A PROCESS ALGEBRAIC APPROACH TO FAULT-TOLERANCE

PADMANABHAN KRISHNAN and BRUCE McKENZIE
Department of Computer Science, University of Canterbury,
Christchurch, New Zealand
E-Mail: paddy@cosc.canterbury.ac.nz
bruce@cosc.canterbury.ac.nz

ABSTRACT

A process algebraic approach to the specification of fault tolerant systems is described. As replication is inevitable for fault tolerance, we extend the process algebra of Aceto and Hennessy with a replication operator. An operational semantics for replicated processes with majority voting is developed. We model faults as action refinement and show how the effect of faults on a replicated system can be modelled.

1 Introduction

In this paper we present a process algebraic approach to the semantics of fault tolerant systems. A fault is an event which causes the system to deviate from its expected behaviour. Faults may be due to software errors (bugs), hardware design errors, physical malfunctions due to factors such as fatigue. A system specification usually makes certain assumptions about the environment in which it operates. If the environment behaves in an unexpected fashion, the behaviour of the system cannot be predicted. Such changes in the operating environment may also be called faults.[1]

Safety critical systems (also called fault tolerant systems) are those that can continue to exhibit their specified behaviour in the presence of faults. The main aspects in building fault tolerant systems include detection, diagnosis, masking and containment. Strategies for fault detection/diagnosis depends on what is classified as a fault. Once a fault has been identified/detected it is necessary to protect the subsequent system behaviour from it (masking) and the fault must not be permitted to propagate (containment).

Users of a system are rarely concerned with the cause of a fault. They are primarily interested in the final behaviour of the system. Therefore, the techniques used in building fault tolerant systems must be hidden from the final behaviour.

It has been recognised that any attempt to mask systems from faults requires some form of replication. For example, if a system has to be impervious to single processor failures, at least two processors (a primary and a standby) are necessary. It is also essential to relate the computations being performed by the replicated components. If the stand by processor does not have an accurate view of the computation being performed, its behaviour after the failure of the primary unit may not be acceptable. Failures are only one type of fault (and usually benign compared to the other types). In communicating systems, messages may be corrupted, duplicated, omitted etc. Masking these types of faults can also be handled by replication.

The number of replicated components depends among other things on the types of faults, and the number of faults the system has to handle. Even though there is no single uniform technique to build fault tolerant systems, a common strategy is to replicate the system and obtain reliable results via majority voting.[2,3] In this paper we develop a calculus for systems with replicated systems and majority voting.

While various protocols (algorithms) which hide fault tolerance from the user exist, there is very little by the way of an algebraic characterisation. Mancini and Pappalardo[4] define a correctness criteria for a trace semantics of replicated CSP processes but they do not consider general replication. Furthermore they do not model faults. Process algebras such as ACP,[5] CCS,[6,7] CSP[8] have played an important role in the development of theories for concurrency. The relevance of these theories in real systems is illustrated by the ISO specification language LOTOS.[9,10] In this paper we extend a calculus similar to ACP with replication (RCP).

As there is no single replication factor to handle different faults, processes in RCP exhibit different "fault properties". For example, a single replication can handle failure faults but is not sufficient for message garbling. Let a represent a correct message and b a garbled message. If one considers single replication of a with majority voting the behaviour of the system under the garbling is unpredictable. On the other hand if the replication factor is greater than two and only one of the messages is garbled, a correct behaviour is exhibited. Therefore to prove fault properties of a system one needs to model faulty actions of a system. Formal models of fault behaviours have focussed on probabilities.[11] While probabilistic process algebras exist,[12] their usefulness is limited as it requires the coding of all possible faults into the system.

The notion of action refinement[13,14] has been used to develop a theory of "incremental" process development. A fault (of a particular action) can be modelled as an action being refined to the faulty behaviour. While the refined process has the faults encoded, the use of refinement results in a theory where there is a separation of concerns. The unrefined process deals with replication while the refinement operators specify the faults. It is for this reason we use action refinement to model faults.

