An Introduction on Extensions of Process Algebra: Concurrent and Communicating Systems

Arpit[1], A'fza Binti Shafie[2] and Wan Fatimah Binti Wan Ahmad[3]
Department of Computer and Information Sciences
Universiti Teknologi PETRONAS, Bandar Seri Iskandar, 31750 Tronoh, Perak, Malaysia
[1]arpittabelabux@gmail.com, [2]afza@petronas.com.my and [3]fatimhd@petronas.com.my

Abstract—**This paper addresses a brief introduction on various extensions of process algebra for concurrent and communicating systems as an area of research in concurrency theory, the theory of parallel and distributed systems in computer science. Literature has tracked the growth of process algebra since CCS (Calculus of Communicating Systems) was formulated till RCCS (Calculus for reversible concurrent systems), in order to reflect the changes in computing environment and comparison has been sketched on the basis of this. A relationship between concurrency and communication has been explored in this literature. Literature presents some analytical reasoning on this matter. In this paper, present situation and current trends in process algebra is also considered and makes a reason why some models are being preferred for extensions by researchers over others. Some challenges for the future work are also mentioned in this literature.**

Keywords-Process algebra; distributed systems; reversibility probabilistic system; stochastic behavior and observational equivalence.

I. Introduction

The term process is coined with two terms; first one is "process" and other one is "algebra". Both terms, are used in many meanings, often inconsistent, so it has been tried to find something precise which those meanings share. Let's consider the word "process". It refers to the behavior of a system. A system is anything showing behavior, in particular the execution of a software system, the actions of a machine or even the actions of a human being. Behaviour is the total of events or actions that a system can perform, the order in which they can be executed and maybe other aspects of this execution such as timing or probabilities. Always, one describes certain aspects of behaviour, disregarding other aspects, so, an abstraction or idealization of the 'real' behaviour is considered. Rather, it can be said that an observation of behaviour and an action is the chosen unit of observation. Usually, the actions are thought to be discrete: occurrence is at some moment in time, and different actions are separated in time. This is why a process is sometimes also called a discrete event system.

The abstract meaning of "algebra" is to take axiomatic approach to talk about the behavior of some system. Mathematically an algebraic structure is defined as an ordered pair consisting of one or more sets called underlying sets or sorts and a set of operators defined on it, denoted by $<Z, *>$; where Z is a set of sorts and $*$ is a set of operators defined on Z. The various properties are exhibited by operators like communicative, associative, distributive, existence of inverse etc. Such properties create axioms for particular algebraic structure and these axioms together with calculations on them define algebra. In short algebra is way to explore one's behavior by performing calculations on existing behavior.

It is implicit that algebra explores system behavior by the means of calculations, so in this respect every algebra is process algebra because process refers behavior of system and indeed it is true. But one must ask, why process algebra is associated with concurrent systems only, as this has been stated in our abstract. To answer this question one has to pause for a while to explore the meaning of "behavior".

The behavior of a system is exactly what is observable and to observe a system is to communicate with it and this communication give raises the concurrent systems. In concurrent systems, communication between its constitute components or with external environment is general phenomenon. So, behavior of system is its ability to communicate with other systems and this communication constitutes a concurrent system.

II. Process algebra for Concurrent systems

Three variants of process algebra named as, Calculus of Communicating Systems (CCS), Communicating Sequential Processes (CSP) and Algebra of Communicating Processes (ACP) have been defined here. Each of three is intended to model communications between concurrent systems. In next three sections we describe each of them respectively.

A. Calculus of Communicating Systems (CCS)

Milner placed first milestone in theory of process algebra. He developed a process theory called CCS [1] for specifying and reasoning about "complex dynamic systems". It is based on the preemptive notation of handshaking communications. It has by far the most well established syntax and semantics that has been applied to a number of real world problems. Consequently it provides a good model to use to explain concepts prevalent in the study of concurrency and an excellent backdrop against which to study newer and more radical models, which are detailed in subsequent sections.

