However, associativity of
the VRSP is guaranteed if we require the graphs on which we apply the VRSP to be
pairwise consistent.

Recall that our processes are acyclic, but are started again at every period of the
PHRCS. Therefore, whenever two processes $P_1$ and $P_2$ are consistent, this means that
in their parallel execution both processes will reach their set of final states. For the two
components $G_1$ and $G_2$ representing these two processes, this means that the sinks of
$G_1$ and $G_2$ must represent the final states of $P_1$ and $P_2$. But for $G_1 \boxtimes G_2$ this only makes
sense if the sink of $V(G_1 \boxtimes G_2)$ represents the final states of the process $P_{1,2}$ (where $P_{1,2}$
is strongly bisimilar to $P_1 || P_2$). We will introduce several contraction concepts in graphs
to describe and analyse consistency of the associated processes. This is explained and
formalised by the concepts of a weak contraction, a strong contraction and a pseudopath
in Section 1.3.

### 1.3. Graph Isomorphism and Graph Contraction

The isomorphism we introduce in this section is an analogue of a known concept for
unlabelled graphs, but involves statements on the labels.

Formally, an isomorphism from $G$ to $H$ consists of two bijections $\phi : V(G) \to V(H)$
and $\psi : A(G) \to A(H)$ such that for all $a \in A(G)$ $\mu(a) = (u, v)$ if and only if $\mu(\psi(a)) =
(\phi(u), \phi(v))$ and $\lambda(a) = \lambda(\psi(a))$. Since we assume that two different arcs with the same
head and tail have different labels, however, the bijection $\psi$ is superfluous. The reason is
that, if \((\phi, \psi)\) is an isomorphism, then \(\psi\) is completely determined by \(\phi\) and the labels. In fact, if \((\phi, \psi)\) is an isomorphism and \(\mu(a) = (u, v)\) for an arc \(a \in A(G)\), then \(\psi(a)\) is the unique arc \(b \in A(H)\) with \(\mu(b) = (\phi(u), \phi(v))\) and label \(\lambda(b) = \lambda(a)\). Thus, we may define an isomorphism from \(G\) to \(H\) as a bijection \(\phi : V(G) \rightarrow V(H)\) such that there exists an arc \(a \in A(G)\) with \(\mu(a) = (u, v)\) if and only if there exists an arc \(b \in A(H)\) with \(\mu(b) = (\phi(u), \phi(v))\) and \(\lambda(b) = \lambda(a)\).

We distinguish two types of contractions. The first type contracts vertices in \(G_1 \sqcup G_2\) related to asynchronous arcs of graphs \(G_1\) and \(G_2\) and is called a weak contraction. The second type contracts a set of vertices without taking into account whether the arcs belonging to these vertices are synchronous or asynchronous and is called a strong contraction.

Let \(a \in A(G)\) with \(\mu(a) = (u, v)\). By contracting \(a\) we mean replacing \(u\) and \(v\) by a new vertex \(\overline{uv}\), deleting all arcs \(b \in A(G)\) with \(\mu(b) = (u, v)\) or \(\mu(b) = (v, u)\), and for any \(x \neq u, v\) replacing each pair of arcs \(c \in A(G)\) and \(d \in A(G)\) with \(\mu(c) = (u, x)\), \(\mu(d) = (v, x)\) and \(\lambda(c) = \lambda(d)\) by one arc \(e\) with \(\mu(e) = (\overline{uv}, x)\) and \(\lambda(e) = \lambda(c)\), and, similarly replacing each pair of arcs \(c \in A(G)\) and \(d \in A(G)\) with \(\mu(c) = (x, u)\), \(\mu(d) = (x, v)\) and \(\lambda(c) = \lambda(d)\) by one arc \(e\) with \(\mu(e) = (x, \overline{uv})\) and \(\lambda(e) = \lambda(c)\).

