Strand Algebras for DNA Computing

Luca Cardelli

Microsoft Research, Cambridge UK

Abstract. We present a process algebra for DNA computing, discussing compilation of other formal systems into the algebra, and compilation of the algebra into DNA structures.

KEYWORDS: Process Algebra, DNA Computing.

1 Introduction

DNA technology is reaching the point where one can envision automatically compiling high-level formalisms to DNA computational structures [25]. Examples so far include impressive demonstrations of 'manual compilation' of automata and Boolean networks [2,12,20,21]. In addition to such sequential and functional computations, realized by massive numbers of molecules, we should also strive to take direct advantage of the massive concurrency available at the molecular level. To that end it should be useful to consider *concurrent* high-level formalism, in addition to sequential ones. In this paper we describe three compilation processes for concurrent languages. First, we compile a low-level combinatorial algebra to a certain class of composable DNA structures [23]: this is intended to be a direct (but not quite trivial) mapping, which provides an algebraic notation for writing concurrent molecular programs. Second, we compile a higher-level expression-based algebra to the lower-level combinatorial algebra, as a paradigm for compiling expressions of arbitrary complexity to 'assembly language' DNA combinators.

Third is our original motivation: translating concurrent interacting automata [5] to molecular structures. How to do that was until recently unclear, because one must decompose concurrent communication patterns into a form suitable for molecular interactions (a quadratic process that is described in [5]), and then one must find some suitable 'general programmable matter' as a physical substrate. Fortunately there has been recent progress on the latter [23], so that we can provide a solution to this problem in Section 6.2 based on the combinatorial DNA algebra. The general issue is how to realize the *external choice* primitive of interacting automata (also present in most process algebras and operating systems), for which there is currently no direct DNA implementation. In DNA we can instead implement a *join* primitive, based on [23]: this is a powerful operator, widely studied in concurrency theory [11,18], which can indirectly provide an implementation of external choice. The DNA algebra supporting the translation is built around such a join operator.

We begin with an introduction to process algebras, which are formal languages designed to describe and analyze the concurrent activities of multiple processes. The standard technical presentation of process algebras was initially inspired by a chemical metaphor [3], and it is therefore natural, as a tutorial, to see how the chemistry of diluted well-mixed solutions can itself be presented as a process algebra. Having chemistry in this form also facilitates relating it to other process algebras.

Take a set C of chemical solutions denoted by P,Q,R. We define two binary relations on this set. The first relation, *mixing*, $P \equiv Q$ is an equivalence relation: its

purpose is to describe reversible events that amount to 'chemical mixing'; that is, to bringing components close to each other (syntactically) so that they can conveniently react by the second relation. Its basic algebraic laws are the commutative monoid laws of + and 0, where + is the chemical combination symbol and 0 represents the empty solution. The second relation, *reaction*, $P \to Q$, describes how a (sub-)solution P becomes a different solution Q. A reaction $P \to Q$ operates under a dilution assumption; namely, that adding some R to P does not make it then impossible for P to become Q (although R may enable additional reactions that overall quantitatively repress $P \to Q$ by interfering with P). The two relations of mixing and reaction, are connected by a rule that says that the solution is well mixed: for any reaction to happen it is sufficient to mix the solution so that the reagents can interact.

In first instance, the reaction relation does not have chemical rates. However, from the initial solution, from the rates of the base reactions, and from the relation $\to$ describing whole-system transitions, one can generate a continuous time Markov chain representing the kinetics of the system. In terms of system evolution, it is also useful to consider the symmetric and transitive closure, $\to^*$, representing sequences of reactions.