A.1. Syntax of CCS

In CCS the process is denoted by set of atomic actions and a set of operators. Let these atomic actions are in a set A and $\alpha \in A$, where α is an atomic action. There is also a complementary set of actions denoted by A' and $\alpha' \in A'$ iff

$\alpha \in A$. Generally, α is used for receiving message by port α and α' for sending message by port α'. There is also a special action called silent action, represented by τ. This represents the internal actions of process and it is not observable in external environment. So, the complete set of actions is defined by $Act = A \cup A' \cup \{\tau\}$.The syntax for basic CCS is as follows:

1. The set P of (concurrent) processes expressions is defined by $P: = A< a_1 \ldots a_n > \mid \alpha.P \mid \sum P_i \mid P_1 \mid P_2 \mid P [f] \mid nil \mid P \setminus L$. Where I is any finite indexing set, ranged by i.
2. $A< a_1 \ldots a_n >$ is a notation of standard agent.
3. $\alpha.P$ denotes a process which can only perform action α and behave like P.
4. $\sum P_i$ is a process which shows nondeterministic choices between P_i's .
5. $P_1 \mid P_2$ defines two concurrent processes.
6. $P \setminus L$ restricts the scope of name $\alpha \in L$ to process P.
7. P $[f]$ and $f = [a_1 / b_1 \ldots a_n / b_n]$ denotes the process derived from P by replacing action $b_1, b_2, \ldots\ldots\ldots, b_n$ in P with $a_1, a_2, \ldots\ldots\ldots, a_n$. This is called relabeling.
8. *nil* shows no action.

A.2 Semantics of CCS

The semantics of CCS is well defined by labeled transition system (LTS). LTS is defined as $(S, T, \{ \xrightarrow{t} : t \in T \})$.

Where S is a set of states, T is a set of transition labels and a transition relation $\xrightarrow{t} \subseteq S \times S$ for each $t \in T$.

$$\frac{}{\alpha.P \longrightarrow P} \quad (1) \qquad \frac{P_j \xrightarrow{\alpha} P_j'}{\sum P_i \xrightarrow{\alpha} P_j'} \quad (2)$$

$$\frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \quad (3) \qquad \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'} \quad (4)$$

$$\frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\alpha'} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \quad (5)$$

$$\frac{P \xrightarrow{\alpha} P'}{P'/L \xrightarrow{\alpha} P/L} \quad (\alpha, \alpha') \notin L \quad (6)$$

$$\frac{P \xrightarrow{f(\alpha)} P'}{P[f] \xrightarrow{\alpha} P[f]} \quad (7)$$

$$\frac{P \xrightarrow{\alpha} P'}{Q \xrightarrow{\alpha} Q'} \quad (A \overset{\text{def}}{=} P) \quad (8)$$

In CCS a process is considered to go through a number of states. The states are determined by the possible courses of actions in which the process is ready to engage. For example $\alpha.P$ is our process expression then it is permitted to do only α action and after α action it changes its state as $\alpha.P \xrightarrow{\alpha} P$.

In CCS internal actions are represented by τ. Internal actions are not observable to the environment. It is caused by the communication of sub-components of an agent. In CCS two processes are considered equivalent when they exhibit observation equivalence and this equality is termed as bisimulation [1, 2]. In bisimulation, any sequence of internal actions represented by τ^n and $n > 0$ can be equated to zero or no action, because they are internal not visible to environment. This bisimilarity is the main consequence placed by CCS.

B. Communicating Sequential Processes (CSP)

A very important contributor to the development of process algebra is by Tony Hoare. He introduced a language CSP (Communicating Sequential Process) [3] capable of modeling synchronous communication; which was later adopted by CCS. No model and semantics are provided by Hore. Further CSP, extended by Brookes and Roscoe [4], provides syntax and semantics for CSP model [4]. CSP allows the description of systems in terms of component processes that operate independently, and interact with each other solely through message passing communication. However, the *"Sequential"* part of the CSP name is now something of a misnomer, since modern CSP allows component processes to be defined both as sequential processes, and as the parallel composition of more primitive processes. The relationships between different processes, and the way each process communicates with its environment, are described using various process algebraic operators. Using this algebraic approach, quite complex process descriptions can be easily constructed from a few primitive elements.

