Design of a Biomolecular Device that Executes Process Algebra

Urmi Majumder and John H. Reif
Department of Computer Science,
Duke University, Durham, NC, USA.
Tel: 1-919-660-6568, Fax: 1-919-660-6519
{urmim, reif}@cs.duke.edu

Abstract: *Process algebras* are widely used for defining the formal semantics of *concurrent communicating processes*. This paper considers *stochastic π-calculus* which is a particularly expressive kind of process algebra providing a specification of probabilities of process behavior such as stochastic delays, communication and branching, as well as rates of execution. In this paper, we implement stochastic π-calculus at the *molecular* scale, providing a design for a *DNA-based biomolecular device* that executes the stochastic π-calculus. Designing this device is challenging due to the requirement that a specific pair of processes must be able to communicate repeatedly; this appears to rule out the use of many of the usual classes of DNA computation (e.g., tiling self-assembly or hybridization chain reactions) that allow computational rule molecules to float freely in solution within a test tube. Our design of the molecular stochastic π-calculus system makes use of a modified form of *Whiplash-PCR* (WPCR) machines. In our machine which we call π-*WPCR* machine, we connect (via a tethering DNA nanostructure) a number of DNA strands, each of which corresponds to a π-WPCR

machines. This collection of π-WPCR machines is used to execute distinct concurrent processes, each with its own distinct program. To implement process communication protocols, our modifications to the original design of WPCR machines include the incorporation of additional secondary structure in the single strand (*stem-loop*) as well as *multiple-temperature* thermal cycling. The enforced locality of the collection of π-WPCR machines insures that the *same pair* (or any subset of the entire collection) of processes be able to repeatedly communicate with each other. Additionally, our design of the devices include implementation of *sequential execution* of multiple process and limited *process branching* through use of *restriction enzymes*.

Keywords: autonomous molecular computation, finite state automata, DNA self-assembly, strand displacement, DNA polymerization, programmable molecular machines, polymerase chain reaction, autocatalytic biomolecular computers, state transition, π calculus, stochastic π calculus, process algebra, Concurrency, Distributed system

Abbreviations: PCR: Polymerase Chain Reaction; DNA: DeoxyriboNucleic Acid, ds-DNA: double stranded DeoxyriboNucleic Acid; WPCR: Whiplash Polymerase Chain Reaction; π-WPCR: π-Whiplash Polymerase Chain Reaction

1 Motivation

1.1 Process Algebra

Process algebra has been popularly used to design concurrent, distributed and mobile systems (Milner, 1999). Process algebra has also traditionally been seen as a paradigm for practical concurrent languages and as a specification language for software and hardware systems that are encoded in more pragmatic ways. The main

idea in process algebra is to model processes as *communicating systems* with decentralized control. In other words, in this model of computation, concurrent processes can be specified to execute distinct programs as well as communicate repeatedly with another process or set of processes. Process communication can be enforced to be *synchronous* via *handshake communication* protocols that require acknowledgment of message reception. Also, processes can replicate and generate new processes. This paper considers *stochastic π-calculus*, a particularly expressive form of process algebra developed by Cardelli (Cardelli, 2008) that provides a clear specification of probabilities of process behavior such as stochastic delays, branching, sequentialization of processes, communication as well as rates of execution.

Stochastic π-calculus has been shown to be particularly useful for modeling biological systems. Here, biological components are modeled as concurrent processes and their interaction is treated as process communication. A precise connection between process algebra and chemical reactions was established by Cardelli (Cardelli, 2008). These modeling techniques provide researchers with better models and simulations of living matter. Consequently, they provide a better understanding of how nature works and, sometimes, a tool to predict unknown behavior of living systems.

Biology already has several sophisticated example of inter-cellular communication. For instance, bacteria use such communication for gene regulation (Bassler, 1999). Yet another fascinating example is the interaction that takes place during the developmental transition of fertilization which initiates a rapid series of changes that restructures the egg into the zygote. There are several signaling agents that mediate several of these rapid modifications in cell structure. Studies indicate that elements from several of the key signaling pathways, co-localize on molecular scaffolds in the egg and provide a means for these pathways to interact (Koeneman and Capco, 2005).

1.2 The Need for a Molecular Process Algebraic System

The central question considered in this paper is how to implement process algebraic systems with biomolecules. This direction of research would allow us to use biological matter as flexible information and material processing devices. The most important feature of process algebra is that this model allows computation to proceed via interprocess communication. At the molecular scale, adding the capability to interact with each other will allow us to build far more complex and powerful molecular computing devices than those that have been proposed to date. In fact, communicating nano machines can spur the creation of entirely new applications such as nano-scale distributed computing systems or nano-scale sensing systems.

1.3 The Challenges of a Molecular Implementation of Process Algebra

One of the primary challenges of molecular process algebra is *local execution* of *distinct programs* meaning *parallel execution* of such programs in several machines *without interference*. For this, we need to design a biomolecular device, where multiple copies of the same or distinct devices can simultaneously compute without interfering with each other. These needs can be satisfied by a number of known biomolecular computing device designs including *tiling assemblies* (Winfree, 1998a), *Whiplash PCR machines* (Sakamoto et al., 1999) and *hybridization chain reactions* (Dirks and Pierce, 2004).

One considerably more challenging design requirement is the requirement for *handshaking communication* between two (or more) processes, where one of the interacting processes might send data to the other and wait to resume its own program execution until it receives acknowledgment from the other process. To implement synchronous process communication at the molecular scale, carrier molecules going from process A to process B are insufficient; rather process B must be able to send

an acknowledgment back to A that allows A to resume its execution.

1.4 Need for Locality in a Molecular Process Algebraic System

Designing a biomolecular device to execute process algebra is made particularly challenging due to the requirement that communication among the same processes might be repeated any number of times (for example, a pair of processes repeatedly communicate). This requirement appears to rule out the use of many of the usual classes of DNA computation (e.g., tiling self-assembly (Winfree, 1998a) or hybridization chain reactions (Dirks and Pierce, 2004)) that allow computational rule molecules to float freely in solution within a test tube, making it difficult to insure, for example, that the *exactly same* two processes communicate repeatedly. In other words, in these models, computation proceeds *globally* by the assembly of pre-programmed components. These components need to first *locate* each other before computation can proceed. Furthermore, communication in these computational models can only be implementing by the *emission* and *capture* of *carrier molecules*. Synchronous communication, on the other hand, requires that the processes be co-located in some way (we use DNA nanostructure tethering for this).

1.5 Implementation of a Molecular Process Algebraic System via Modified Whiplash PCR Machines

Whiplash PCR (WPCR) machines (Sakamoto et al., 1999) appear to be an ideal candidate for a molecular implementation of process algebra. Recall that WPCR machines are DNA devices composed of a single strand of DNA containing a distinct program (via segments of the DNA that encode computational rewrite rules) and a current state (via a short segment at the $3'$ end of the strand). Its computational steps are executed by repeated rounds of thermal-cycling that comprises of cooling (that facilitates hybridization of the $3'$ end with an interior segment that encodes

a computational rule), polymerase extension (to copy the new current state of the machine from the rewrite rule) and heating (to release the extended the $3'$ end encoding the new current state). Hence, these machines can hold both programs and their inputs in close proximity, allowing parallel computation. WPCR machines can also be connected (via a tethering DNA nanostructure) as shown in Figure 3(Left). This particular set up would allow the same two copies of process P_1 and process P_2 to communicate not only the first time but as many times needed thereafter. Consequently, a collection of WPCR machines can be used to execute distinct concurrent processes, each with their own distinct program that can communicate when required.

1.6 Engineering DNA for biomolecular computation

Before we discuss how DNA can be the building block of a process algebraic machine, we must introduce some of the primary methods by which DNA can be engineered and used in the correct execution of a WPCR machine. The three primary techniques are hybridization, strand displacement and polymerization.

DNA Hybridization is the process of combining two complementary, single-stranded DNA molecules into double-stranded molecule. DNA (and RNA) can bind to their complements under normal conditions or by when the strand mixture is cooled. Simple hybridization has been used to make a large variety of complex structures such as tiles and lattices (Mathieu et al., 2005; Shih et al., 2004; Goodman et al., 2004). The opposite of hybridization is *dehybridization*. The latter can be achieved by heating the solution mixture.