To define the notion of weak contraction, let \(T\) be the set of asynchronous arcs in \(G_1 \sqcup G_2\) that correspond to arcs in \(G_1\). Then the weak contraction of \(G_1 \sqcup G_2\) with respect to \(G_1\), denoted by \(\rho_{G_1}(G_1 \sqcup G_2)\), is defined as the graph obtained from \(G_1 \sqcup G_2\) by successively contracting each arc \(a \in T\). Likewise, let \(T\) be the set of asynchronous arcs in \(G_1 \sqcup G_2\) that correspond to arcs in \(G_2\). Then the weak contraction of \(G_1 \sqcup G_2\) with respect to \(G_2\), denoted by \(\rho_{G_2}(G_1 \sqcup G_2)\), is defined as the graph obtained from \(G_1 \sqcup G_2\) by successively contracting each arc \(a \in T\). We also use \(G_1^\rho\) as shorthand for \(\rho_{G_2}(G_1 \sqcup G_2)\) and \(G_2^\rho\) as shorthand for \(\rho_{G_1}(G_1 \sqcup G_2)\).

Let \(H\) be a subgraph of \(G_1 \sqcup G_2\). Then in \(\rho_{G_2}(G_1 \sqcup G_2)\), \(H\) corresponds to a subgraph \(H'\) of \(G_1\). We denote this \(H'\) by \(\rho_{G_2}(H)\), and say that \(H\) is mapped to \(\rho_{G_2}(H)\) by \(\rho_{G_2}\). We use similar terminology and notation with respect to for \(\rho_{G_1}\).

We now turn to the definition of strong contraction. Let \(X\) be a nonempty proper subset of \(V(G)\), and let \(Y = V(G) \setminus X\). Then to obtain the strong contraction of \(G\) with respect to \(X\), we first replace \(X\) by a new vertex \(\tilde{x}\), deleting all arcs with both ends in \(X\), delete all arcs \(a \in A(G)\) with \(\mu(a) = (u, v)\) for \(u \in X\) and \(v \in Y\) by an arc \(c\) with \(\mu(c) = (\tilde{x}, v)\) and \(\lambda(c) = \lambda(a)\), and replace each arc \(b \in A(G)\) with \(\mu(b) = (u, v)\) for \(u \in Y\), and replace \(v \in X\) by an arc \(d\) with \(\mu(d) = (u, \tilde{x})\) and \(\lambda(d) = \lambda(b)\). If after this contraction there are arcs with the same ends and labels, then these arcs are replaced by one arc with the same ends and label. We denote the resulting graph as \(G/X\), and say that \(G/X\) is the strong contraction of \(G\) with respect to \(X\).

We use the strong contraction in particular to remove non-determinism, in the following way. Recall that non-determinism occurs in a graph \(G\) if there is a set of arcs \(B \in A(G)\) with the same tail and label, but different heads. In this case, let us denote such a set of different heads by \(Z\). In \(G/Z\), all the arcs of \(B\) (with heads in \(Z\)) are replaced by one arc with the same tail and label and a new head. So, this removes the non-determinism from \(G\) caused by the arc set \(B\). If there occurs non-determinism in the graph \(G/Z\), we iteratively repeat the above contraction procedure until the resulting graph is deterministic. We denote the resulting graph by \(G^\delta\).

Let \(H\) be a subgraph of \(G\). Then in \(G^\delta\), the graph that corresponds to \(H\) is denoted by \(H^\delta\). We say that \(H\) is mapped to \(H^\delta\) by \(\delta\).

The above two types of contractions play a key role in our notion of consistency of graphs. Before we define this notion, we first introduce one additional concept. This concept relates paths in \(G_1 \sqcup G_2\) to paths in \((G_1^\rho)^\delta\) and \((G_2^\rho)^\delta\), in the following way.