B.1. Syntax of CSP

CSP provides two classes of primitives, namely as Events and Primitive processes [5].

1. Events represent communications or interactions. They are assumed to be indivisible and instantaneous. They may be atomic names (e.g. *on*, *off*), compound names (e.g. *port.open*, *port.close*), or input/output events (e.g. *mouse ? xy*, *screen !bitmap*).
2. Primitive processes represent fundamental behaviors: examples include *STOP* (the process that communicates nothing, also called deadlock), and *SKIP* (which represents successful termination).

CSP has a wide range of algebraic operators defined as follows:

1. The set P of (concurrent) processes expressions is defined by $P: = P \square Q \mid P \sqcap Q \mid P \,{}_b||_c\, Q \mid P \mid\mid\mid Q \mid P \backslash a \mid f[P] \mid f^{-1}[P] \mid x \rightarrow P_x$; where X, P, Q are the names of language CSP or sometimes called agents, ranged over set of names E and a, b, c are the actions, range over the set of actions A.
2. $P \square Q$ shows nondeterministic choice between P and Q.
3. $P \sqcap Q$ allows the future evolution of a process to be defined as a choice between two component processes, but does not allow the environment, any control over which one of the component processes will be selected.
4. $P \,{}_b||_c\, Q$ represents concurrent activity that requires synchronization between component processes– any

event in the interface set can only occur when all component processes are able to engage in that event.

5. $P \mid\mid\mid Q$ represents completely independent concurrent activity. The process behaves as both P and Q simultaneously. The events from both processes are arbitrarily interleaved in time.
6. $P \backslash a$ represents action a unobservable to process P.
7. $f[P]$ denotes the process derived by P by relabeling actions of P, according to function f. This is called relabeling
8. $f^{1}[P]$ is same as relabeling, but here relabeling function is f^{1}.
9. $x \rightarrow P_x$ denotes a process which can only perform action x and after that behave like P_x.

B.2. Semantics of CSP

As CCS, the operational semantics of CSP is defined by LST [5, 6]. Here set of actions A is defined as $A = E \cup \{\tau\}$, where E is the set of names and τ is silent or internal action (borrowed from CCS).

$$\frac{P \xrightarrow{x} P'}{P \square Q \xrightarrow{x} P'} \quad (9) \qquad \frac{Q \xrightarrow{x} Q'}{P \square Q \xrightarrow{x} Q'} \quad (10)$$

$$\frac{P' \xrightarrow{\tau} P}{P \square Q \xrightarrow{\tau} P'} \quad (11) \qquad \frac{Q \xrightarrow{\tau} Q'}{P \square Q \xrightarrow{\tau} Q'} \quad (12)$$

$$\frac{P \xrightarrow{\tau} P'}{P\, {}_b||_c\, Q \xrightarrow{\tau} P'\, {}_b||_c\, Q} \quad (13) \qquad \frac{Q \xrightarrow{\tau} Q'}{P\, {}_b||_c\, Q \xrightarrow{\tau} P\, {}_b||_c\, Q'} \quad (14)$$

$$\frac{P \xrightarrow{x} P'}{P\, {}_b||_c\, Q \xrightarrow{x} P'\, {}_b||_c\, Q} \quad (x \in b - c) \quad (15)$$

$$\frac{Q \xrightarrow{x} Q'}{P\, {}_b||_c\, Q \xrightarrow{x} P\, {}_b||_c\, Q'} \quad (x \in c - b) \quad (16)$$