Polymerized Chain Reaction (PCR) is a common method for amplifying a target DNA molecule, which can be a single gene or merely part of it [Figure 1(Right)]. Through PCR, a small amount of DNA can be amplified several times and the richness and the fidelity of the product facilitates many applications, such as detecting hereditary diseases, identifying bacterial species, cloning genes, DNA computing and

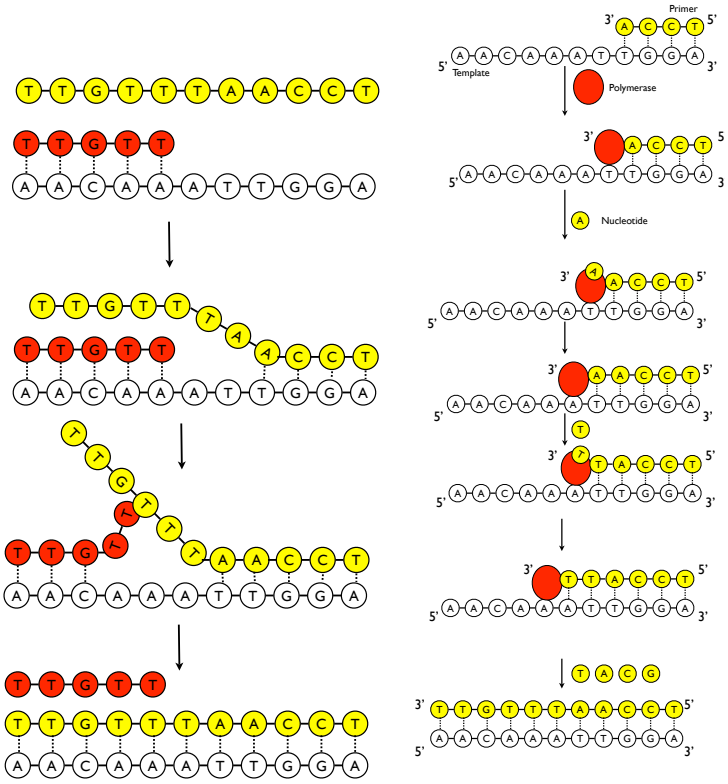

Figure 1: *(Left) Branch Migration, (Right)Extension of primer strand bound to the template by DNA polymerase.*

many others. A recent paper used DNA polymerase $\phi 29$ to power a nano-transport device (Sahu et al., 2008).

Branch migration or *strand displacement* is the process by which a single, invading DNA strand extends its partial pairing with its complementary strand as it displaces a resident strand from a DNA duplex [Figure 1(Left)].

Restriction is the process of recognizing specific bases in DNA sequences and cutting the DNA at that site by a specific type of enzyme from bacteria called restriction enzyme. In other words, such an enzyme acts as a pair of biochemical scissors.

1.7 Previous Work on Molecular Computers Capable of Executing Programs Locally

The concept of WPCR machines was introduced by Hagiya *et al.* (Sakamoto et al., 1999) in 1997. Winfree was the first to coin the name of *Whiplash PCR* machines for these biomolecular state transitioning systems and adopt it to a wider range of problems (Winfree, 1998b). Most of the work in the field of WPCR machines is devoted to addressing the problem of back-hybridization and, consequently, improving its success rate (Rose et al., 2001, 2006). Another shortcoming of the machine is that it is not isothermal and auto-catalytic which limits its range of applications. We addressed that issue in an earlier paper (Reif and Majumder, 2008) that provided an isothermal design, but this involved a rather complicated construction. Hence, in this paper, for simplicity, we adapt the design methodology of the original WPCR machines (Sakamoto et al., 1999) for the molecular implementation of process algebra. Moreover, we introduce additional capabilities in the original WPCR machine so that we can implement the basic primitives of process algebra at the nano-scale.

1.8 Biochemical Techniques used in the Molecular Implementation of Process Algebra

The various biochemical techniques that allow us to implement the various primitives of process algebra are *hybridization*, *polymerization* and *restriction* (Berg et al., 2002). Additionally, we modify the original WPCR machine to support the process communication construct. These modifications include incorporation of additional secondary structure such as a *stem-loop* in each computational rewrite rule as well as in the current state of the machine and use of a *multi-temperature thermal cycle* that starts in a very high temperature T_0 where all the DNA strands (each representing a process) are denatured, followed by a lower temperature T_1 ($T_0 > T_1$) that facilitates hybridization, polymerization and restriction and back to the higher temperature T_0.

Intermittently, the temperature is cooled to T_2 ($T_1 > T_2$) so that secondary structure such as stem-loop formation is facilitated. The stem loops allow for successful process communication (discussed in Section 6). The selective hybridization at T_1 and T_2 can be controlled through careful design of the secondary structure (e.g. hybridization length and sequence composition). In our design, polymerization uses a *non-strand-displacing polymerase* (such as Taq). The use of a non-strand-displacing enzyme is the key to a successful process communication. Restriction used for implementing replication and sequential composition utilizes two types of enzymes: *nicking enzymes* that creates only one nick in a double-stranded DNA (used to create an output strand from the first process and this in turn, can initiate the second process in sequential composition) and *restriction enzymes* that creates nicks in both the strands in a double stranded region (used to create child processes from a parent process).

1.9 Contributions

In this paper we present a molecular design of process algebra using design techniques similar to that of the original WPCR machines. We implement the basic primitives of process algebra in this new biomolecular device (which we call π-Whiplash PCR (π-WPCR)) machine including *stochastic delay operation, parallel and sequential composition, replication and process interaction.* We call the new machine *π-WPCR machine* since we implement the stochastic π-calculus version of process algebra at the molecular scale.

2 Process Algebra Overview

Till date several variants of process algebra have been proposed. The variant of process algebra that we use in this paper is called *stochastic π calculus* (Phillips and Cardelli, 2004; Milner, 1999; Phillips et al., 2004). *π calculus* can describe concurrent

processes whose *configuration may change during interaction.* Stochastic π-calculus is a type of π calculus where *stochastic rates are imposed on the processes.*

2.1 The Basic Operators in Process Algebra

The main constructs of stochastic π-calculus that are used are in this paper are: (1) *Parallel Composition,* (also known as *concurrency*), defined as two processes P and Q executing concurrently and denoted as $P|Q$, (2) *Process interactions* (also known as *communication*), defined as two processes interacting through *channels*, shared by the interacting processes. The possible process interactions π are (i) *delays* at a rate r, (ii) *input* on channel a at a rate r (denoted as $?a_{(r)}$) and (iii) *output* on channel a at a rate r (denoted as $!a_{(r)}$). *Stochastic delay operation* is the event of reordering a process at a rate r (denoted as $\tau_{(r)}$) while in *complementary synchronous interactions* processes can interact by performing complementary input and output on a common channel at a rate r (denoted by $a_{(r)}$), (3) *Sequential composition*, defined as ordering of processes and denoted by $a(x).P$. This means that P will wait for an input on channel a and P is only activated when data is received through a and substituted for identifier x, (4) *Summation*, defined as a choice between zero or more output $a < y >$ or input $a(x)$ actions that a process can perform and denoted as Σ, (5) *Nil process*, defined as a process whose execution is complete and has stopped and denoted as 0 and finally (6) *Replication*, defined as a process which can always create a new copy of P and denoted as $!P$. In other words, replication is a compact representation of the parallel composition of a countably infinite number of processes i.e. $!P = P|!P$.

Algebraic laws for the basic operators (such as parallel composition of processes, specifying which channel to use for sending and receiving data, sequentialization of interactions and process replication) allow process expressions to be manipulated. The most important rule, however, is the *reduction rule*: $x\langle y\rangle.P|x(v).Q \rightarrow P|Q_{y/v}$. This rule contains the computational essence of process algebra and can be expressed

solely in terms of parallel composition, sequential composition, input and output. The equation can be interpreted in the following manner: the process $x < y > .P$ sends a message y along channel x and then behaves like process P. Conversely, $x(v).Q$ is a process that waits for a value to be received through channel x, binding the variable v to the received value (in this case y), and then behaves like $Q\{y/v\}$, where $Q\{y/v\}$ indicates the substitution of the variable v by the value y in the body of Q. In other words, the process $x(v).Q$ receives this message on the same channel as $x\langle y\rangle.P$ sends its out. Once the message has been sent, $x < y > .P$ becomes P while $x(v).Q$ becomes $Q_{y/v}$.