As a process algebra, chemistry therefore obeys the following general laws:

Definition 1. *Chemistry as a Process Algebra*

$P \equiv P; \quad P \equiv Q \;\Rightarrow\; Q \equiv P; \quad P \equiv Q, Q \equiv R \;\Rightarrow\; P \equiv R$	*equivalence*
$P \equiv Q \;\Rightarrow\; P + R \equiv Q + R$	*congruence*
$P + Q \equiv Q + P; \quad P + (Q + R) \equiv (P + Q) + R; \quad P + 0 \equiv P$	*diffusion*
$P \to Q \;\Rightarrow\; P + R \to Q + R$	*dilution*
$P \equiv P', P' \to Q', Q' \equiv Q \;\Rightarrow\; P \to Q$	*well mixing*

In addition to these general rules, any given chemical system has a specific set of reaction rules. For example, consider a chemical process algebra with species: H, O, OH, H_2, H_2O. The set of solutions is given by those basic species, plus the empty solution 0 and any solution $P + Q$ obtained by combining two solutions. The mixing relation is exactly the one above. The reaction relation is given, for example, by the following specific reactions, plus dilution and well-mixing: $H+H \to H_2$; $H+O \to OH$; $H_2 + O \to H_2O$; $H + OH \to H_2O$. The mixing and reaction relations are defined inductively; that is, we consider the smallest binary relations that satisfy all the given rules. We can then deduce, for example, that $H + O + H \to\to H_2O$, that is we can produce water molecules in two steps (and by two different paths), and that $H + H + H + H + O + O \to^* H_2O + H_2O$. Chemical evolution is therefore encoded in the two relations of mixing and reaction: a solution P can evolve to a solution Q iff $\langle P, Q \rangle \in \to^*$.

Algebra is about equations, but in process algebra equations are usually a derived concept. Instead of axiomatizing a set of equations, we can use the reaction relation to study the equations that hold in a given algebra, meaning that $P = Q$ holds if P and Q produce the same reactions [15]. The complexity of these derived equational theories varies with the algebra. A simple instance here is the equation $P + 0 = P$, whose validity requires verifying that in our definition of $\to$ there is no reaction for 0, nor for 0 combined with something else.

This way, chemistry can be presented as a process algebra. But the algebra of chemical '+' is one among many: there are other process algebras that can suit biochemistry more directly [9,19] or, as in this paper, that can suit DNA computing.

2 Strand Algebras

By a *strand algebra* we mean a process algebra [15] where the main components represent DNA strands, DNA gates, and their interactions. We begin with a non-deterministic algebra, and we discuss a stochastic variant in Section 5. Our strand algebras may look very similar to either chemical reactions, or Petri nets, or multiset-rewriting systems. The difference here is that the equivalent of, respectively, reactions, transitions, and rewrites, do not live *outside* the system, but rather are part of the system itself and are *consumed* by their own activity, reflecting their DNA implementation. A process algebra formulation is particularly appropriate for such an internal representation of active elements.

2.1 The Combinatorial Strand Algebra, $\mathcal{P}$

Our basic strand algebra has some atomic elements (*signals* and *gates*), and only two combinators: *parallel (concurrent) composition* $P \mid Q$, and *populations* P^*. An inexhaustible population P^* has the property that $P^* = P \mid P^*$; that is, there is always one more P that can be taken from the population. The set $\mathcal{P}$ is formally the set of finite trees P generated by the syntax below; we freely use parentheses when representing these trees linearly as strings. Up to the algebraic equations described below, each P is a multiset, i.e., a solution. The *signals* x,y,... are taken from a countable set.

Definition 2. *Syntax of Combinatorial Strand Algebra*

$$P \ ::= \ x \mid [x_1,..,x_n].[y_1,..,y_m] \mid 0 \mid P_1|P_2 \mid P^* \qquad n \geq 1, m \geq 0$$

A *gate* is an operator from signals to signals: $[x_1,..,x_n].[y_1,..,y_m]$ is a gate that binds signals $x_1,..,x_n$, produces signals $y_1,..,y_m$, and is consumed in the process. We say that this gate *joins* n signals and then *forks* m signals; see some special cases below. An inert component is indicated by 0. Signals and gates can be combined into a 'soup' by parallel composition $P_1 \mid P_2$ (a commutative and associative operator, similar to chemical '+'), and can also be assembled into inexhaustible populations, P^*. Square brackets are omitted for single inputs or outputs.