However the current theories of action refinement place restrictions (such as disjoint behaviours, conflict free event structures) on the types of refinement. We will be forced to relax some of these conditions to model realistic faults. For example, message garbling cannot assume disjoint behaviours. In this paper we only consider the operational behaviour of processes under refinement. The denotational models for processes with refinement are available only for refinements which satisfy certain constraints. Further research is necessary to develop denotational models for RCP.

In the next section we define the syntax and semantics of RCP. In section 3 we consider refinement.

2 Replicated Processes

We assume, as in previous work,[5–7] a countable set of atomic actions Λ. The usual combinators of ";", "$|$ " and "+" are extended with $\amalg$ to indicate replication. The set of processes is defined as follows

$$\mathrm{P} ::= nil \mid \delta \mid \mu \in \Lambda \mid (\mathrm{P;P}) \mid (\mathrm{P} \mid \mathrm{P}) \mid (\mathrm{P} + \mathrm{P}) \mid (\mathrm{P} \amalg \mathrm{P})$$

The two special processes nil and δ represent termination and deadlock respectively. Thus neither can exhibit any action. The atomic action terminates after exhibiting itself. ; represents sequential composition, $|$ parallel execution, + non determinism and $\amalg$ replication. In this paper we only consider finite processes. $\mathcal{P}_r$ denotes all finite processes with replication, while $\mathcal{P}$ denotes finite processes without replication (i.e., the Aceto and Hennessy[15] subset of $\mathcal{P}_r$). For purely behavioral specifications, sequential composition can be replaced by action prefix. But sequential composition is necessary when faults are introduced in the system.

As Aceto and Hennessy[15] explain, a set $\surd$ is necessary to develop an operational semantics for ACP and hence RCP. $\surd$ represents the set of all terminated processes with nil indicating termination.

Definition: 1 *Define $\surd$ to be the smallest set satisfying the following conditions.*

- $nil \in \surd$
- *if P, $Q \in \surd$, $(P;Q)$, $(P \mid Q)$, $(P + Q)$ and $(P \amalg Q) \in \surd$.*

The operational semantics consists of two parts. The first specifies the internal moves of a replicated system and the process of obtaining votes (the relation $\rightarrowtail$). The second specifies the interaction of the system with the environment ($\longrightarrow$), viz., exhibiting the action that has received the maximum number of votes. Figure 1 is a pictorial description of our model.

This is similar to the notion of high-level and low-level transitions introduced in Gorrieri *et.al.*[16] but they do not consider voting. Their main concern is the decomposition of atomic actions at an implementation level. In our semantics, actions are atomic for both the internal and the external moves.

To capture the state of the voting machine, we need the following auxiliary definitions.

Definition: 2 Define Residual_Process as the set of partial functions: $\Lambda \rightarrow$ (Integer $\times\, \mathcal{P}_r$)