$$\frac{P \xrightarrow{x} P' \;,\; Q \xrightarrow{x} Q'}{P\, {}_b||_c\, Q \xrightarrow{x} P'\, {}_b||_c\, Q'} \quad (x \in b \cap c) \quad (17)$$

$$\frac{P \xrightarrow{x} P'}{P \mid\mid\mid Q \xrightarrow{x} P \mid\mid\mid Q'} \quad (18) \qquad \frac{Q \xrightarrow{x} Q'}{P \mid\mid\mid Q \xrightarrow{x} P \mid\mid\mid Q'} \quad (19)$$

$$\frac{P \xrightarrow{x} P'}{P \backslash a \xrightarrow{x} P \backslash a} \quad (x \neq a) \quad (20) \qquad \frac{P \xrightarrow{x} P'}{f[P] \xrightarrow{f(x)} f[P']} \quad (21)$$

$$\frac{P \xrightarrow{x} P'}{f^{1}[P] \xrightarrow{y} f^{1}[P']} \quad (y = f^{1}(x)) \quad (22)$$

In CSP the equivalence theory is based on the trace theory [6]. In trace theory a process is considered to be fully determined by the possible sequences of atomic actions, which it can perform (its traces) [7, 8]. Therefore, a model is created in which a process is represented by a set of traces. Usually traces sets are required to be prefix closed and to contain only finite traces of infinite processes. In this setting any non-empty prefix closed set of finite words over A represents a process. Two processes are considered equivalent when they have same traces and this equivalence is called trace congruence. Although, both CCS and CSP provides there theory of equivalence based on observational equivalence and trace congruence but both don't agree with each other. Moreover in CCS, it is just like game theory to find out nonequivalence (dissimulation) but in CSP it is just matter of proving two sequences dissimilar [9].

C. Algebra of Communicating Processes (ACP)

Another process theory that should be mentioned is the metric approach by De Bakker and Zucker [10, 11]. There is a notion of distance between processes: processes that do not differ in behavior before the nth step have a distance of at most 2−n. This turns the domain of processes into a metric space, that can be completed, and solutions to guarded recursive equations exist by application of Banach's fixed point theorem. Jan Bergstra and Jan Willem Klop [12] started work on a question of De Bakker as to what can be said about solutions of unguarded recursive equations. As a result, they wrote the paper [12]. In this paper, the phrase "process algebra" was used for the first time. This paper establishes process algebra in the strict sense. In this, process algebra was defined with alternative, sequential and parallel composition, but without communication. A model was established based on projective sequences (a process is given by a sequence of approximations by finite terms) and in this model, it was established that all recursive equations have a solution. In adapted form, this paper was later published as [13]. In [14], this process algebra PA was extended with communication to yield the theory ACP (Algebra of Communicating Processes).

In contract of CCS and CSP, ACP does not provide any new model for concurrency but provides an axiomatic-algebraic approach to concurrency. There is a staggering amount of properties which one may or may not attribute to process, there are dozens of views (semantics) which one may have on (a particular kind of) processes and there are infinitely many models of process. So, ACP was an attempt to organize this field of process theories and to axiomate the methodology. Indeed, ACP provides a good motivation for developing new process algebra but here, full details on ACP is not provided because of two reasons. First one is that it only laid a frame work by the means of properties that any process algebra should possess; for example, notation of deadlock. Secondly it doesn't provide any new model for concurrency and communication as CCS and CSP does. For more details on ACP one can refer "Algebra of communicating process" by Bergstra [15].