We explain the operators discussed above through an example. Suppose we want to model a remote procedure call (RPC) between a server and a client using process algebra. Consider the following function, *Cube*, running on the server. *Cube* returns the integer which is cube of its argument, x: $int\ Cube(intx)\{return\ x^3;\}$. To represent the interaction using process algebra, we first model the *Cube* function in the server as follows: $!Cube(a,x).\overline{a}\langle x^3\rangle$. This expression can be interpreted in the following manner: Channel *Cube* accepts two inputs: the name of the channel a and the argument x for the *Cube* function. This argument is instantiated with an integer on a client call. After the call, the process will send back the result of calculating the cube of its argument, x, on the channel a. The ! operator at the beginning of the expression means that the server function can make multiple copies of itself on each client interaction.

We can model a client call to the *Cube* server (for instance, say we call *Cube* with an input of 3 as $y := Cube(3)$ in this manner: $(va)(\overline{Cube\langle a,3\rangle|a(y)})$. This expression essentially means that the client sends on the *Cube* both the channel a and the argument for the function 3 and simultaneously receives on the channel a the result y. Note that v represents the bind variable which guarantees that a private channel of communication is set up for each client interaction with the *Cube* server.

Putting the client and server processes together we have

$$!Cube(a,x).\overline{a}\langle x^3\rangle|(va)(\overline{Cube\langle a,3\rangle}|a(y))$$

.

2.2 Process Algebra as Interacting Automata

In this paper, we use the *interacting stochastic automata* version of stochastic π calculus. In other words, in this model of computation, each automata represents a process. These processes do not split up dynamically into more processes.[1] These automata can be either executing a stochastic delay transition or multiple automata can be interacting. For instance, an automaton in state A can move to state $A^{'}$ at a specified rate r by a spontaneous delay transition $\tau@r$. There can also be two other kinds of automata: the ones in state A can perform an input $?a$ on a channel a, and move to state $A^{'}$, provided that each can coordinate its transition with another automaton in state B that at the same time performs an output $!a$ on the same channel a, to move to state $B^{'}$. Each channel a has an associated rate r represented as $a@r$. We could even restrict our operations to only one kind of automata. For instance, it could be the case that an automaton in state A can choose to either perform an input $?a$ and move to $A^{'}$ or an output $!a$ and move to $A^{''}$. With two such automata, interaction is possible since one automaton can move to state $A^{'}$ while the other can move to state $A^{''}$ after the interaction.

[1] In this paper, we present a molecular design of "restricted" replication meaning a process cannot create new processes that are identical copies of the original one.

3 A Whiplash PCR Machine Representation of Process Algebra

3.1 Original Whiplash PCR machine

Before describing how to encode a process algebraic machine using WPCR, for completeness purposes, we will first describe how the original machine works. In the original WPCR machine, the transition table is encoded on a single stranded DNA, W as $S - a_1 - b_1 - S - a_2 - b_2 - \ldots - S - a_n - b_n$ where each pair $a_i - b_i$ represents the transition from state a_i to state b_i. Here and in the rest of the paper, any symbol s encodes for a DNA sequence and s^* encodes for its complementary sequence. The stopper sequence S isolates one state transition rule from another. The $3^{'}$ end of the same strand encodes the current state. For the description of the rest of the protocol, refer to Figure 4. We represent the stopper sequence S as a black square in the figure. Without loss of generality (w.l.o.g.), let us assume that the current state of the machine is a_i^*. Following the transition table, a_i can transition to b_i. Once a_i^* hybridizes with a_i (Figure 4: State S1) in W, polymerase extends the $3^{'}$ end of W to copy b_i (Figure 4: State S3). The polymerase halts after transcribing the bases complementary to b_i because of S which is generally implemented by emitting one of the bases in the solution. Using appropriate thermal cycling, W is then denatured. Consequently, it loses the hairpin structure (Figure 4: State S4). Once the mixture is cooled, the $3^{'}$ end of newly extended W (now bearing b_i^* as the current state) hybridizes with another section of itself which encodes the appropriate transition rule (in this rule $a_j = b_i$ is the current state and b_j is the next state) (Figure 4: State S5). Although input is not part of the description of the WPCR machine, it has been suggested that input be provided as part of the initial state and the encoding of the transition table updated to include inputs in the manner $S - a_i - I_i - b_i$ for the i^{th}

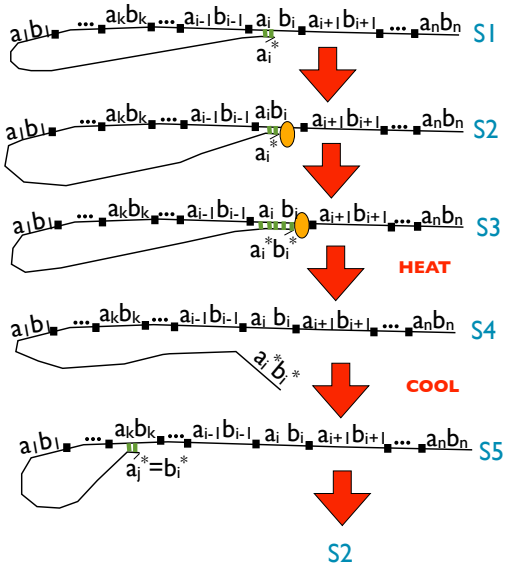

Figure 2: Schematic of the protocol for the original Whiplash PCR machine: (S1) Initial state of the WPCR strand W with current state being a_i^*. (S2) Polymerase binds to the $3^{'}$ end of W (bearing the current state). (S3) Next state b_i^* is copied at the head of W by primer extension. (S4) The mixture is heated so that W loses its hairpin structure. (S5) The solution is cooled so that the head of W can bind to the new current state $b_i^* = a_j^*$ encoded at the $3^{'}$ end of the strand and the whole state transition repeats again beginning with State S2.

transition rule.

3.2 Process Representation

The π-WPCR machine is encoded as $a_{11}a_{12}xa_{12}^*a_{13}b_1S \ldots a_{n1}a_{n2}xa_{n2}^*a_{n3}b_nS$ (in a single strand) where each rule R_i is encoded as $a_{i1}a_{i2}xa_{i2}^*a_{i3}b_i$ (with $a_{i1}a_{i2}xa_{i2}^*a_{i3}$ as the current state and b_i as the next state) and the current state of the machine encoded at the $3^{'}$ end of the strand as $a_{i1}^*a_{i2}^*x^*a_{i2}a_{i3}^*$ (let us assume that the current state of the machine corresponds to the current state of rule R_i). In the current state encoded as $a_{i1}a_{i2}xa_{i2}^*a_{i3}$ in each rule R_i, the most important segment is the encoding of a stem-loop $a_{i2}xa_{i2}^*$ (reason is explained in detail in Section 3.3). The encoding a_{i3} is used for information exchange in case of process interaction. Furthermore, the next state b_i is also the current state of a different rule R_j and is represented as $b_i = a_{j1}a_{j2}xa_{j2}^*a_{j3}$. We use a shorthand notation for clarity in the figures. A compact

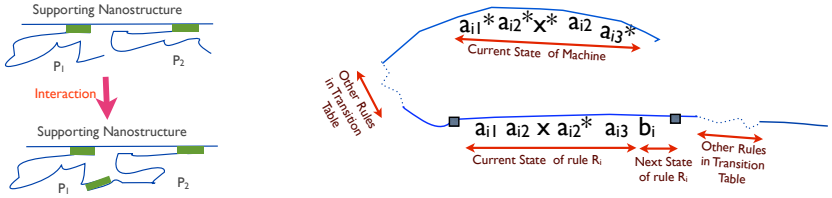

Figure 3: (Left)Supporting nanostructure holding two processes, represented by single stranded DNA, in close proximity (via hybridization) that facilitates interprocess communication, (Right) Shorthand representation of the modified WPCR strand simply showing its current state and the rule with which it is going to bind with next. The current state of the machine and rule R_i is indicated (with current state $a_{i1}a_{i2}xa^*_{i2}a_{i3}$ and next state b_i). The $a_{i2}xa^*_{i2}$ encoding in the current state of each R_i can form a stem-loop. A similar stem-loop encoding is hidden in b_i as well. It is not shown for clarity of representation. It is this loop that contributes to the loop at the $3^{'}$ end of the machine after a state transition. The stem-loop near the $3^{'}$ end of the machine is essential for process communication. The other rules are not shown for clarity and their presence is indicated by the dotted portion of the strand. Every adjacent pair of rules on the transition table are separated by a stopper sequence indicated by a dark square.

representation of a rule is shown in Figure 3.