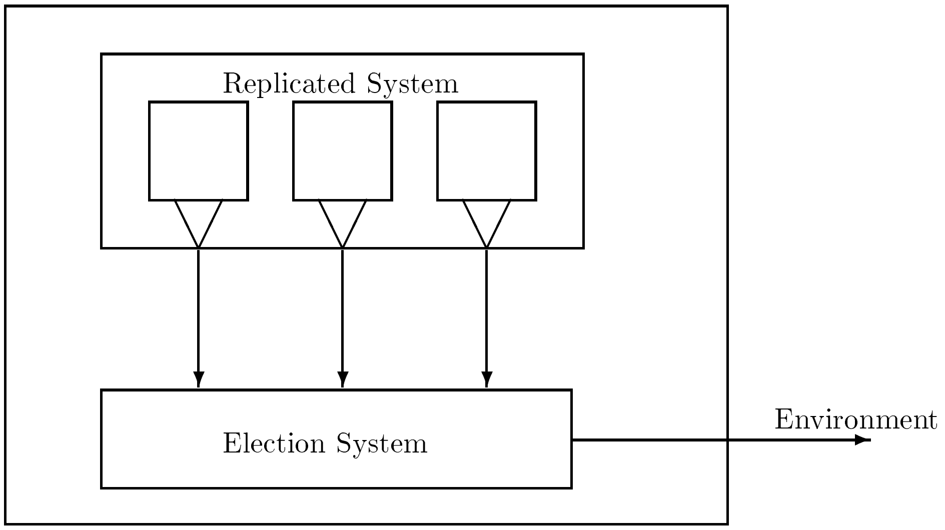

Figure 1: Fault Tolerant System

Definition: 3 For S an element of Residual_Process and a an element of Λ, define projection functions π_1 and π_2 as follows. If S(a) = $\langle$n,P$\rangle$, π_1(S(a)) = n and π_2(S(a)) = P.

Each process indicates an action it can perform and the process it evolves to under the action. This is noted by the voting machine. For each action, the voting machine stores the number of votes and the process to be executed after exhibiting the chosen action. The current state of votes obtained is indicated by an element of Residual_Process .

The effect of combining two votes is defined by $\uplus$ as follows.

Definition: 4 For S1 and S2 elements of Residual_Process , define S1 $\uplus$ S2 as S where S(a) = $\langle$ π_1(S1(a))+π_1(S2(a)), π_2(S1(a)) $\amalg$ π_2(S2(a)) $\rangle$.

The $\times_r$ ($\times_l$) operators extend to the right(left) the residual processes with a parallel branch. Its usage will become clear when the internal moves are formally defined.

Definition: 5 Let S be an element of Residual_Process and P an element of $\mathcal{P}_r$. Define $\times_r$ :: Residual_Process , $\mathcal{P}_r \rightarrow$ Residual_Process .

S $\times_r$ P = S' where S'(a) = $\langle$ π_1(S(a)), (π_2(S(a))$|$ P) $\rangle$.

S $\times_l$ P = S' where S'(a) = $\langle$ π_1(S(a)), (P $|$ π_2(S(a))) $\rangle$.

An operation $+_r$ similar to $\times_r$ is defined for sequential composition.

Definition: 6 Let S be an element of Residual_Process and P an element of $\mathcal{P}_r$. Define $+_r$ :: Residual_Process , $\mathcal{P}_r \rightarrow$ Residual_Process .

S $+_r$ P = S' where S'(a) = $\langle$ π_1(S(a)), (π_2(S(a));P) $\rangle$.

Definition: 7 Define the ready set of P, rd(P), as the set of actions P can exhibit initially. rd(P) = $\{a \mid \exists$ P', P $\xrightarrow{a}$ P'$\}$.

Definition: 8 *The internal moves of a replicated system is defined to be the smallest relation* $\rightarrowtail \subseteq \mathcal{P}_r \times$ Residual_Process *satisfying the axioms in figure 2.*

An atomic action yields a single vote following which it terminates. The usual rules for sequential composition, choice and parallel apply here. Thus, we adopt an operationally "interleaving" model for the parallel combinator. The first rule for replication takes into account the votes from its two branches. This is similar to SCCS,[17] i.e, there is no asynchronous evolution. However replication differs from SCCS in that if one of the branches cannot exhibit any action (terminated or deadlocked), it is discarded while in SCCS (*nil* $\times$ P) will not be able to proceed.

The external transition rules are derived from the internal moves. This requires identifying the action that has received maximum number of votes.

Definition: 9 As the current state of the voting machine is indicated by an element of Residual_Process , for S an element of Residual_Process , define Voted_Action (S) = $\{a \mid \forall\, b \in \Lambda,\, \pi_1(\mathrm{S}(a)) \geq \pi_1(\mathrm{S}(b))\}$.

Definition: 10 Define $\longrightarrow$ as the smallest set satisfying the axioms in figure 3.