A full path \(P\) of \(G_1 \sqcup G_2\) is called a pseudopath if \(\rho_{G_2}(P)\) is isomorphic to a full path.
in $G_1$, and $\rho G_1(P)$ is isomorphic to a full path in $G_2$. Note that in this case $(\rho G_2(P))^\delta$ is a unique full path in $(G_1^n)^\delta$, and $(\rho G_1(P))^\delta$ is a unique full path in $(G_2^n)^\delta$, that satisfy this condition. In particular, $P$ is a full path in $(\rho G_2(P))_p \sqcap (\rho G_1(P))_p$. We often say there exists a full path in $G_1 (G_2)$ for a pseudopath in $G_1 \sqcap G_2$ if we mean that these paths exist in the above sense when $(G_1^n)^\delta \cong G_1$ (and $(G_2^n)^\delta \cong G_2$). Similarly, we often say there exists a pseudopath in $G_1 \sqcap G_2$ for every full path in $G_1 (G_2)$ if we mean that there exists a pseudopath $P$ in $Q \sqcap R$ for full paths $Q$ in $G_1$ and $R$ in $G_2$.

2. The Relational Semantics of the Enhanced Extended Half-Synchronous Parallel Operator

If processes write synchronously to a channel this synchronous writing is inhibited by the early versions of CSP\(^5\). Later on this was relaxed to multiple writers to the same channel [13]. In [5] we described asynchronous writing and reading in such a manner that the writers will deadlock if they are trying to invoke the same writing action. We lift these restrictions such that the writers are allowed to write synchronously as well as asynchronously, and the readers are allowed to read synchronously as well as asynchronously. Apart from the synchronous and asynchronous writing, this has been described in [6]. In this section, we complete the description of asynchronous writing and asynchronous reading by two extensions of the Half-Synchronous Operator:

- Indexing of the $\xi$-action, allowing set-wise asynchronous reading and intra-set-wise synchronous reading.
- Indexing of the $\eta$-action, allowing set-wise asynchronous writing and intra-set-wise synchronous writing.

The relational semantics of the enhanced extended half-synchronous alphabetised parallel operator $\langle X, Y \rangle$ is given in Figure 1. But first, we give an example of asynchronous writing and asynchronous reading describing a majority vote. Assume we are considering a safety-critical system controlling one actuator via three sensors. Each sensor is read from by a different process and the actuator is written to by a different process. The value read from the sensor is mapped into three ranges, say high, middle and low. The values are only invalid if one process maps its value into the high range, one process maps it value into the middle range, and the remaining process maps it value into the low range. In this case an error has to be raised, which is beyond the scope of this example. All other combinations lead to a valid value and the value which has a majority is sent to the actuator. To achieve consistent processes and thereby avoid a deadlock, we use a waitForNextPeriod-action (the $wfnp$ in the model) to align with the period of the PHRCS. This requirement is easily modelled by synchronising actions representing the high, middle and low ranges.

As an example, if in Listing 3 sensor $S_0$ has read the value 74, sensor $S_1$ has read the value 75, and sensor $S_2$ has read the value 76, the value middle will be chosen and sent to the actuator. To avoid that all processes engage at the start in a $wfnp$ action, a clock process $C$ with alphabet $Z$ has been added that enables the $wfnp$ action after the timer has expired. The $wfnp$ action is enabled just before the end of the period. The time-out value must be chosen in such a manner that after expiration there is only just enough time (in the order of a few $\mu$s) for a read and write communication to finish execution, e.g., if the communication and actions of the sensors and the actuator takes

\(^5\)Because we consider acyclic processes and therefore acyclic graphs only, our extension of CSP is in fact more a domain-specific CSP dialect.
\[ S_i = read.x_i \{ x_i < 75 \& c_i \{ (i, (i+1) \% 3) \} low \rightarrow wfnp \rightarrow \text{SKIP} \]

\[ \Box \]

\[ 75 \leq x_i < 125 \& c_i \{ (i, (i+1) \% 3) \} middle \rightarrow wfnp \rightarrow \text{SKIP} \]

\[ \Box \]

\[ 125 \leq x \& c_i \{ (i, (i+1) \% 3) \} high \rightarrow wfnp \rightarrow \text{SKIP} \]

\[ \Box \]

\[ wfnp \rightarrow \text{SKIP} \}

\[ \text{Listing 3: Indexed reading from and writing to a buffer.} \]