3.3 π-WPCR Machine vs Original WPCR Machine

Unlike the original WPCR machine (Section 3.1), the π-WPCR machine encodes for *stem loops* ($a_{i2}x(a_{i2})^*$ in each R_i) both in the rules as well as in the $3^{'}$ end of the machine. Stem loops are crucial in the correct simulation of the process communication operator in process algebra. Hence to facilitate their formation, we use a *multi-temperature thermal cycling* in π-WPCR machine instead of the dual temperature thermal cycling protocol used in the original WPCR machine. In the former, the system is periodically cooled to a temperature T_2 below the melting temperature of the stem loops to facilitate process communication. However, this is done at a much lower frequency than the cooling to T_1 that facilitates all other operator simulations. With only a dual-temperature thermal cycle, stem-loop formation in the b_i (encoded

as $a_{j1}a_{j2}xa^*_{j2}a_{j3}$) region of each R_i, would prevent copying of the complete next state b_i from R_i (supposing R_i corresponds to the current transition of the machine). Moreover, recall that our protocol does not use a strand-displacing polymerase. Consequently, only the encoding just before the stem-loop would be copied. In other words, the stem loops are necessary for preventing the polymerase from going too far and copy undesired symbols at the $3^{'}$ end during process interaction (Section 6), thus affecting correct program execution in each machine thereafter. Hence, use of a multi-temperature thermal cycle is critical to the success of the protocol.[2]

4 Biochemical Simulation of Stochastic Delay Operation

A *stochastic delay operation* in π-WPCR machine is the same as the state transition event in the original WPCR machine and is simulated in exactly the same manner. However, its success rate is crucially dependent on the multi-temperature thermal cycle (Section 3.3). Refer to Figure 4 for pictorial reference on the complete process. In spite of the stem-loop encodings $a_{i2}xa^*_{i2}$ in each rule R_i and $a^*_{i2}x^*a_{i2}$ at the $3^{'}$ end of the strand, the loops are not preferentially formed when the system is cooled to T_1 above their melting temperature. Instead, the $3^{'}$ end (encoded as $a^*_{i1}a^*_{i2}x^*a_{i2}a^*_{i3}$) of the machine binds with the current state of R_i ($a_{i1}a_{i2}x(a_{i2})^*a_{i3}$). As usual, the polymerase can attach to the $3^{'}$ end of the machine and copy the next state from R_i. Next the strand is denatured by applying heat (meaning that the temperature of the solution is raised to T_0). The $3^{'}$ end of the π-WPCR strand has b^*_i encoded

[2]We cannot completely eliminate the formation of local stem loops at T_1 since it is a probabilistic event. Nevertheless, we can argue that polymerization (with non-strand-displacing polymerases, such as Taq, that can replicate a 1000 base pair strand of DNA in less than 10 seconds at $72^{\circ}C$ (Lawyer et al., 1993)) is a faster activity than hybridization (whose rate is proportional to the square root of the length of the segment to be hybridized (Hames and Higgins, 1995)) at high temperatures. Consequently, if a^*_{i3} at the $3^{'}$ end of the machine binds to a_{i3} in R_i, polymerase can copy the bases from b_i, before $a_{j2}xa^*_{j2}$ can form a loop in b_i. Furthermore, careful choice of length and sequence composition of the various segments of the rewrite rules can decrease the error rate in the multi-temperature thermal cycle.

in it. b_i^* is equivalent to $a_{j1}^* a_{j2}^* x^* a_{j2} a_{j3}^*$ which can now bind to the current state of rule R_j which is encoded as $a_{j1} a_{j2} x (a_{j2})^* a_{j3}$ and the chain of events described above repeats to execute another state transition. Such a state transition is referred to as a *stochastic delay operation* in interacting stochastic automata.

5 Biochemical Simulation of Parallel Composition of Processes

Parallel composition is the simplest of all operators to implement with π-WPCR machines. π-WPCR machines run in isolation by virtue of its design. Hence, simulation of parallel composition with π-WPCR machines, do not need any biochemical techniques other than the ones already being used. We have already mentioned that in WPCR machines both program and its input are internal to the machine, thus allowing parallel execution of distinct programs. Hence, it is possible for two processes P_1 and P_2 to have the same transition table but different current states and yet run in parallel without interfering with the smooth execution of the other process.

6 Biochemical Simulation of Process Communication

6.1 Overall Strategy

In this paper, the type of *process interaction or communication* we implement is *complementary synchronous interaction*. Suppose we have two π-WPCR machines that are executing state transitions independent of one another, until at one point one machine, say M_1 (corresponding to process P_1) cannot bind to any of the current state of the rules in its transition table. It can, however, resume its normal stochastic delay operation after it copies at the $3^{'}$ end an encoding that matches the current state of one of its rewrite rules from another machine (corresponding to process P_2)

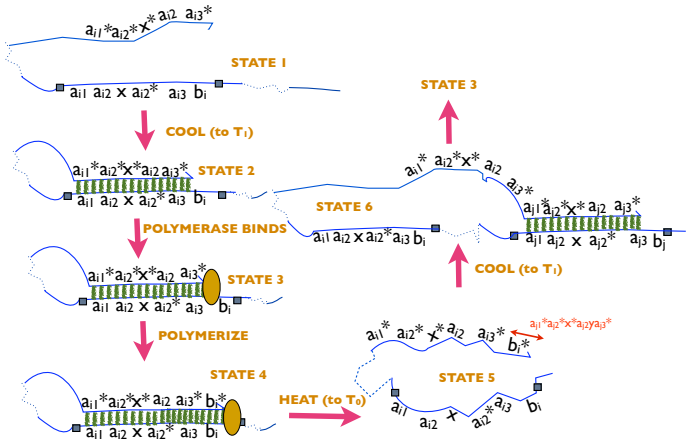

Figure 4: Stochastic Delay Operation with π-WPCR machines: (State 1) Original π-WPCR strand just after completion of the last state transition (system temperature T_0), (State 2) When cooled to T_1 below the melting temperature of the $3^{'}$ end of the machine but above the melting temperature of all the local stem loops in the rules and the stem-loop at the $3^{'}$ end, the former binds to the current state of rule R_i , (State 3) Polymerase binds to the $3^{'}$ end of the machine, (State 4) Polymerase extends the $3^{'}$ end of the machine to copy the next state in R_i (b_i), (State 5) Strand is denatured by the application of heat (system temperature raised to T_0), (State 9) The $3^{'}$ end of the machine now encodes $b_i^* = a_{j1}^* a_{j2}^* x^* a_{j2} a_{j3}^*$ and it can bind to the current state of R_j which is encoded as $a_{j1} a_{j2} x a_{j2}^* a_{j3}$.