The first transition rule describes the process of exhibiting the action that received the maximum number of votes. As only the residual process for the chosen action is selected, containment is achieved. The other rules are as usual. There is no direct external transition for replicated processes. Thus replicated processes can only evolve via voting.

Example 1 *Consider the process (a;P $\amalg$ b;Q $\amalg$ a;R). As the action a receives the maximum votes, the system exhibits action a and evolves to (P $\amalg$ R). The "faulty" action b and its residual process Q is discarded.*

2.1 Behavioral Relations

In this section we describe the properties for replicated processes.

Definition: 11 *A preorder relation $\preceq$ on $\mathcal{P}_r$ is defined as: P $\preceq$ Q iff P $\xrightarrow{a}$ P' implies $\exists$ Q', Q $\xrightarrow{a}$ Q' and P' $\preceq$ Q'.*

If $\preceq$ is a symmetric relation we indicate it by $\simeq$ called the bisimilar relation.

In this paper we do not distinguish between a terminated process and a deadlocked process. Thus $\preceq$ is not a precongruence.

Action	$a; P \rightarrowtail \{\langle a, \langle 1, P\rangle\rangle\}$
Sequential Composition$_1$	$\dfrac{\mathrm{P} \rightarrowtail \mathrm{S},\ \mathrm{S'} = +_r(\mathrm{S},\mathrm{Q})}{\mathrm{P};\mathrm{Q} \rightarrowtail \mathrm{S'}}$
Sequential Composition$_2$	$\dfrac{\mathrm{P} \in \surd,\ \mathrm{Q} \rightarrowtail \mathrm{S}}{\mathrm{P};\mathrm{Q} \rightarrowtail \mathrm{S}}$
Non-deterministic Choice	$\dfrac{\mathrm{P} \rightarrowtail \mathrm{S}}{\begin{array}{c}\mathrm{P} + \mathrm{Q} \rightarrowtail \mathrm{S} \\ \mathrm{Q} + \mathrm{P} \rightarrowtail \mathrm{S}\end{array}}$
Parallelism$_1$	$\dfrac{\mathrm{P} \rightarrowtail \mathrm{S},\ \mathrm{S'} = \mathrm{S} \times_r \mathrm{Q}}{\mathrm{P} \mid \mathrm{Q} \rightarrowtail \mathrm{S'}}$
Parallelism$_2$	$\dfrac{\mathrm{Q} \rightarrowtail \mathrm{S},\ \mathrm{S'} = \mathrm{S} \times_l \mathrm{P}}{\mathrm{P} \mid \mathrm{Q} \rightarrowtail \mathrm{S'}}$
Replication$_1$	$\dfrac{\mathrm{P} \rightarrowtail \mathrm{S}_1,\ \mathrm{Q} \rightarrowtail \mathrm{S}_2,\ \mathrm{S}=\mathrm{S}_1 \uplus \mathrm{S}_2}{\mathrm{P}\ \mathrm{II}\ \mathrm{Q} \rightarrowtail \mathrm{S}}$
Replication$_2$	$\dfrac{\mathrm{P} \rightarrowtail \mathrm{S},\ \mathrm{rd}(\mathrm{Q}) = \emptyset}{\begin{array}{c}\mathrm{P}\ \mathrm{II}\ \mathrm{Q} \rightarrowtail \mathrm{S} \\ \mathrm{Q}\ \mathrm{II}\ \mathrm{P} \rightarrowtail \mathrm{S}\end{array}}$