\[ A = c_i \text{low} \rightarrow \text{writeActuator}(\text{low}) \rightarrow wfnp \rightarrow \text{SKIP} \]

\[ \Box \]

\[ c_i \text{middle} \rightarrow \text{writeActuator}(\text{middle}) \rightarrow wfnp \rightarrow \text{SKIP} \]

\[ \Box \]

\[ c_i \text{high} \rightarrow \text{writeActuator}(\text{high}) \rightarrow wfnp \rightarrow \text{SKIP} \]

\[ \Box \]

\[ wfnp \rightarrow \text{SKIP} \]

\[ C = \text{timeout} \rightarrow wfnp \rightarrow \text{SKIP} \]

\[ \text{SafetyCriticalSystem} = (A_X \uparrow_{y_0 \lor y_1 \lor y_2} (A_{x_1} \uparrow_{y_1 \lor y_2} A_{x_2} \uparrow_{y_1 \lor y_2} (S_0_{x_1} \downarrow_{y_0 \lor y_1 \lor y_2} S_1_{y_0 \lor y_1 \lor y_2} S_2_{y_0 \lor y_1 \lor y_2}))) \]

in total 100 \( \mu \)s and the period is 1 ms then the time-out should expire at less than 900 \( \mu \)s.

We continue with the definition of the relational semantics of the enhanced extended half-synchronous operator \( \uparrow \). Let \( I_i \) be a non-empty subset of the set \( I = \{1, 2, \ldots, m\} \). Let \( J_j \) be a non-empty subset of the set \( J = \{1, 2, \ldots, n\} \). Let \( P = \{P_1, \ldots, P_m\} \) be the set of processes containing an indexed asynchronous \( i_{t_i} - \text{action} \). Let \( Q = \{Q_1, \ldots, Q_n\} \) be the set of processes containing an indexed asynchronous \( j_{t_j} - \text{action} \). Then, in Figure 1 we give

- the semantics of the enhanced extended half-synchronous operator,
- if we need more than one process \( P \) we use \( P_i \) otherwise we use \( P \),
- the alphabets of \( P, P_1, \ldots, P_m, Q_1, \ldots, Q_n \) are denoted as \( X, X_1 \cdot \cdot \cdot, X_m, Y_1 \cdot \cdot \cdot, Y_n \), respectively, and
- for ease of reading, we omit the alphabets for the extended half-synchronous operator, therefore \( P_i \downarrow_{P_j} \) is denoted as \( P_i \downarrow P_j \) (likewise for \( Q_i \) and \( Q_j \)).

The rules \( R_1 \) through \( R_7 \) are given in Figure 1, where

\( R_1 \) specifies that two indexed writing actions with different index sets are asynchronous.

\( R_2 \) specifies that for two or more writing actions to the same channel by two or more processes are synchronous\(^6\) if the labels of the writing actions have an index in common and are identical as far as the labels without the index sets are concerned, as, for example, in Listing 2.

\(^6\)This is similar to a voting design pattern, where a majority of voters is necessary to reach a decision.
\[ R_1 : \]
\[
P_1^{c_{ij_1}x^T} \cdot P_1' \xrightarrow{c_{ij_2}x^T} P_2' \cdot (P_1 \upharpoonright P_2) \rightarrow (P_1' \upharpoonright P_2') \oplus (P_1 \upharpoonright P_2'), \quad I_1 \cap I_2 = \emptyset, \]

\[ R_2 : \]
\[
P_1^{c_{ij_2}x^T} \cdot P_1' \xrightarrow{c_{ij_2}x^T} P_2' \cdot \ldots \cdot P_k' \xrightarrow{c_{ij_k}x^T} P_k' \cdot (P_1 \upharpoonright P_2 \upharpoonright \ldots \upharpoonright P_k) \rightarrow (P_1' \upharpoonright P_2' \upharpoonright \ldots \upharpoonright P_k'), \quad I = I_1 \cap I_2 \cap \ldots \cap I_k \neq \emptyset, \]