III. Probabilistic Extension Of CCS

In previous section, two models for concurrency and communications named as CCS and CSP have been introduced. During the time, researchers show most of their interest in CCS in comparison to CSP and provide more useful extensions of CCS. Reason for their interest is "observational equivalence" on which CCS was based. To insulate this, let's consider a simple example. Suppose there are two processes named as P and Q respectively, making there computation in distributed environment. P has to take some numeric data from file F_1 and has to perform some mathematical computation, say as addition with the data of file F_2. Result will be stored in separate file F_3. Now, process P is defined in CCS as

$$P \overset{def}{=} \text{read}\,(F_1).\text{read}\,(F_2).\text{write}\,(F_3).P \qquad (23)$$

Process Q is also performing same task but sequence of reads has been altered.

$$Q \overset{def}{=} \text{read}\,(F_2).\text{read}\,(F_1).\text{write}\,(F_3).Q \qquad (24)$$

Now, one can easily observe that processes P and Q are equivalent and CCS, also deduces same result by providing $P \sim Q[F_1/F_2]$. Here ~ shows strong equivalence between P and Q.

Now, take a look on the sequence of actions made by P and Q. Those are < read (F_1), read (F_2), Write (F_3) > and < read (F_2), read (F_1), Write (F_3) > respectively. As, it has been pointed out earlier that equivalence in CSP is based on sequence of actions(traces); one can easily find out that traces for both P and Q are not similar and this concludes us that P and Q are not equivalent, if P and Q are modeled on the basis of CSP. But this is not the case; P and Q are performing same computations so they are equivalent from the point of view of the observer. So, CSP sometimes differs from observer's interventions. But, against this CCS derives equality with observer's interventions and this is the main reason of showing researchers interest in CCS in comparison to CSP.

One of the main extensions of CCS is providing stochastic choice operator in CCS [16]. This makes CCS capable for modeling systems that exhibits stochastic characteristics or based on distributed random algorithms; specially distributed security protocols. Formal abbreviation of this calculus is Probabilistic CCS [17].

A. Syntax of Probabilistic CCS

The probabilistic extension of CCS with a binary choice operator $\otimes_p$ indexed with probability $p \in]\,0,1\,[$ to model internal stochastic and non-deterministic processes. Intuitively process $P \otimes_p Q$, may move to P immediately with probability p, written as $P \otimes_p Q \xrightarrow{\varepsilon_p} P$ or to Q with a probability 1-p. The syntax of this CCS was formulated in a way that it can well mess with original CCS.

As in CCS, A is a set of names, ranged over by α, β, γ and a set A' of co-names, ranged over by α', β', γ'. A complete set of actions is described by $Act = A \cup A' \cup \{\tau\}$. Let, S be a bijective function from Act to Act with $S(\alpha') = S'(\alpha)$ and $S(\tau) = \tau$. It uses p to range over the open interval] 0, 1[and s, t to range over the left open interval] 0, 1]. The calculus has the following syntax:

1. The set P of (concurrent) processes expressions is defined by $P := X \mid \alpha.P \mid P1{+}P2 \mid P1|P2 \mid P \otimes_p Q \mid P[f] \mid nil \mid P\backslash L$ Where I is any finite indexing set and ranged over by i.

2. $\alpha.P$ denotes a process which can only perform action α and after that behave like P.

3. $P_1 + P_2$ is a process which behaves like P_1 or P_2.

4. $P_1 \mid P_2$ defines two concurrent processes.

5. $P \otimes_p Q$ is a process which can behave like P with probability p and Q with probability 1-p.

6. $P\,[f]$ and $f = [a_1 / b_1 \ldots a_n / b_n]$ denotes the process derived from P by replacing action $b_1, b_2, \ldots\ldots\ldots, b_n$ in P with $a_1, a_2, \ldots\ldots\ldots, a_n$.

7. *nil* shows no action.

8. $P\backslash L$ is a process which cannot perform actions in L.