Figure 2: Internal Moves

System Transition	$\dfrac{\text{P} \rightarrowtail \text{S},\ a\ \in \textsf{Voted\_Action}\ (\text{S}),\ \text{P'}{=}\pi_2(\text{S(a)})}{\text{P} \xrightarrow{a} \text{P'}}$
Sequential Composition$_1$	$\dfrac{\text{P} \xrightarrow{a} \text{P'}}{\text{P;Q} \xrightarrow{a} \text{P';Q}}$
Sequential Composition$_2$	$\dfrac{\text{P} \in \surd\ ,\ \text{Q} \xrightarrow{a} \text{Q'}}{\text{P;Q} \xrightarrow{a} \text{Q'}}$
Nondeterministic Choice	$\dfrac{\text{P} \xrightarrow{a} \text{P'}}{\begin{array}{c}\text{P} + \text{Q} \xrightarrow{a} \text{P'} \\ \text{Q} + \text{P} \xrightarrow{a} \text{P'}\end{array}}$
Parallelism	$\dfrac{\text{P} \xrightarrow{a} \text{P'}}{\begin{array}{c}\text{P} \mid \text{Q} \xrightarrow{a} \text{P'} \mid \text{Q} \\ \text{Q} \mid \text{P} \xrightarrow{a} \text{Q} \mid \text{P'}\end{array}}$

Figure 3: External Transition Rules

Proposition 1 *(nil* $\amalg$ *P)* $\simeq$*P* *(*$\delta$$\amalg$ *P)* $\simeq$*P*
(P $\amalg$ *Q)* $\simeq$*(Q* $\amalg$ *P)* *((P* $\amalg$ *Q)* $\amalg$ *R)* $\simeq$*(P* $\amalg$ *(Q* $\amalg$ *R))*
(P $\amalg$ *P)* $\simeq$*P* *(P + Q)* $\amalg$ *R* $\simeq$*(P* $\amalg$ *R) + (Q* $\amalg$ *R)*
P $\preceq$ *Q implies (P* $\amalg$ *Q)* $\preceq$ *Q*

As mentioned earlier $\simeq$ is not a congruence within the language. This is true even if we distinguish deadlock and termination. For example (a $\amalg$ a) $\simeq$a, but (a $\amalg$ a $\amalg$ b) $\not\simeq$ (a $\amalg$ b) This is not surprising as weak bisimulation is a congruence with respect to all operators except +. Replication generalised the notion of an internal move and $\amalg$ the generalisation of choice.

The following two propositions are expansion like theorems for replicated processes. The first takes a replication of choice into choice, while the second takes a replication of guarded processes into a choice of processes.

Proposition 2 *Let* $P = \sum_{i \in I} a_i; P_i$ *and* $Q = \sum_{j \in J} b_j; Q_j$. $P \amalg Q$ *is bisimilar to*

$$\sum_{i \in I, \forall j \in J : a_i \neq b_j} a_i; P_i + \sum_{j \in J, \forall i \in I : b_j \neq a_i} b_j; Q_j + \sum_{i \in I, \exists j \in J : a_i = b_j} a_i; (P_i \amalg Q_j)$$

The above can be extended to handle n-replication of guarded processes. The conversion of replication into choice is defined as follows.

Proposition 3 $\coprod_{i \in I} a_i; P_i \simeq \sum_{a \in A} a; (\coprod_{k \in K_a} P_k)$ *where* $A_I = \{a_i \mid i \in I\}$ *and* $count(a) = \#\{k \in I \mid a_k = a\}$ *and* $A = \{a \in A_I \mid (\forall j \in I. count(a) \geq count(a_j))\}$ *and* $K_a = \{k \in I \mid a_k = a\}$.

Definition: 12 *Let* $\sim^c$ *be the coarsest congruence relation contained in* $\simeq$*(*$\sim^c \subseteq \simeq$*).*

Proposition 4 *Let* $P \sim^c Q$. *If* $P \rightarrowtail S_p$ *then* $\exists\ Q \rightarrowtail S_q$ *such that for all* $a \in \Lambda$, $\pi_1(S_p(a)) = \pi_1(S_q(a))$ *and* $\pi_2(S_p(a)) \sim^c \pi_2(S_q(a))$.

Proof: We prove this by contradiction. If for any a, $S_p(a)$ is undefined and $S_q(a)$ is defined, consider R=$\coprod_k a$ where k is difference between the votes received by a in S_q and the maximum number of votes received by any event. Clearly Q in replication with R can exhibit an a while P cannot.