\[ c_{ij_n} x : T \in X_n, n \notin \{1, \ldots, k\} \Rightarrow I \cap I_n = \emptyset \]

\[ R_3 : \]
\[
P_1^{c_{ij_1}x^T} \cdot P_1' \xrightarrow{c_{ij_1}y^T} P_1' \cdot P_2' \xrightarrow{c_{ij_2}y^T} P_2' \cdot \ldots \cdot P_k' \xrightarrow{c_{ij_k}y^T} P_k', \quad i \notin \{1, \ldots, k\}, y \notin X, \]

\[ c_{ij_j} x : T \in X_j, j \in \{1, \ldots, k\}, I_1 \subseteq I_1 \cup \ldots \cup I_k, \quad i \text{ not in } y \]

\[ R_4 : \]
\[
P_1^{c_{ij_1}x^T} \cdot P_k \xrightarrow{c_{ij_k}x^T} P_k', Q_1 \xrightarrow{c_{ij_1}x^T} Q_1', \ldots , Q_k \xrightarrow{c_{ij_k}x^T} Q_k' \cdot (P_k \upharpoonright Q_1 \upharpoonright Q_j) \rightarrow ((P_k' \upharpoonright Q_1' \upharpoonright Q_j) \oplus (P_k \upharpoonright Q_1 \upharpoonright Q_j)) \rightarrow (P_k' \upharpoonright Q_1' \upharpoonright Q_j) \]

\[ R_5 : \]
\[
P_1^{c_{ij_1}x^T} \cdot P_1' \xrightarrow{c_{ij_k}x^T} Q_1' \cdot \ldots \cdot Q_k \xrightarrow{c_{ij_k}x^T} Q_k' \cdot P_1 \upharpoonright Q_1 \upharpoonright \ldots \upharpoonright Q_k \rightarrow P_1' \upharpoonright Q_1' \upharpoonright \ldots \upharpoonright Q_k' \]

\[ c_{ij_i} x : T \notin X_n, n \notin \{1, \ldots, k\} \]

\[ R_6 : \]
\[
P_1 \xrightarrow{c_{ij_i}x^T} P', Q_1 \xrightarrow{c_{ij_i}x^T} Q_1', \ldots , Q_k \xrightarrow{c_{ij_i}x^T} Q_k' \cdot (P_1 \upharpoonright Q_1 \upharpoonright Q_j) \rightarrow (P_1' \upharpoonright Q_1' \upharpoonright Q_j) \]

\[ R_7 : \]
\[
Q_1 \xrightarrow{c_{ij_i}x^T} Q_1', Q_2 \xrightarrow{c_{ij_i}x^T} Q_2', \ldots , Q_k \xrightarrow{c_{ij_i}x^T} Q_k', y \notin c_{ij_i} x : T, c_{ij_i} x : T \in Y_j \}

\( \text{Figure 1. Relational semantics of the enhanced extended half-synchronous operator for a specification comprising the processes } P_1, \ldots , P_n, Q_1, \ldots , Q_n. \)

\( R_3 \) specifies that a writing action with index \( i \) in its index set cannot be performed if one or more processes that contain this writing action with index \( i \) in their index set are not in a state where this writing action can be performed.

\( R_4 \) specifies that an indexed reading action must always be preceded by a related\(^7\) writing action and the reading actions with different indexes are asynchronous.

\( R_5 \) specifies that a set of indexed reading actions with the same index must be preceded by a related writing action and that these reading actions are synchronous.

\( R_6 \) specifies that indexed reading actions must be preceded by a related writing action.

\( R_7 \) specifies that indexed reading actions with the same index are synchronous.