M_2's $3^{'}$ end. Additionally, for complementary synchronous interaction, we have to implement M_2 as a machine where it has to provide the information that M_1 needed before it can proceed with its own state transitions.[3] Consequently, M_2 needs to wait for M_1 to finish receiving the desired information. As mentioned earlier, by the rules of communication in process algebra, M_2 not only modifies M_1 but is modified in the process as well. Hence, we implement M_2 waiting for M_1, by encoding in M_1's $3^{'}$ end, an encoding that matches the current state of a rule in M_2 and that is missing from the next state of all the rules in M_2. Thus, after a certain number of state transitions (need not be the same number as M_1's), M_2's encoding in its $3^{'}$ end does not correspond to any of the current states in its rules. On the contrary, it is the same encoding that M_1 has at its $3^{'}$ end when it needs to copy the current state encoding from M_2's $3^{'}$ end. Consequently, the machines cannot proceed without interacting

6.2 Complete Protocol

Suppose after several state transitions, the $3^{'}$ end of M_1 encodes $d_1c_1c_2$ (just before the last stem-loop from the $3^{'}$ end)(Figure 5(P1, P2: State 1)). The stem-loop is formed when the system is cooled to T_2 which is below the melting temperature of all the stem loops in the strand. Observe that this is lower than the temperature the system was being cooled for stochastic delay operations and simulations of other constructs. Though we cannot monitor each process individually, the frequency of cooling to the lower temperature T_2 is much lower than that to the higher temperature T_1 where all operations other than interprocess communication are favored. The reason for doing so is that one of the basic underlying assumption in this design is that process communication is a rarer event that stochastic delay operation. d_1 encodes for part of the current state in $R^{'}_m$ in P_2 since it is represented as $xa^{'*}_{m2}a^{'}_{m3}$. Similarly d_2 region

[3] In case of asynchronous interaction M_2 could have proceeded on its own or choose to wait until its current state encoded at its $3^{'}$ end is modified while communicating with M_1.

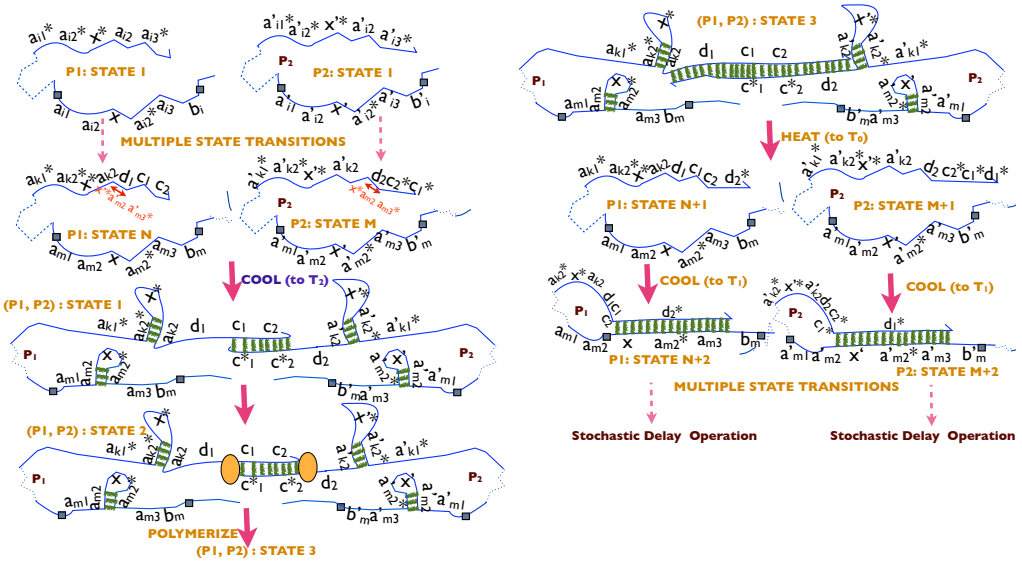

Figure 5: π-WPCR simulation of interprocess communication: (P1: State 1) Initial state of process P_1, (P2: State 1) Initial state of process P_2, (P1:State N) State of P_1 after N transitions. The $3^{'}$ end of P_1 encodes $d_1c_1c_2$. d_1 encodes for part of the current state in $(R_m)^{'}$ in P_2 since it is represented as $xa^{'*}_{m2}a^{'}_{m3}$. The dashed arrow between States P1:State 1 and P1:State N is used to denote multiple state transitions, (P2:State M) State of P_2 after M transitions. Its $3^{'}$ end encoded as $d_2c^*_2c^*_1$ Now d_2 encodes for part of the current state in R_m in P_1 represented as $xa^*_{m2}a_{m3}$, (P1,P2: State 1) The $3^{'}$ ends of P_1 and P_2 hybridize when cooled to T_2 below the melting temperature of not only c_1c_2 but also the local stem loops. Consequently, stem loops are formed near the $3^{'}$ ends of the machines as well as the rules. The lower temperature is indicated by representing this particular "COOL" operation in blue, (P1,P2: State 2) A polymerase attaches to the $3^{'}$ end in each of the machines, (P1,P2: State 3) In presence of the polymerase, the $3^{'}$ end of P_1 copies d_2 from P_2 and the $3^{'}$ end of P_2 copies d_1 from P_1, (P1: State N+1) Once heated to T_0, the interacting processes can separate and P_1's $3^{'}$ end d^*_2 now partly encodes for the current state of R_m, (P2: State M+1) Similar to P_1, after dehybridizing from the P_1P_2 complex, P_2's $3^{'}$ end encoded as d^*_1 can partly bind to the current state of $R^{'}_m$, (P1:State N+2) P_1's $3^{'}$ end binds to R_m as a first step to a stochastic delay operation, (P2:State M+2) P_2's $3^{'}$ end binds to $R^{'}_m$ as a first step to a stochastic delay operation.

in machine M_2's $3^{'}$ end (encoded as $d_2 c_2^* c_1^*$) corresponds to part of the current state in R_m in P_1 since it is represented as $x a_{m2}^* a_{m3}$. Thus each machine cannot proceed with its own state transition. The only way normal execution can be restored if the two machines interact and copy the required current state from the others $3^{'}$ end. This is possible since $c_1 c_2$ portion at the $3^{'}$ end of M_1 is complementary to $c_1^* c_2^*$ portion at the $3^{'}$ end of M_2 and the melting temperature of the $c_1 c_2$ duplex is higher than T_2. Once hybridized (Figure 5(P1,P2: State 1)), a polymerase attaches to the $3^{'}$ end for both the machines and it extends to copy the encoding until the first stem-loop on the strand (Figure 5(P1,P2: State 2)). Hence, the $3^{'}$ end of M_1 copies d_2 from M_2 and the $3^{'}$ end of M_2 copies d_1 from M_1 (Figure 5(P1,P2: State 3)). The solution can now be heated to T_0 to dehybridize the two machines and each being equipped with a $3^{'}$ end that corresponds to the current state in one of its rules (Figure 5(P1: State N+1) and Figure 5(P2: State M+1)), the machines can proceed with their respective stochastic delay operations, until a need for interprocess communication arises again (Figure 5(P1: State N+2) and Figure 5(P2: State M+2)).

The hybridization of P_1's and P_2's $3^{'}$ ends can be viewed as a *channel* and its rate of hybridization and subsequent copying of the next state as its *stochastic rate*. Our design of process communication simulates the *reduction rule* as well. Recall the reduction rule: $x\langle y\rangle.P | x(v).Q \rightarrow P | Q_{y/v}$. The expression essentially means the following: the process $x\langle y\rangle.P$ sends a message y along channel x. The process $x(v).Q$ receives this message on the same channel. Once the message has been sent, $x\langle y\rangle.P$ becomes P while $x(v).Q$ becomes $Q_{y/v}$. Note that v is the variable that is bound to the received value y over channel x. In our biochemical design of process communication, we implement reduction rule in the following manner: P_1 sends the message d_1 across the channel $c_1 c_2$; the process P_2 receives this message on the same channel. Once the message has been sent, P_1's state changes (from copying d_2) so that it can bind to one of its own rules and P_2's state changes (from copying d_1) such that it can proceed

with its own program execution as well.

6.3 Observations

One may wonder why we used two separate symbols c_1 and c_2 at the $3^{'}$ end of P_1 for describing interprocess communication when only one symbol may have sufficed. It is used merely for consistency reasons and is only used in the description of the summation protocol (Section 9) where we need to represent two distinct reaction pathways possible with the same $3^{'}$ end encoding.