B. Semantics of Probabilistic CCS

The semantics of this calculus is given by probabilistic transitions. They are described as follows

$$\frac{}{nil \xrightarrow{\varepsilon_1} nil} \quad (25) \qquad \frac{}{\alpha.P \xrightarrow{\varepsilon_1} \alpha.P} \quad (26)$$

$$\frac{}{P \otimes_p Q \xrightarrow{\varepsilon_p} P\ (P \neq Q)} \quad (27) \qquad \frac{}{P \otimes_p Q \xrightarrow{\varepsilon_{1-p}} Q\ (P \neq Q)} \quad (28)$$

$$\frac{}{P \otimes_p P \xrightarrow{\varepsilon_1} P} \quad (29)$$

$$\frac{P \xrightarrow{\varepsilon_p} P' \quad Q \xrightarrow{\varepsilon_s} Q'}{P + Q \xrightarrow{\varepsilon_{p.s}} P' + Q'} \quad (30) \qquad \frac{P \xrightarrow{\varepsilon_p} P' \quad Q \xrightarrow{\varepsilon_s} Q'}{P \mid Q \xrightarrow{\varepsilon_{p.s}} P' \mid Q'} \quad (31)$$

$$\frac{P \xrightarrow{\varepsilon_p} P'}{P\backslash L \xrightarrow{\varepsilon_p} P'\backslash L} \quad (32) \qquad \frac{P \xrightarrow{\varepsilon_p} P'}{P\,[f] \xrightarrow{\varepsilon_p} P'\,[f]} \quad (33)$$

The set of action rules are essentially from CCS with one main change. The action rules for summation "+" and the interleaving rules for parallel composition "|" have an extra premise $P \xrightarrow{\varepsilon_1} P'$ that is the stochastic-stable condition, which requires that each process must be stochastic-stable first before it is ready to perform a real action.

IV. CCS Extension For Reversible Process

Another main extension of CCS is RCCS [18] also known as "Calculus for Reversible Concurrent Systems". It was designed to model the backward computations of concurrent processes .This provides a formal framework for modeling various distributed security protocols such as "ElGamal algorithm Cryptographic protocol" [18] and various distributed multithreaded systems whose behavior are sometimes can exhibit stochastic both in terms of communication failures and random roll-back.

Backtracking means, rewinding one's computation steps. In distributed setting actions are taken by different sites of computation, and no concurrently running site retains the complete history of others actions. Therefore, it is not certain that, if a given site goes a step back in its local computation, this will corresponds to going back a step in the global computation. One could think that a site willing to go back a step, to first verify that it was last to take an action. But, then all concurrent behaviors will be lost. Other possibility is that, if one site freely backtracks then it could result in reaching those computation states that were formally inaccessible. So, one has to make compromise here.

CCS of distributed backtracking built on top of Milner's CCS [1]. At any time each process may either performing internal action or synchronizing with another process. In both cases whatever action was taken is recorded in a memory. When the process wants to rewind a computation step, it has to synchronize with either its sub-components, in the case of a internal action; or with its partner in the case of a former synchronization. Thus backtrack is also considered as a synchronization mechanism.

A. Syntax of RCCS

The concept behind implementing backtrack is to assign to each currently running process an individual memory stack keeping track of past communications. This memory will also serve as a naming scheme and yield a unique identifier for the process. Upon doing a forward transition, this information that is needed for a potential roll-back will be pushed on the memory stack. The syntax of backtrack is defined as Simple processes: a Simple process was taken from CCS. They have their usual meaning as in CCS but abbreviated as threads in RCCS [19, 20]

Actions:	$\alpha ::= a \mid a' \mid \ldots$	Action on a channel
	$\mid \tau$	Silent action
Processes:	$P ::= 0$	End of process
	$\mid \sum \alpha_i . P_i$	Choice
	$\mid (P \mid Q)$	Fork
	$\mid (a)\, P$	Restriction

Monitored processes: In RCCS, simple processes are not runnable as such, only monitored processes are. This second kind of process in RCCS are defined as follows:

Memories:	$m ::= < >$	Empty Memory
	$\mid < 1 > . m$	Left-Fork
	$\mid < 2 >. m$	Right-Fork
	$\mid < *, \alpha, P >. m$	Semi-Synch
	$\mid <m, \alpha, P >. m$	Synch

Monitored Processes:

$R ::= m \rhd P$	Threads
$\mid (R \mid R)$	Product
$\mid (a)R$	Restriction

In RCCS, monitored process can be uniquely constructed from a set of terms of the form $m \rhd P$, which is called thread. In a thread $m \rhd P$, m represents a memory carrying the information that this process will need in case it wants to backtrack. That memory is organized as a stack with the last action taken by the thread sitting on the top together with additional information. There is an evident prefix ordering between memories which will be denoted simply $\leq$. As an example, consider this monitored process:

$$R = <1> < *, a, 0 > \rhd b.\, c'.\, 0 \mid < 2 > < *, a, 0> \rhd c.0 \quad (34)$$

It consists of two threads, namely $S_1 = <1> < *, a, 0 > \rhd b.\, c'.\, 0$ and $S_2 = < 2 > < *, a, 0 > \rhd c.0$. Taking a closer look at S_1, one sees a fork action <1> sitting on top of its memory stack, indicating that the last interaction the thread took part in was a fork. Below one finds< *, *a*, 0 > indicating that the penultimate action was an *a* action exchanged with an unidentified partner *. That part of the past of S_1 is shared by S_2 as well. Actually, they both can be obtained from a same process $<> \rhd c.0\ a.\ (b.\ c'.\ 0 \mid c.0)$. It will become evident from next section, where the precise notions of computation have been formulated.

B. Semantics of RCCS

The semantics of RCCS is given by LTS (Labeled transition system). They are described as follows:

$$\text{Act}_1 \quad \frac{}{m \rhd \alpha.P + Q \xrightarrow{m:\alpha} < *, \alpha, Q >\cdot m \rhd \text{P}} \quad (35)$$

$$\text{Act}_2 \quad \frac{}{< *, \alpha, Q>\cdot m \rhd \text{P} \xrightarrow{m:\alpha} m \rhd \alpha.P + Q} \quad (36)$$

$$\text{Par}_1 \quad \frac{P \xrightarrow{m:\alpha} P'}{P \mid Q \xrightarrow{m:\alpha} P' \mid Q} \qquad \text{Par}_2 \quad \frac{Q \xrightarrow{m:\alpha} Q'}{P \mid Q \xrightarrow{m:\alpha} P \mid Q'} \quad (37)$$

$$\text{Rec} \quad \frac{P \xrightarrow{m:\alpha} P'}{(a)P \xrightarrow{m:\alpha} (a)P'} \quad \alpha \neq a \quad (38)$$

In RCCS two processes are said to be equivalent when one can reshape in the other by commuting successive concurrent actions or canceling successive inverse actions.

V. CONCLUSION

In this paper, various extensions of process algebra for concurrent and communicating systems have been introduced. The early work was centered on giving the model for concurrency, communication and equivalence. As a result process algebras, CCS and CSP evolved. In doing so, process algebra became an underlying theory of all parallel and distributed systems, extending formal language and automata theory with the central ingredient of interaction. CCS emphasized observational equivalence while CSP laid its equivalence theory on trace equality. Researchers found observational equivalence more obvious because it maps, human intervention of equality more closely than equivalence on trace. So, CCS became more dominant than CSP to model concurrency. Although, CCS had rich syntax and semantics but it lacked in algebraic taste and this results a new research, called ACP. ACP provided a framework by the means of properties, which any process algebra should possess. ACP provided a great motivation to researchers to enhance or to add new features in current process algebra. During the time, distributed systems become more prominent but reliability was an issue. This was not of because of difficulty in codification of such complex systems but due to their verification. Verification requires a formal framework to model systems and some times, available models do not able to fulfill the needs of computation environment. So, researches in the field of process algebra are start guided by the new trends in field of computational environment. As in above scenario, stochastic behavior of process gave arise Probabilistic-CCS and when software industries demanded a greater confidence on reliability of such complex systems, when they rollback; then CCS was extended to RCCS.