\(^7\)Based on [6], two actions are related if and only if

- one action contains the \( i_n \), precisely once and does not contain the \( \iota_n \), and the other action contains the \( \iota_n \) precisely once and does not contain the \( i_n \),
- the prefix of the labels of both actions with respect to the \( i_n \) and \( \iota_n \) is identical and
- the postfix of the labels of both actions with respect to the \( i_n \) and \( \iota_n \) is identical.
Remark 1. Clearly, both the \( i_{1,2} \)-action and the \( i_{3} \)-action are prone to deadlocks. As an example, if one process contains \( c \{ i_{1,2} \} x : T \) followed by \( c \{ i_{3,4} \} x : T \) and another process contains the same actions in reversed order the two processes may deadlock. Because we consider processes represented by consistent graphs only, such a process definition is inhibited.

In the next section we discuss the impact of these relational semantics on the VRSP.

3. The VRSP of the Enhanced Extended Half-Synchronous Alphabetised Parallel Operator

As we are taking into account pairs of consistent graphs only, \( c_{\infty} \) in one process without a related \( c_{i_{n}} x : T \) in any other process is inhibited, because the process may end in a deadlock and the deadlock violates the consistency requirements. Also, the processes containing indexed writing actions must be pairwise consistent and the processes containing indexed reading actions must be pairwise consistent. The definition of consistency for the EEVRSP is given in Definition 1. The EEVRSP is defined on page 12 just below Definition 1.

We start with a simple example showing a majority vote specified in Listing 2. This example is smaller than the example given in Listing 3, because otherwise the figure would become unreadable. In Figure 2, we give the graphs \( G_{1} \) and \( G_{2} \) representing the specification of the writing processes \( P_{1} \) and \( P_{2} \), and the EEVRSP of \( G_{1} \) and \( G_{2} \), \( G_{1,2} = G_{1} \otimes^{i} G_{2} \), leaving out the not relevant actions (the dots, \( \cdots \) ) of these processes.

![Figure 2](image-url)

**Figure 2.** Graphs \( G_{1}, G_{2}, \) and \( G_{1,2} = \otimes^{i=1}_{2} G_{i} \) representing processes \( P_{1} \) and \( P_{2} \), and their parallel composition \( (P_{1} \otimes x_{1}, P_{2} \otimes x_{2}) \) of Listing 2.

In Figure 3, we give the graphs \( G_{1}, G_{2} \) and \( G_{3} \), and the EEVRSP of \( G_{1}, G_{2} \) and \( G_{3} \), \( G_{1,2,3} = \otimes^{i=1}_{3} G_{i} \) representing the specification of the writing processes \( P_{1}, P_{2} \) and \( P_{3} \). The EEVRSP of these graphs shows clearly that whenever an indexed writing action has a majority, it will be selected for execution, leading to a transition to the next state. The \( wfnp \)-action makes sure that all processes involved in this subsystem are pairwise consistent.
Figure 3. Graphs $G_1, G_2, G_3,$ and $G_{1,2,3} = \bigoplus_{i=1}^{3} G_i$ representing processes $P_1, P_2, P_3,$ and their parallel composition $(P_{1_x1_x2_x} P_2)_{x_1\leftarrow x_2} P_3$ of Listing 2.

In Figure 4, we give the graphs representing the interaction of the graph $G_{1,2,3}$ representing the parallel execution of the three writing processes $P_1, P_2$ and $P_3,$ and the reading process $P_4$ of Listing 2.

Figure 4. Graphs $G_{1,2,3}, G_4,$ and $G_{1,2,3,4} = \bigoplus_{i=1}^{4} G_i$ representing processes $(P_{1_x1_x2_x} P_2)_{x_1\leftarrow x_2} P_3, P_4,$ and their parallel composition $((P_{1_x1_x2_x} P_2)_{x_1\leftarrow x_2} P_3)_{x_1\leftarrow x_2\leftarrow x_3} P_4$ of Listing 2.
Remark 2. The EEVRSP of the graphs $G_1, \ldots, G_4$, $G_{1.2.3.4} = \bigotimes_{i=1}^4 G_i$ gives a performance gain because the length of the graph $\sum_{i=1}^4 G_i$ equals eight, whereas the length of the graph $G_{1.2.3.4}$ equals three.