Definition 3. *Explanation of the Syntax and Abbreviations*

x		*signal*	0	*inert*
$x_1.x_2$	$\triangleq [x_1].[x_2]$	*transducer gate*	$P_1 \mid P_2$	*composition*
$x.[x_1,..,x_m]$	$\triangleq [x].[x_1,..,x_m]$	*fork gate*	P^*	*population*
$[x_1,..,x_n].x$	$\triangleq [x_1,..,x_n].[x]$	*join gate*		

The relation $\equiv \subseteq \mathcal{P} \times \mathcal{P}$, called mixing, is the smallest relation satisfying the following properties; it is a substitutive equivalence relation axiomatizing a well-mixed solution [3]:

Definition 4. *Mixing*

$P \equiv P$	*equivalence*	$P \equiv Q \ \Rightarrow \ P \mid R \equiv Q \mid R$	*congruence*
$P \equiv Q \ \Rightarrow \ Q \equiv P$		$P \equiv Q \ \Rightarrow \ P^* \equiv Q^*$	
$P \equiv Q, Q \equiv R \ \Rightarrow \ P \equiv R$			
		$P^* \equiv P^* \mid P$	*population*
$P \mid 0 \equiv P$	*diffusion*	$0^* \equiv 0$	
$P \mid Q \equiv Q \mid P$		$(P \mid Q)^* \equiv P^* \mid Q^*$	
$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$		$P^{**} \equiv P^*$	

The relation $\rightarrow \subseteq \mathcal{P} \times \mathcal{P}$, called *reaction*, is the smallest relations satisfying the following properties. In addition, $\rightarrow^*$, *reaction sequence*, is the symmetric and transitive closure of $\rightarrow$.

Definition 5. *Reaction*

$x_1 \mid .. \mid x_n \mid [x_1, .., x_n].[y_1, .., y_m] \rightarrow y_1 \mid .. \mid y_m$	*gate* ($n \geq 1$, $m \geq 0$)
$P \rightarrow Q \;\Rightarrow\; P \mid R \rightarrow Q \mid R$	*dilution*
$P \equiv P', P' \rightarrow Q', Q' \equiv Q \;\Rightarrow\; P \rightarrow Q$	*well mixing*

The first reaction (gate) forms the core of the semantics: the other rules allow reactions to happen in context. Note that the special case of the gate rule for $m = 0$ is $x_1 \mid .. \mid x_n \mid [x_1, .., x_n].[] \rightarrow 0$. And, in particular, $x.[]$ annihilates an x signal. We can choose any association of operators in the formal gate rule: because of the associativity of parallel composition under $\equiv$ the exact choice is not important. Since $\rightarrow$ is a relation, reactions are in general nondeterministic. Some examples are:

$x_1 \mid x_1.x_2 \rightarrow x_2$
$x_1 \mid x_1.x_2 \mid x_2.x_3 \rightarrow^* x_3$
$x_1 \mid x_2 \mid [x_1, x_2].x_3 \rightarrow x_3$
$x_1 \mid x_1.x_2 \mid x_1.x_3 \rightarrow x_2 \mid x_1.x_3$ and $\rightarrow x_3 \mid x_1.x_2$
$y \mid ([y, x_1].[x_2, y])^*$ a catalytic system ready to transform multiple x_1 to x_2 , with catalyst y

Note that signals can interact with gates but signals cannot interact with signals, nor gates with gates. As we shall see, in the DNA implementation the input part of a gate is the Watson-Crick dual of the corresponding signal strand, so that the inputs are always 'negative' and the outputs are always 'positive'. This Watson-Crick duality need not be exposed in the syntax: it is implicit in the separation between signals and gates, so we use the same x_1 both for the 'positive' signal strand and for the complementary 'negative' gate input in a reaction like $x_1 \mid x_1.x_2 \rightarrow x_2$.