A major issue with this simulation of process communication is that after the next states are copied from the $3^{'}$ end of the other process, in the next cooling cycle when the system is cooled to T_1 above the melting temperature of the local hairpins and the complementary segment at the $3^{'}$ ends of both the machines, the current states of the machines can still bind with each other. If that happens, in absence of any stem-loop at the $3^{'}$ ends of the machines, polymerase can copy the symbols encoded in those segments of the machine until the first stopper sequence and may completely jeopardize correct program execution in the subsequent thermal cycles. However, it is possible to engineer proximity of processes (in other words tie them to the same nanostructure) in a manner such that processes can still interact but internal interaction (interaction with itself) is preferred to external interaction (interaction with other processes). Now it is possible that one can control the relative concentration of processes such that some processes are more likely to interact than others. In fact, physical proximity can be induced by concentration. However, tethering processes which we intend to interact to a common nanostructure increases the likelihood of their interaction because the physical proximity of the processes induced by the shared nanostructure causes the concentration of the processes relative to each other to be much higher than would have been possible if they were free-floating in the solution, separate from each other.

Another important point to note here is that although any two processes should be capable of interacting (if they have matching channel names) , our biochemical design is currently limited to allowing only two processes on the same nanostructure to interact at a time meaning although multiple processes can be on the tethering nanostructure at any stage only a pair of processes have the same channel name, thus eliminating ambiguity.

7 Nil Process

One can implement a nil process by encoding a next state b_n encoded as $a_{k1}a_{k2}xa^*_{k2}a_{k3}$ in one of the rules R_k in a π-WPCR machine such that it does not have a corresponding current state in another rule from R_1 to R_n for the same machine ($k > n$). Thus the program execution halts once b_n is copied at the $3^{'}$ end of machine.

8 Sequential Composition of Processes

8.1 Overall Strategy

Sequential composition of processes in process algebra can be implemented by cascading WPCR machines (Matsuda and Yamamura, 2002), each corresponding to one of the participating processes. Cascading WPCR machines were first proposed by Matsuda *et. al.* It is a scheme to cascade results of WPCR from molecules to molecules by using a nicking enzyme.

The basic idea is as follows: Suppose we are composing only two processes P_1 and P_2 in sequence, then the π-WPCR machine M_1, corresponding to the first process P_1 executes as usual. However, the second machine M_2, corresponding to process P_2, cannot start until M_1 is finished. Essentially, M_2 does not have a $3^{'}$ end whose encoding matches with that of the current state in any of the rules in its transition

table. Now M_1 does not run forever. Eventually, after polymerase copies the next state b_j in a particular rule R_j, nicking enzyme E_N that recognizes a particular sequence a_{j3} in R_j nicks M_1 at the $3^{'}$ end to release the strand B ($B = za^{'}_{i1}a^{'}_{i2}xa^{'*}_{i2}a^{'}_{i3}$). Now B can bind partially with the $3^{'}$ end of M_2 using only z in the next cooling cycle (to T_1). The polymerase can then use B to copy $a^{'}_{i1}a^{'}_{i2}x^{'}a^{'*}_{i2}a^{'}_{i3}$ at the $3^{'}$ end of M. This corresponds to the current state of rule $(R_i)^{'}$ in M_2 and hence M_2 can start transitioning from one state to another. Refer to Figure 6 on details of the design.

8.2 Observations

It is important to note that since P_2 does not undergo any state transitions until receiving the current state encoding from P_1, its $3^{'}$ end encoding remains the same as the one at the beginning of computation for the whole test tube. In other words, at the time when P_2 is waiting for information from P_1, its $3^{'}$ end does not change dynamically from copying new symbols during stochastic delay operations. Consequently, it is possible to encode any appropriate symbol at its $3^{'}$ end and we encode z^* that allows it to receive data from P_1. This is essentially input prefixing where the input $a^{'}_{i1}a^{'}_{i2}xa^{'*}_{i2}a^{'}_{i3}$ is received over the channel z before P_2 can start operating.[4]

Another observation is that after the nick is created, in the next cooling cycle, B may prefer to bind to R_j in P_1 instead, because of stronger hybridization. However, it is free floating (not tied to any nanostructure). Consequently, there is a non-zero likelihood that it would collide with the correct region of P_2 since a copy of P_1 and a copy of P_2 share a supporting nanostructure. It may also happen that B hybridizes with the $3^{'}$ end of P_2 after the current state is copied at this end. This would prevent stochastic delay operations in P_2. However, as mentioned in Section 6.3 it is possible to engineer π-WPCR machines such that internal hybridization in a process is more

[4]Input prefixing (denoted as $a(x).P$) is a process waiting for a message that was sent on a communication channel named a before proceeding as P, binding the name received to the name x.

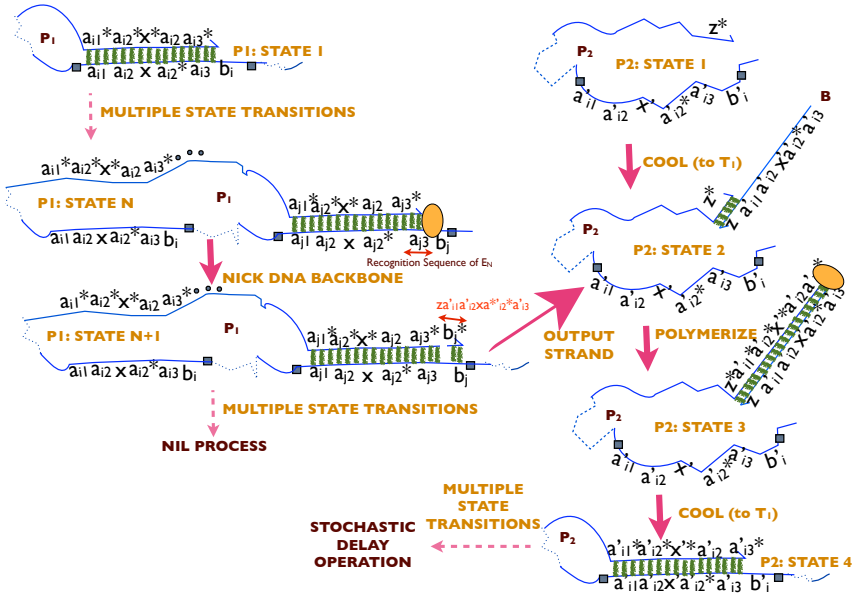

Figure 6: π-WPCR simulation of composing two processes in sequence: (P1: State 1) Initial state of process $P1$. Current state of the machine is bound to rule R_i, (P1: State N) State of $P1$ after N steps, where the current state of the machine is bound to R_j, (P1: State N+1) Nicking enzyme E_N uses the recognition sequence a_{j3} to create a nick between a^*_{j3} and b^*_j in the $3^{'}$ end of P_1. P_1 ultimately becomes a nil process when E_N nicks between a_{j3} and b_j in R_j, (P2: State 1) Initial state of process P_2. The encoding of its $3^{'}$ end is such that it cannot bind to the current state of any of the rules in its transition table until it receives the missing encoding from P_1, (P2: State 2) The small nicked strand B (encoded as b^*_j) floats from P_1 to P_2 and binds with the $3^{'}$ end of P_2 since B is encoded as $za^{'}_{i1}a^{'}_{i2}xa^{'*}_{i3}a^{'}_{i3}$, (P2: State 3) Polymerase binds at the $3^{'}$ end of P_2 and copies $a^{'}_{i1}a^{'}_{i2}xa^{'*}_{i3}a^{'}_{i3}$ from B, (P2: State 4) P_2's $3^{'}$ end binds with the current state of $R^{'}_i$.