Let $a \in \Lambda$, $\pi_1(S_p(a)) = k_p$ and $\pi_1(S_q(a)) = k_q$. Without loss of generality assume $k_p > k_q$. By a similar argument to the above one can construct a process which when joined with Q can exhibit an action which is not possible for P. (The formal proof of this part is tedious.) As an example, consider P = ($a \amalg a \amalg b \amalg b$) and Q = ($a \amalg b$). The process ($c \amalg$ P) cannot exhibit c but ($c \amalg$ Q) can. □

The following two propositions are immediate consequences of the above theorem.

Proposition 5 $\sim^c$ *is completely axiomatised by any complete axiomatisation for* $\mathcal{P}$ *and the axioms:* $(nil \amalg P) = P$, $(\delta \amalg P) = P$, $\amalg$ *is commutative and associative.*

Note: The proof of soundness of the axiomatisation requires an extended syntax.

Proposition 6 *An extension to the Hennessy-Milner logic*[18] *with multiset of actions instead of actions characterises* $\sim^c$ *completely. That is, the logic obtained by replacing the modality* $\langle a \rangle$ *with the modality* $\langle m \rangle$ *where* a *is an action and* m *a multiset of actions is a sound and complete characterisation of* $\sim^c$.

It is also easy to show that finite replication is not sufficient to make a system totally fault tolerant.

Proposition 7 *Given a finitely replicated system* S*, there exists a process* P *such that* $(P \amalg S) \not\sim^c S$.

3 Modelling Faults

So far we have only considered replicated systems. We have not mode-led faults. As mentioned earlier, behaviour of systems under refinement can model a system with faults. Most of the theories of refinement place restrictions on the types of refinement (such as no autoparallelism, no empty process, disjoin behaviours). These restrictions are relaxed for fault modelling.

Another drawback in the traditional theories is that applying a refinement to a process alters all occurrences of the relevant action. As the degree of replication used depends on the number of faults the system has to withstand, fault refinement should alter only a bounded number of actions. Therefore refinement of a process is indexed by the number of potential faults that can be introduced.

Given a refinement operator ϱ and an integer n, the application of it to a process is defined in figure 4.

Example 2 *Given below are a few examples showing the modelling of faults via refinement. As faults do not always manifest themselves, they are defined as the choice between the correct action and the incorrect process.*

$$a\text{-}Omission = \varrho : \varrho\ (b) = \begin{cases} a + nil & b = a \\ b & otherwise \end{cases}$$

$$a\text{-}Deadlock = \varrho : \varrho\ (b) = \begin{cases} a + \delta & b = a \\ b & otherwise \end{cases}$$

$$a\text{-}Duplication = \varrho : \varrho\ (b) = \begin{cases} a + (a;a) & b = a \\ b & otherwise \end{cases}$$

$$
\begin{array}{|lcl|}
\hline
\mathrm{P}\ \varrho\ 0 & = & \mathrm{P} \\
\mathit{nil}\ \varrho\ n & = & \mathit{nil} \\
\delta\ \ \varrho\ n & = & \delta \\
a\ \ \varrho\ n & = & \varrho\ (a) \\
(\mathrm{P;Q})\ \varrho\ n & = & \sum\limits_{0\le i\le n} (\mathrm{P}\ \varrho\ i);\ (\mathrm{Q}\ \varrho\ n-i) \\
(\mathrm{P}\mid \mathrm{Q})\ \varrho\ n & = & \sum\limits_{0\le i\le n} (\mathrm{P}\ \varrho\ i)\mid (\mathrm{Q}\ \varrho\ n-i) \\
(\mathrm{P}\amalg \mathrm{Q})\ \varrho\ n & = & \sum\limits_{0\le i\le n} (\mathrm{P}\ \varrho\ i)\amalg (\mathrm{Q}\ \varrho\ n-i) \\
(\mathrm{P}+\mathrm{Q})\ \varrho\ n & = & (\mathrm{P}\ \varrho\ n)+(\mathrm{Q}\ \varrho\ n) \\
\hline
\end{array}
$$