3 DNA Semantics

In this section we provide a DNA implementation of the combinatorial strand algebra. Given a representation of signals and gates, it is then a simple matter to represent any strand algebra expression as a DNA system, since 0, $P_1 \mid P_2$ and P^* are assemblies of signals and gates.

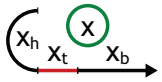

Fig. 1. Signal Strand

There are many ways of representing signals and gates as DNA structures. First one must choose an overall architecture, which is largely dictated by a representation of signals, and then one must implement the gates, which can take many forms with various qualitative and quantitative trade-offs. We follow the general principles of [23], where DNA computation is based on strand displacement on loop-free structures. Other architectures are possible, like computation with hairpins [25], but have

not been fully worked out. The four-domain signal structure in [23] yields a full implementation of the combinatorial strand algebra (not shown here, but largely implied by that paper). Here we use a novel, simpler, signal structure.

We represent a signal x as a DNA *signal strand* with three domains x_h,x_t,x_b (Figure 1): x_h = *history*, x_t = *toehold*, x_b = *binding*. A toehold is a domain that can reversibly interact with a gate: the interaction can then propagate to the adjacent binding domain. The history is accumulated during previous interactions (it might even be hybridized) and is not part of signal identity. That is, x denotes the equivalence class of signal strands with any history, and a gate is a structure that operates uniformly on such equivalence classes. We generally use arbitrary letters to indicate DNA *domains* (which are single-stranded sequences of bases).

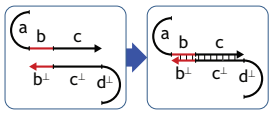

Fig. 2. Hybridization

A strand like b, c, d has a *Watson-Crick complement* $(b, c, d)^\perp = d^\perp, c^\perp, b^\perp$ that, as in Figure 2, can partially hybridize with a, b, c along the complementary domains. For two signals x,y, if $x \neq y$ then neither x and y nor x and $y^\perp$ are supposed to hybridize, and this is ensured by appropriate DNA coding of the domains [13,14]. We assume that all signals are made of 'positive' strands, with 'negative' strands occurring only in gates, and in particular in their input domains; this separation enables the use of 3-letter codes, which helps design independent sequences [14,28].

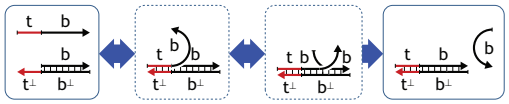

Fig. 3. Strand Displacement

The basic computational step of *strand displacement* [23] is shown in Figure 3 for matching single and double strands. This reaction starts with the reversible hybridization of the toehold t with the complementary $t^\perp$ of a structure that is otherwise double-stranded. The hybridization can then extend to the binding domain b by a neutral series of reactions between base pairs (*branch migration* [26]) each going randomly left or right through small energy hills, and eventually ejecting the b strand when the branch migration randomly reaches the right end. The free b strand can in principle reattach to the double-stranded structure, but it has no toehold to do so easily, so the last step is considered irreversible. The simple-minded interpretation of strand displacement is then that the strand t, b is removed, and the strand b is released irreversibly. The double-stranded structure is consumed during this process, leaving an inert residual (defined as one containing no single-stranded toeholds). Figure 4 shows the same structure, but seen as a gate G absorbing a signal x and producing nothing (0). The annotation 'x_h generic' means that the gate works for all input histories x_h, as it should.

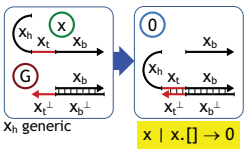

Fig. 4. Annihilator

Signal toeholds do not have to be distinct for different signals (with some caveats discussed below), because what is important to signal recognition is the successful branch migration over the binding domains: any binding mismatch there reverts by random walk until the toeholds unbind, and any mistaken toehold match has no consequence. Moreover the toeholds have to be short to guarantee reversibility; hence, their number is limited, and it is necessary in general to identify some of them. The binding regions however have no limitation, in principle, and hence can provide a very large address space for distinct signals.