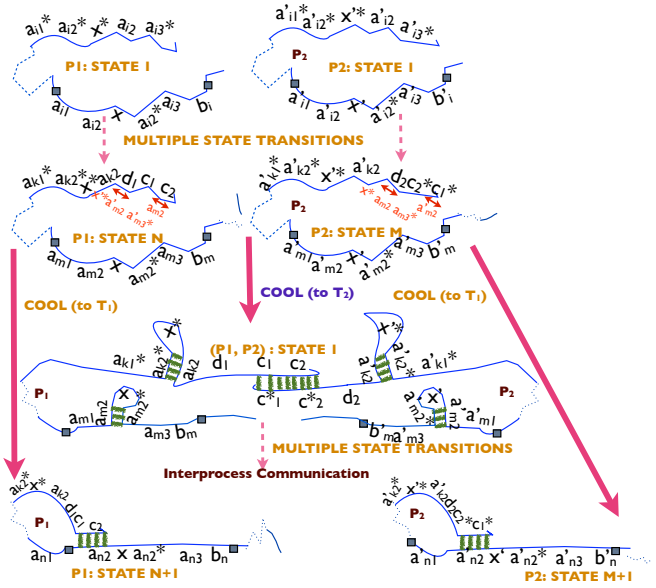

Figure 7: π-WPCR simulation of summation: (P1: State 1) Initial state of process P_1, (P2: State 1) Initial state of process P_2, (P1:State N) State of P_1 after N transitions. The $3^{'}$ end of P_1 encodes $d_1c_1c_2$. c_2 is complementary to a_{n2} part of the current state of rule R_n in P_1. However, it is only complementary to a_{n2} and cannot open the $a_{n2}xa^*_{n2}$ loop by strand displacement at the lower cooling temperature T_2. At higher system temperature T_1, in absence of the stem loops, the $3^{'}$ end of P_1 can partially bind to R_n and copy its next state and hence perform a stochastic delay operation. d_1 as in Section 6 encodes for the $xa^{'*}_{m2}a^{'}_{m3}$ of $R^{'}_m$ in P_2. This encoding allows P_1 to interact with P_2 as previously described (Section 6). (P2:State M) State of P_2 after M transitions. Its $3^{'}$ end is encoded as $d_2c^*_2c^*_1$ of which c^*_1 is complementary to $a^{'}_{n2}$ in $R^{'}_n$ encoded as $a^{'}_{n1}a^{'}_{n2}xa^{'*}_{n2}a^{'}_{n3}b^{'}_n$. However, the binding is not strong enough to facilitate strand displacement at lower cooling temperature T_2. Nevertheless at higher temperature T_1, this encoding allows the $3^{'}$ end of P_2 to partially bind to $R^{'}_n$ and copy its next state and hence perform a stochastic delay operation. On the other hand, similar to P_1, d_2 encodes for $xa^*_{m2}a_{m3}$ of R_m in P_1 and this encoding allows P_2 to communicate with P_1. (P1,P2: State 1) The $3^{'}$ ends of P_1 and P_2 hybridize when the system is cooled to T_2 below the melting temperature of the local stem loops. Next in presence of the polymerase, the $3^{'}$ end of P_1 copies d_2 from P_2 and the $3^{'}$ end of P_2 copies d_1 from P_1, thus performing a process communication operation, (P1: State N+1) P_1 can also choose to perform a stochastic delay operation by copying the next state of R_n. This path is favored at a higher system temperature T_1 above the melting temperature of the local stem loops because otherwise the matching encoding a_{n2} remains hidden in the loop, (P2:State M+1) P_2's $3^{'}$ end can similarly bind to $R^{'}_n$ as a first step to stochastic delay operation as opposed to choosing the interprocess communication path at a higher temperature T_1.

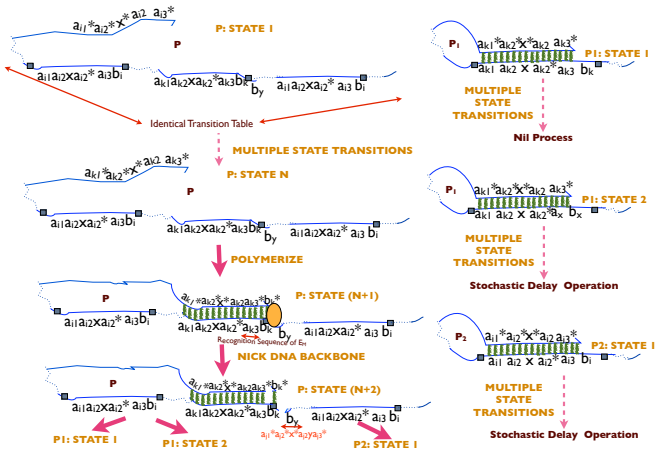

Figure 8: π-WPCR simulation of replicating a process in a restricted manner: (P: State 1) Initial state of process P that redundantly encodes two copies of the transition table and an additional rule R_k and a current state encoding b_y, encoded as $a^*_{i1}a^*_{i2}x^*a_{i2}a^*_{i3}$ between them, (P: State N) After N stochastic delay operations, P reaches a state where the $3^{'}$ end encoding matches the current state of rule R_k, (P:State N+1) Polymerase binds to the $3^{'}$ end of the machine bound to rule R_k and copies the next state b_k, (P:State N+2) Based on the recognition sequence a_{k3} in rule R_k, E_M creates a nick between a^*_{k3} and b^*_k in the machine's $3^{'}$ end as well as a nick after the stopper sequence in rule R_k. (P2:State 1) The new strand P_2 created by introduction of the nick after the stopper sequence in rule R_k contains not only the full transition table but also the current state b_y at the $3^{'}$ end, which can bind to rule R_j in its transition table, (P1:State 1) The remainder of P (now P_1), can bind to R_k that cannot be nicked since there are no DNA segments left after the stopper sequence. Consequently, for every thermal cycle neither any new process is generated nor the state of the machine changes. Hence P_1 becomes a nil process, (P1:State 2) Instead of binding to R_k, P_1 hybridizes with R_x where a_x (not a recognition sequence for E_M) is partially matched with a^*_{k3}. This allows P_1 to resume its normal stochastic delay operations.

favored that external hybridization. In other words, because of the locality of the rules and the current state in P_2, they are much more likely to bind than B hybridizing with the $3^{'}$ end of P_2.

If B does not bind with R_j, in the next cooling cycle P_1 binds to the current state of R_j and polymerase extends the $3^{'}$ end to copy b_j. As a result, the nicking enzyme E_N using the recognition sequence a_{j3} releases b_j^*. This is quite a favorable situation since B being a free floating strand, increase in its concentration implies that this strand is more likely to collide with the right complementary region of P_2. However, recall that the nicking enzyme cuts the top strand preferentially in a duplex region. Since the transition rules are bound to the supporting nanostructure, the $3^{'}$ end of the machine automatically plays the role of a top strand. Consequently, at some point of time, a nick is created in the rule R_j instead of P_1's $3^{'}$ end and with none of the rules matching the $3^{'}$ end of P_1, it turns into a nil process. Until that happens, P_1 continues to copy only b_j in every cooling cycle. Hence for all intended purposes, we assume that P_1's program execution has come to a halt.

9 Biochemical Simulation of Summation

If we combine interprocess communication with stochastic delay operations then it is an instance of summation operation. For instance, in the interprocess communication (Section 6), it may be possible that the encoding of the $3^{'}$ end of M_1 (say $d_1c_1c_2$ as in interprocess communication encoding), corresponding to process P_1 is such that it can continue to execute state transitions in an isolated fashion (meaning that it can bind to the current state of R_n in case of P_1 since c_2 encodes for a_{n2}). Similarly, P_2's $3^{'}$ end encoded as $d_2c_2^*c_1^*$ can bind to $R_n^{'}$ (since c_1^* encodes for $a_{n2}^{'}$). As before, P_1 and P_2 can also communicate via c_1c_2 hybridization of their respective $3^{'}$ ends and, consequently, modify themselves to encode part of their respective current states at

their respective $3^{'}$ ends ($d_2^* = x^* a_{m2} a_{m3}^*$ in R_m in P_1 and $d_1^* = x^{'*} a_{m2}^{'} a_{m3}^{'*}$ in $R_m^{'}$ in P_2) to continue program execution (bind to current state of R_m (for P_1) and $R_m^{'}$ (for P_2). These two possible pathways is an instance of summation operation. One observation that should be made here is that one pathway would be favored over the other depending on the system temperature. For instance, when the system is cooled above the melting temperature of the stem loops, the stochastic delay operation is favored while when it is cooled below the same, interprocess communication is favored. At the lower temperature T_2, part of the current state encoding of the desired rule tile is hidden in a stem-loop and the $3^{'}$ end fails to bind with it, hence favoring interprocess communication. Refer to Figure 7 for details on the simulation of summation with π-WPCR machines.