We have defined the notion of consistency of graphs under EVRSP in [6] and adjust it here to consistency of graphs under the EEVRSP. Because we introduce the notion of an indexed writing action in this paper, we give the following definition for this indexed writing action. An indexed writing action $c_{i,k}^{i}x:T$ is represented by the set of arcs $a$ with $\mu(a) = (u, v)$ and $\lambda(a) = \{c_{i,k}^{i}x : T \mid i \in I_i\}$. Furthermore, we adapt the definition of the path read and path write cardinality defined in [6] such that it meets the requirements for the EEVRSP.

The number of occurrences of an indexed write action $c_{i,k}^{i}x:T$ in the path $P$ with respect to an index $k \in I_i$, is called the path write cardinality of a path with respect to $c_{i,k}^{i}x:T$ for the index $k$, denoted as $P(c_{i,k}^{i}x:T)$.

The number of occurrences of an indexed read action $c_{i,k}^{i}x:T$ in the path $P$, is called the path read cardinality of a path with respect to $c_{i,k}^{i}x:T$, denoted as $P(c_{i,k}^{i}x:T)$.

Definition 1. Components $G_i$ and $G_j$ are consistent if and only if the following three requirements apply:

1. $\rho_{G_i}(G_iO_iG_j) = G_j$ and $\rho_{G_j}(G_iO_iG_j) = G_i$.
2. $S'(G_iO_iG_j) = S'(G_i) \times S'(G_j)$ and $S''(G_iO_iG_j) = S''(G_i) \times S''(G_j)$.
3. Whenever $Q$ an $R$ are paths from the source to the sink of $G_i$ ($G_j, G_iO_iG_j$), $Q(c_{i,k}^{i}x:T) = R(c_{i,k}^{i}x:T)$ for all $k \in I_i \cup I_j$ and $Q(c_{i,k}^{i}x:T) = R(c_{i,k}^{i}x:T)$.

The EEVRSP of $G_i$ and $G_j$, $G_iO_iG_j$ is closely related to the VRSP and EVRSP of $G_i$ and $G_j$, and is constructed in two stages, where the definition of the intermediate stage of DVRSP is identical to the intermediate stage of EEVRSP, $G_iO_iG_j = G_iO_iG_j$, with

- $v_xw_x \in A_{i,j}$ is an arc with operator $\otimes_n$ in $l(v_xw_x) = l_r$,
- $P_n$ is a path from the source of $G_iO_iG_j$ through $w_x$,
- $P_m$ is the path from the source to the sink of $G_iO_iG_j$.

As in [6], we modify the Cartesian product $G_iO_iG_j$ according to the existence of synchronising arcs, but now with the extra constraint that indexed writing actions containing an index $k \in I_i, I_j$ are synchronous and indexed writing actions containing an index $k \in I_i$ and $k \notin I_j$ (k \notin I_i and $k \in I_j$) are asynchronous.

The first step in this modification consists of ignoring the synchronising arcs while forming arcs in the product, but additionally combining pairs of synchronising arcs of $G_i$ and $G_j$ into one arc, yielding the intermediate product which we denote by $G_iO_iG_j$.

To be more precise, $G_iO_iG_j$ is obtained from $G_iO_iG_j$ by first ignoring all except for the so-called asynchronous arcs, i.e., by only maintaining all arcs $a \in A_{i,j}$ for which $\mu(a) = ((v_j, v_j), (w_i, w_j))$, whenever $v_j = w_j$ and $\lambda(a) \notin L_j$, as well as all arcs $a \in A_{i,j}$ for which $\mu(a) = ((v_i, v_j), (w_i, w_j))$, whenever $v_i = w_i$ and $\lambda(a) \notin L_i$. This set of arcs...
is denoted by $A_{i,j}^u$. Additionally, we add arcs that replace synchronising pairs $a_i \in A_i$ and $a_j \in A_j$ with $\lambda(a_i) = \lambda(a_j)$. If $\mu(a_i) = (v_i, w_i)$ and $\mu(a_j) = (v_j, w_j)$, such a pair is replaced by an arc $a_{i,j}$ with $\mu(a_{i,j}) = ((v_i, v_j), (w_i, w_j))$ and $\lambda(a_{i,j}) = \lambda(a_i)$. The set of these so-called synchronous arcs of $G_i \square G_j$ is denoted by $A_{i,j}^\ast$.