Figure 4: Bounded Refinement

$$a-b\text{-}\mathit{Garbling} = \varrho : \varrho\ (c) = \begin{cases} a+b & c=a \\ c & \mathit{otherwise} \end{cases}$$

Refining a process with no fault yields the process, while processes that cannot exhibit any action cannot be refined. Refining sequential/parallel composition or replication results in the faults being distributed arbitrarily to both the subprocesses. For a process involving choice, there is no distribution of faults as non-determinism does not reduce the chance of a fault.

Definition: 13 The safety (fault tolerant) requirement for a system P can be defined as P $\simeq$ (P ϱ n).

Proposition 8 *If P $\preceq$ P ϱ n and (P ϱ n) $\preceq$ P then (P ϱ $n-1$) $\preceq$ P.*

Proposition 9 *As we do not consider communication, if P and ϱ are deadlock free, then (P ϱ n) is also deadlock free.*

Proposition 10 *Let P be an unreplicated proccess (i.e., element of $\mathcal{P}$).*
If for all ϱ and k ($\coprod\limits_{2k+1}$ P) ϱ k $\simeq$P then, the cardinality of rd(P) is less than or equal to 1, and if P $\xrightarrow{a}$ P', the cardinality of rd(P') is less than or equal to 1.

Proof: Let the cardinality of rd(P) be greater than 1. Then P $\xrightarrow{a}$ P$_1$ and P $\xrightarrow{b}$ P$_2$ with $a \neq b$. Thus ($\coprod\limits_{2k+1}$ P) $\xrightarrow{a}$ $\coprod\limits_{k+1}$P$_1$ and ($\coprod\limits_{2k+1}$ P) $\xrightarrow{b}$ $\coprod\limits_{k+1}$P$_2$ Let $k = 1$ and let $\varrho\ (a) = a + c$, where $c \notin$ rd(P). It is easy to see that (P $\amalg$ P $\amalg$ P) ϱ 1 can exhibit c (one of the branches selects an a the other b and the third c). Therefore (P $\amalg$ P $\amalg$ P) ϱ 1 $\not\simeq$ P.

If P $\xrightarrow{a}$ P$_1$ and P$_1$ does not satisfy the requirement, the existence of a refinement is proved by a straightforward induction. □

The significance of the above result is that majority replication is adequate, only if Process P is deterministic. By a combinatorial argument, a degree of replication for a particular cardinality of ready sets and the length of a computation can be obtained.

Example 3 *Consider for example, the process $P = a;(b+c)$. Applying ϱ for a single fault results in the process $\varrho\ (a);(b+c) + a;(\varrho\ (b) + \varrho\ (c))$.*

Clearly a single replication (i.e., $P \amalg P$) is not sufficient. If $\varrho\ (a) = d$, then ($P \amalg P$) can exhibit d with d obtaining a single vote from $\varrho\ (a)$ and a getting only one vote from the correct P.

$Q = (P \amalg P \amalg P)$ is also not sufficient. If $\varrho\ (b) = d$, after exhibiting a, Q can evolve to $(b+c) \amalg\ (b+c) \amalg\ (\varrho\ (b) + \varrho\ (c))$ which can exhibit a d. P replicated 4 times can with stand single faults.

For single failures, and processes with only one subterm P whose ready set is greater than 1, a replication factor of cardinality of rd(P) + 2 is sufficient.

For the process $(a+b);(c+d)$ to withstand a single fault, a replication factor of 8 is essential.

A proposition similar to proposition 7 can be proved for processes with replication. That is, given a finite replicated process, it cannot handle more than a bounded number of faults.