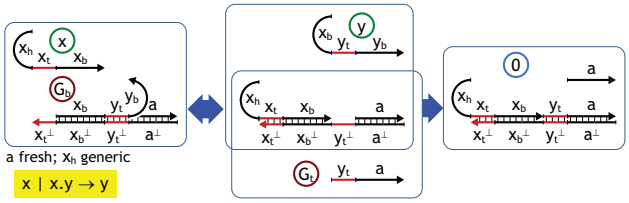

Fig. 5. Transducer

In Figure 5 we implement a gate $x.y$ that transduces a signal x into a signal y. The gate is made of two separate structures G_b (gate backbone) and G_t (gate trigger). The forward G_b reaction can cause y to detach because the binding of a toehold (y_t) is reversible. That whole G_b reaction is reversible via strand displacement from right to left, but the G_t reaction eventually 'locks' the gate in the state where x is consumed and y is produced. The annotation 'a fresh' means that the domain a is not shared by any other gate in the system to prevent interference (while of course the gate is implemented as a population of identical copies that share that domain). In general, we take all gate domains to be fresh unless they are non-history domains of input or output signals. Abstractly, an x to y transduction is seen as a single step but the implementation of $x.y$ takes at least two steps, and hence has a different kinetics than a single-step process. This is a common issue in DNA encodings, but its impact can be minimized [23], e.g. in this case by using a large G_t population.

In Figure 6 (*cf.* Figure 2 in [23]), we generalize the transducer to a 2-way fork gate, $x.[y, z]$, producing two output signals; this can be extended to n-way fork, via longer trigger strands.

Many designs have been investigated for join gates [8]. The solution shown in Figure 7 admits the coexistence of joins with the same inputs, $[x, y].z \mid [x, y].z'$, without disruptive crosstalk or preprocessing of the system (not all join gates have this property). It is crucial for join to fire when both its inputs are available, but not to absorb a first input while waiting for the second input, because the second input may never come, and the first input may be needed by another gate (e.g., another join with a third input). The solution is to reversibly bind the first input, taking advantage

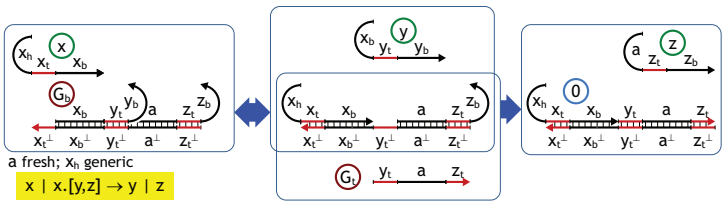

Fig. 6. 2-way Fork

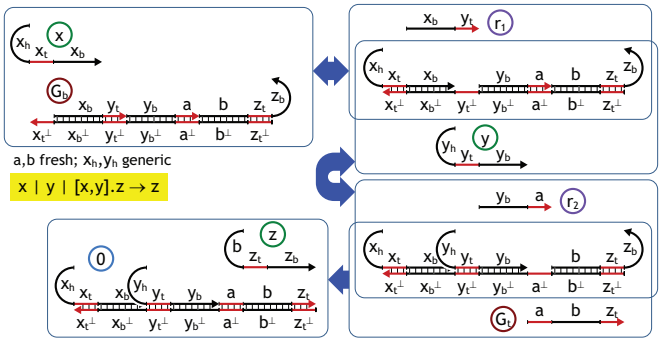

Fig. 7. 2-way Join - core function

of chemical reversibility. Given two inputs x,y, a G_b backbone releases two strands r_1 ,r_2, with r_1 providing reversibility while waiting for y (*cf.* Figure 3 in [23]); the trigger G_t finally irreversibly releases the output z (or outputs).