The second step in this modification consists of removing (from $G_i \square G_j$) the vertices $(v_i, v_j) \in V_{i,j}$ and the arcs $a$ with $\text{tail}(a) = (v_i, v_j)$, whenever $(v_i, v_j)$ has level $> 0$ in $G_i \square G_j$ and $(v_i, v_j)$ has level $0$ in $G_i \square G_j$ and all arcs $v_y w_y \in A_{i,j}$ for which there exists a related arc $v_y w_y \in A_{i,j}$ with operator $l_{i,j}$ in $l(v_x w_x)$ for which there does not exist at least $n$ related arcs $v_y w_y$ with operator $l_{i,j}$ in $l(v_y w_y)$ with $v_y w_y < v_x w_x$. This is then repeated in the newly obtained graph, and so on, until there are no more vertices at level $0$ in the current graph that are at level $> 0$ in $G_i \square G_j$.

The resulting graph is called the EEVRSP of $G_i$ and $G_j$, denoted as $G_i \square G_j$.

For $k \geq 3$, the EEVRSP $G_1 \square G_2 \square \cdots \square G_k$ is defined recursively as $((G_1 \square G_2) \square \cdots) \square G_k$.

**Remark 3.** Because arcs $v_i w_i$ with $i \in l(v_i w_i)$ are indexed, the arcs $v_i w_i$ with different indexes represent asynchronous actions, because they have different labels due to different indexes.

In Figure 5 we show an example based on an example from [6] that shows the stages of the EEVRSP with respect to the delayed reading actions. Figure 5.a shows the Cartesian Product of graphs $G_1$ and $G_2$, where $G_1$ represents only one indexed writing action and $G_2$ represents a series of two indexed reading actions. Obviously, the index set of the writing action may contain more than one element. The dotted arcs in Figure 5.b are selected for removal. After removing the dotted arcs we have Figure 5.c. In Figure 5.c the vertices $u_3$ and $u_5$ and their arcs are removed because in the Cartesian Product (Figure 5.a) they have an in-degree greater than zero, whereas in Figure 5.c the vertices $u_3$ and $u_5$ have an in-degree of zero. After removal of the vertices $u_3$ and $u_5$ and their arcs in Figure 5.c we get Figure 5.d.

![Figure 5.](image-url)
4. Discussion and Conclusions

In this paper we have discussed the new $X \Downarrow Y$ operator, the new $i_i$-action and the $i_a$-action, which enables the possibility to model a majority vote in an easy and straightforward manner. The $X \Downarrow Y$ operator together with the $i_i$-action replaces a series of choices, reducing the size of the specification and making the design less error prone. As for the EEVRSP, the writing processes do not have to wait for the reading processes to synchronise and the designer has the choice to have synchronous as well as asynchronous writes to a channel. Furthermore, the overall design cycle will gain because the improved description on design level will lead to less effort for the implementation and less effort for testing. To summarise these advantages we have:

1. it eases the design in case the application needs both synchronous as well as asynchronous writes to a channel,
2. it gives maximum flexibility by indexing the writing and reading actions,
3. it allows multiple write actions, both synchronous as well as asynchronous to the same channel,
4. the advantages of the VRSP and EVRSP are preserved.

The first, second and third advantage makes the design less error-prone and therefore the design phase needs less time. The fourth advantage leads to an application for which the end-to-end execution time of the application is reduced and due to the reduction of the number of context switches, the overall utilisation of the processor is reduced.

Acknowledgement

The authors would like to express their gratitude to the anonymous reviewers for the very useful suggestions and comments.

The research of the first author has been funded by the InHolland University of Applied Science, Alkmaar, The Netherlands.

References