Proposition 11 *If rd(P) $\neq \emptyset$, there exists ϱ and n such that ($P\ \varrho\ n$) $\not\simeq$ P.*

Proof Outline: We explicitly construct such a refinement and n. We do this only for the replication operator. Consider the process (P ∐ Q). A (P ∐ Q) is deadlock free and not termination there is an action (say a) exhibited by it. Let a receive k_p votes from P and k_q votes from Q. Consider the $a-b$ garbling refinement where b is not the ready set of (P ∐ Q), and $n = k_p + k_q$. As (P ϱ k_p) ∐ (Q ϱ k_q) is a term in the refinement of (P ∐ Q), the action b receives exactly the same number of votes as a and hence (P ∐ Q) ϱ n can exhibit b. But as b was not in the ready set (P ∐ Q) cannot exhibit b. □

4 Conclusion

In this paper we have developed a calculus for replicated processes and showed how faults can be modelled. The ∐ combinator is basic in that it cannot be translated into the others in a congruent fashion. We have also found a novel use for action refinement. While our calculus is general, we envisage a restricted usage viz., ($\coprod_k$ P) ϱ n, where P is an unreplicated process. This is because usually there is a replication of identical processes. Further research is in progress to study the properties satisfied by the restricted calculus.

References

1. *IEEE Computer*, July 1990. All articles in this issue.

2. A. Avizienis. The N-version Approach to Fault-Tolerant Software. *IEEE Transactions on Software Engineering*, 11(12):1491–1501, Dec 1985.

3. F. Cristian. Understanding fault-tolerant distributed systems. *CACM*, 34(2):56–78, February 1991.

4. L. V. Mancini and G. Pappalardo. Towards a Theory of Replicated Processing. In M. Joseph, editor, *Proceedings of the symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems: LNCS 331*, pages 175–192. Springer Verlag, 1988.

5. J. A. Bergstra and J. W. Klop. Process Theory Based on Bisimulation Semantics. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, LNCS 354*, pages 50–122. Springer Verlag, 1988.

6. R. Milner. *A Calculus of Communicating Systems.* Lecture Notes on Computer Science Vol. 92. Springer Verlag, 1980.

7. R. Milner. *Communication and Concurrency.* Prentice Hall International, 1989.

8. C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall International, 1985.

9. T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. In P. H. J. van Eijk, C. A. Vissers, and M. Diaz, editors, *The Formal Description Technique LOTOS*, pages 23–73. North Holland, 1989.

10. P.H.J van Eijk and C.A. Vissers and M. Diaz. *The Formal Description Technique LOTOS.* North Holland, 1989.

11. M. J. Fisher and L. D. Zuck. Reasoning about Uncertainty in Fault-Tolerant Distrbuted Systems. In M. Joseph, editor, *Proceedings of the symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems: LNCS 331*, pages 142–158. Springer Verlag, 1988.

12. R. van Glabbeek, S. Smolka, B. Steffen, and C. Tofts. Reactive, Generative and Stratified Models of Probabilistic Processes. In *IEEE Symposium on Logic in Computer Science*, 1990.

13. M. Nielsen, U. Engberg, and K. S. Larsen. Fully Abstract Models for a Process Language with Refinement. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, LNCS 354*, pages 523–548. Springer Verlag, 1989.

14. R. van Glabbeek and U. Goltz. Equivalence Notions for Concurrent Systems and Refinement of Actions. In *Mathematical Foundations of Computer Science: LNCS 379*, pages 237–248. Springer Verlag, 1989.

15. L. Aceto and M. Hennessy. Termination, Deadlock and Divergence. In *Mathematical Foundations of Programming Semantics, LNCS 442*, pages 301–318. Springer Verlag, 1989.

16. R. Gorrieri, S. Marchetti, and U. Montanari. A2CCS: A Simple Extension of CCS for Handling Atomic Actions. In *13th Colloquim on Trees in Algebra and Programming, LNCS 259*, pages 258–270. Springer Verlag, 1988.

17. R. Milner. Calculus for Synchrony and Asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

18. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the Association of the Computing Machinery*, 32(1):137–161, January 1985.