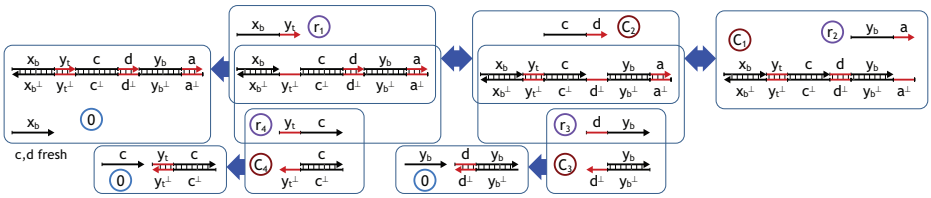

Fig. 8. 2-way Join - cleanup

In a cleanup phase (Figure 8), off the critical path, we use a similar C_1 structure (working from right to left) to remove r_1 and r_2 from the system, so that they do not accumulate to slow down further join operations. This phase is initiated by the release of r_2, so we know by construction that both r_1 and r_2 are available. Therefore, the r_3 and r_4 reversibility strands released by C_1 can be cleaned up immediately by C_3,C_4, ending a possible infinite regression of garbage collections.

We now analyze some subtle interference issues that frequently arise in gate design. In C_1 the interposed c, d double strand has the purpose of separating y_t from y_b: if omitted, a strand y_t, y_b would be released during garbage collection that would function as a y signal and interfere with other gates. Instead, a strand d, y_b is released (and then collected by C_3), and it is essential that $d \neq y_t$ to prevent interferences. Similarly, y_t, c is released (and then collected by C_4) and it is essential that $c \neq y_b$. Hence, c and d have to be chosen fresh; in particular, it is not possible to identify all the toeholds in a system: the d toeholds must be distinct from signal toeholds. Further,

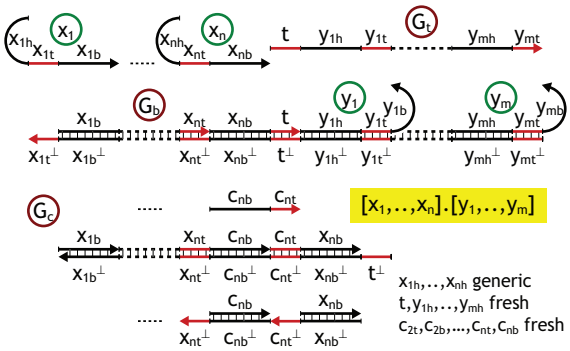

Fig. 9. $n \times m$ gate

the strand x_b, y_t is released from G_b and is absorbed by C_1; this is fine because x_b, y_t cannot function as a signal. However, no other signal in the whole system should have history x_b and toehold y_t, because it would be absorbed here. We should make sure, therefore, that all output signals have fresh history domains. The gates in Figures 5 and 6 *violate* this requirement by releasing an output whose history domain comes from the input, and therefore conflict with this join design: a solution is given next that also provides for general $n \times m$ gates. Finally, the G_b and C_1 structures are chained via the toehold a, which must be unique to each join gate.

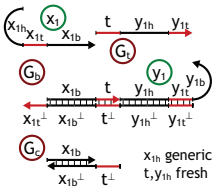

Fig. 10. 1×1 gate

It is possible to implement a 3-way join from 2-way joins and an extra signal x_0, but this encoding 'costs' a population and contains a divergent trace: $[x_1, x_2, x_3].x_4 \triangleq ([x_1, x_2].x_0 \mid x_0.[x_1, x_2])^* \mid [x_0, x_3].x_4$. Therefore, the strand algebra includes general n-way joins. Figure 9 describes a gates design that generalizes the 2-way join gate to a gate with n inputs and m outputs (in the garbage collection structures, c_i ranges from 2 to n). It has a single backbone where all the inputs are reversibly received, after which a trigger strand irreversibly releases all the outputs.