10 Restricted Replication

10.1 Overall Strategy

We implement a restricted version of replication: an operation that leads to not quite identical child processes. Although the replication construct may be hard to simulate with π-WPCR machine following the exact definition of the construct, one may imagine a situation where two copies of the transition table are encoded in the π-WPCR machine intercepted by a rule that has a recognition sequence for a restriction enzyme E_M and a current state, b_y for one of the child processes. The intermediate region has an additional lag region that would allow the child process, once created, to perform stochastic delay operations. When the π-WPCR machine copies the next state in the rule, the restriction enzyme E_M nicks the machine using this recognition sequence and when heated two machines are generated each with a copy of the transition table. Refer to Figure 8 on the details of the simulation of replication.

10.2 Issues

The molecular design described above is a restricted version of replication since the original process P is lost in generating processes P_1 and P_2. We can argue that P redundantly encoded two copies of the same transition table. Nevertheless, it should be remembered that after the restricted replication operation P_1 has an additional rule R_k. Hence neither P_1 or P_2 are identical to P nor they are identical to each other. Simulating the unrestricted version of replication with π-WPCR machines is still an open problem.

11 Summary

The design of π-WPCR machine described in this paper are based on the original WPCR machine and hence, particularly suffers from all its drawbacks. Furthermore, some of the constructs have their own limitations as well (as discussed in Sections 6, 8 and 10). Nevertheless, the design is quite powerful in the sense that it is easy to extend the design for two processes to that for multiple processes. For instance, extending parallel composition to multiple processes does not involve any modification in the encoding of the π-WPCR machines. Sequential composition can also extend to multiple processes if each currently executing process has a rule with a recognition sequence for the nicking enzyme to generate an output strand that initiates the next process. Extension of replication similarly involves having a rule in each replicating process possess a recognition sequence for the restriction enzyme. However, all interacting processes have to be physically close in order to communicate and hence should share a supporting nanostructure.

An implicit assumption in the simulations of all the constructs is that simulation of each operator is irreversible. This is because we assume that all the enzymatic reactions are irreversible. Consequently, even though each design involves reversible

hybridization/dehybridization reactions, each one involves enzymatic reactions as well and hence the overall simulation is an irreversible one. Another important observation that should be made here is that we do not monitor each individual process. We can only control the external factors such as thermal cycling. In terms of interacting processes, this implies that a process P_1 that needs input from another process P_2 before it can proceed with its own program needs to wait for P_2 to attain an appropriate state in which it can provide P_1 with its desired input. The delay in such process interaction is given by the maximum of the time taken by each process to attain the appropriate state in which they can communicate.

Another limitation of the current system is the necessity of physical proximity of communicating processes on a fixed substrate, which in turn, limits the scalability of the system as well as the switching between communication channels useful for modeling larger networks. Nevertheless, the probability of a single bio-operation such as communication or replication can be computed very easily by modeling it as a Markov Chain and using the known probabilities of primitives used in the process such as hybridization, branch migration and polymerization. The probability of iterated bio-operations can be computed from the single-step operation assuming each operation to be independent and identically distributed. However, it should be acknowledged that as each process goes through increasing rounds of stochastic delay operation, process communication and others, the strand extends by adding new bases to its end, thus increasing the thermodynamic stability of undesirable secondary structures which, in turn, may prevent further operations on the process. In other words, with each bio-operation the reliability of the process algebraic system decreases. It has already been noted that the current system is very temperature sensitive. For instance, the temperatures T_0, T_1 and T_2 used for thermal cycling have to be carefully chosen based on strand design of the processes such that only the intended processes can interact successfully. We admit that an isothermal system

would be more robust; however, the current biochemical design uses stem loops to determine how much information is exchanged in a communication step, for example, and the most essential component of an isothermal process algebraic system would be a strand displacing polymerase except that would polymerize the stem loops thus preventing repeated communication. Hence, with our current design we cannot use an isothermal process.

In summary, in this paper we presented a design for biochemical simulations of the key primitives in process algebra using modified WPCR machines. The main constructs in this model of computation are *sending data along a channel, receiving data along a channel, creating channels* and *running processes in parallel.* Since *input* and *program* are *local* to a WPCR machine, multiple processes can be executed *concurrently* in the original WPCR machines. Our simulation of interprocess communication using π-*WPCR machines* simulates rest of the constructs mentioned in Section 2.1. One challenging open question is whether we can use isothermal protocols since the former has evidently more flexibility of operations. The biggest hindrance to using an isothermal WPCR machine for implementing process algebra is that the isothermal protocol uses a strand displacing polymerase and this is not suitable for our simulation of process communication.

Acknowledgment

This work is supported by NSF EMT NANO grant CCF-0829798 and CCF-0523555.

References

Bassler, B. (1999). How bacteria talk to each other: regulation of gene expression by quorum sensing. *Curr Opin Microbiol 2*(582-587).

Berg, J., J. Tymoczko, and L. Stryer (2002). *Molecular Cell Biology.* W.H. Freeman.

Cardelli, L. (2008). On process rate semantics. *Theo. Comp. Sci. 391*(3), 190–215.

Dirks, R. M. and N. A. Pierce (2004, October). Triggered Amplification of Hybridization Chain Reaction. *PNAS 101*(43), 15275–15278.

Goodman, R., R. Berry, and A. Turberfield (2004). The single-step synthesis of a DNA tetrahedron. *Chemical Communications 12*, 1372–1373.

Hames, B. D. and S. J. Higgins (1995). *Gene Probes 2.* Oxford University Press.

Koeneman, A. and D. G. Capco (2005). *A Kaleidoscope of Modern Life Sciences and Modern Medicine Encyclopedia of Molecular Cell Biology and Molecular Medicine*, Volume 2. Larkspur, CA, USA: Ramtech Ltd.

Lawyer, F. C., S. Stoffel, R. K. Saiki, S. Y. Chang, P. A. Landre, R. D. A. RD, and D. H. Gelfand (1993, May). High-level expression, purification, and enzymatic characterization of full-length thermus aquaticus DNA polymerase and a truncated form deficient in 5' to 3' exonuclease activity. *PCR Methods Appl. 2*(4), 275–278.

Mathieu, F., S. Liao, J. Kopatscht, T. Wang, C. Mao, and N. Seeman (2005). Six-helix bundles designed from DNA. *Nano Lett. 5*, 661–665.

Matsuda, D. and M. Yamamura (2002). Cascading Whiplash PCR with a nicking enzyme. *DNA 8, LNCS 2568*, 38–46.

Milner, R. (1999). *Communicating and mobile systems: the pi calculus.* Cambridge Univ. Press.

Phillips, A. and L. Cardelli (2004). A correct abstract machine for the stochastic pi-calculus. In *Bioconcur.*

Phillips, A., N. Yoshida, and S. Eisenbach (2004). A distributed abstract machine for boxed ambient calculi. In *ESOP04, LNCS*, Springer-Verlag.

Reif, J. H. and U. Majumder (2008). Computing with Isothermal Autocatalytic Whiplash PCR. *DNA 14, LNCS*.

Rose, J., K. Komiya, S. Yaegashi, and M. Hagiya (2006). Displacement whiplash PCR: Optimized architecture and experimental validation. *DNA 12, LNCS 4287*, 393–403.

Rose, J. A., R. J. Deaton, M. Hagiya, and A. Suyama (2001). Pna-mediated Whiplash PCR. *Lecture Notes In Computer Science 2340*, 104 – 116.

Sahu, S., T. LaBean, and J. H. Reif (2008, October). A DNA Nanotransport Device Powered by Polymerase phi 29. *Nano Letters 8*(11), 3870–3878.

Sakamoto, K., D. Kiga, K. Komiya, H. Gouzu, S. Yokoyama, S. Ikeda, H. Sugiyama, and M. Hagiya (1999). State transitions by molecules. *Biosystems 52*(1), 81–91(11).

Shih, W., J. Quispe, and G. Joyce (2004). A 1.7-kilobase single-stranded DNA that folds into a nanoscale octahedron. *Nature 427*, 618–621.

Winfree, E. (1998a). Simulation of computing by self-assembly. Technical Report 1998.22, Caltech.

Winfree, E. (1998b). Whiplash PCR for O(1) Computing. Technical report, Caltech.