VI. Future Work

One can extend process algebra having capability of modeling stochastic processes which can backtrack. Backtracking may give arise different probability distribution as compared to forward computation. Formulation of this new distribution is one of the challenges. Secondly, providing syntax and semantics to such computation. On the basis of this one can define a programming language to model reversible process in probabilistic environment.

References

[1] Robin Milner, "Modelling Communication," in *Communication and* Concurrency, C.A.R. Hoare, Hertfordshire, International Series on Computer Science. Prentice Hall, Revised 2008, pp. 11-35.

[2] Ying Jin, "Formal Verification of Protocol Properties of Sequential Java Programs", International Computer Software and Applications Conference(COMPSAC2007) IEEE-2007.

[3] Abdallah, E.Ali, B.Jones Cliff, Sanders and W.Jeff, "Communicating Sequential Processes", The First 25 Years, Lecture Notes in Computer Science, Spinger, July-2004.

[4] A.W Roscoe, "Parallel Operators", in *The Theory and Practice of Concurrency*, C.A.R. Hoare, and Richard, Hertfordshire, Prentice Hall, Revised 1997, pp. 13-51.

[5] Abdallah, E.Ali, B.Jones Cliff, Sanders and W.Jeff "Communicating Sequential Processes: The First 25 Years", LNCS. Springer, 1995.

[6] G.Barrett. "Model checking in practice: The T9000 Virtual Channel Processor", Transactions on Software Engineering. IEEE, 1995.

[7] Volker Diekert and Yves Metivier. "Partial Commutation and Traces", Handbook of Formal Languages, Beyond Words, Berlin, Springer-Verlag, 1997.

[8] Volker Diekert, "Combinatorics on traces", LNCS 454, Springer, 1990.

[9] A.McNeile and E.Roubtsova, "Protocol Modelling Semantics for Embedded Systems", Industrial Embedded Systems, SIES. IEEE, 2009.

[10] W. de Bakker, J.-J. Ch. Meyer, E.R. Olderog and J.I. Zucker, "Transition systems, metric spaces, and ready sets in the semantics of uniform concurrency", Journal of Computer and System Sciences, 1988.

[11] J.W.de Bakker, J.-J.Ch.Meyer, E.R.Olderog and J.I. Zucker, "Transition systems, metric spaces, and ready sets in the semantics of uniform concurrency", Journal of Computer and System Sciences, Rev 2009.

[12] J.V.Tucker and J.I.Zucker, "Continuity of operators on continuous and discrete time streams", Theoretical Computer Science. Science Direct, 2011.

[13] J.A.Bergstra and C.A.Middelburg. "Process algebra semantics for hybrid systems", Technical Report CS-R 03/06, Technische Universiteit Eindhoven, Dept. of Comp. Sci., 2003 .

[14] J.C.M.Baeten, C.A.Middelburg, and M.A.Reniers. "A new equivalence for processes with timing", Technical Report CSR, Eindhoven University of Technology, Computer Science Department, 2002.

[15] J.A.Bergstra, A.Ponse and S.A.Smolka, editors. Handbook of Process Algebra. North Holland, Amsterdam, 2001.

[16] Ruggero Lanotte and Simone Tini, "Probabilistic Bisimulation as a Congruence", Transactions on Computational Logic, ACM, 2007 .

[17] Dr. Anoop Singhal and Dr. Xinming Ou. "Security Risk Analysis of Enterprise Networks: Techniques and Challenges", ACM Conference on Computer and Communications Security. ACM, 2009.

[18] Ed.Dawson, Clark Andrew, ColinBoyd. Lecture Notes in Computer Science, Vol. 1841.5th Australasian Conference, ACISP. Springer, 2000.

[19] A.Naborskyy and R.M Fujimoto. "Using Reversible Computation Techniques in a Parallel Optimistic Simulation of a Multi-Processor Computing System", Principles of Advanced and Distributed Simulation PADS, IEEE, 2007.

[20] Vincent Danosand, Jean Krivine. "Reversible Communicating Concurrent Systems", CNRS, INRIA Rocquencourt, 2010.