Special cases of this $n \times m$ gate therefore include a 1×1 transducer and a 1×2 fork: the 1×1 special case is shown in Figure 10. In contrast to the corresponding gates of Figures 5 and 6, the $1 \times m$ gates generate some garbage (x_1, t) that is however immediately collected. Moreover, all the output signals are released by the trigger strand, and we can make sure that the history domains of released signals are all fresh. Therefore, e.g., 1×1 and 1×2 gates no longer interfere with the garbage collection of 2×1 gates.

This completes the implementation of strand algebra in DNA. For the purposes of the next section, however, we consider also *curried* gates (gates that produce gates). Figure 11 shows a gate $x.H(y)$ accepting a signal x and activating the backbone H_b of

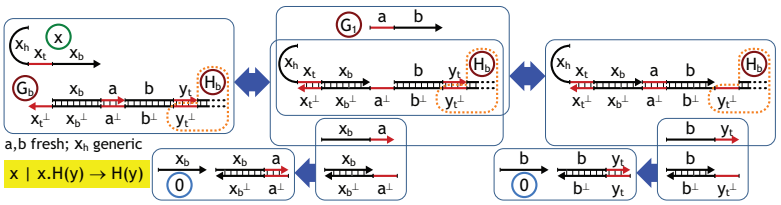

Fig. 11. Curried Gate

a gate $H(y)$, where $H(y)$ can be any gate with initial toehold $y_t^\perp$, including another curried gate. For example, if $H(y)$ is a transducer $y.z$ as in Figure 5, we obtain a curried gate $x.y.z$ such that $x \mid x.y.z \to y.z$. The interposed a, b domains prevent the release of a strand xb, yt that would interfere with r_1 of $[x, y].z$ in Figure 7.

4 Nested Strand Algebra

The purpose of this self-contained section is to allow nesting of join/fork operators, so that natural compound expressions can be written. We provide a uniform translation of this extended language back to the combinatorial strand algebra $\mathcal{P}$, as a paradigm for the compilation of higher level languages to DNA strands.

Consider a simple cascade of operations, $?x_1.!x_2.?x_3$, with the meaning of first taking an input ('?') x_1, then producing an output ('!') x_2, and then taking an input x_3. This can be encoded as follows:

$$?x_1.!x_2.?x_3 \triangleq x_1.[x_2, x_0] \mid [x_0, x_3].[]$$

where the right hand side is a set of $\mathcal{P}$ combinators, and where x_0 can be chosen fresh so that it does not interfere with other structures (although it will be used by all copies of $?x_1.!x_2.?x_3$).

The nested algebra $n\mathcal{P}$ admits such nesting of operators in general. The main change from the combinatorial $\mathcal{P}$ algebra consists in allowing syntactic nesting after an input or output prefix. This has the consequence that populations can now be nested as well, as in $?x.(P^*)$. The new syntax is:

$$P \ ::= \ x \mid ?[x_1, .., x_n].P \mid ![x_1, .., x_n].P \mid 0 \mid P_1|P_2 \mid P^* \qquad n \geq 1$$

Here $![x_1, .., x_n].P$ spontaneously releases $x_1, .., x_n$ into the solution and continues as P, while $?[x_1, .., x_n].P$ extracts $x_1, .., x_n$ from the solution (whenever they are all available) and continues as P. The mixing relation is the same as in $\mathcal{P}$. The reaction relation is modified only in the gate rule:

$?[x_1, .., x_n].P \mid x_1 \mid .. \mid x_n \to P$	input gate	(e.g.: $?x.0 \mid x \to 0$)
$![x_1, .., x_n].P \to x_1 \mid .. \mid x_n \mid P$	output gate	(e.g.: $!x.0 \to x \mid 0$)

Note that $\frac{?}{!}[...].x$ can always be replaced by $\frac{?}{!}[...].!x.0$, but x is kept in the syntax for convenience in expressing the reaction rules.

We now show how to compile $n\mathcal{P}$ to $\mathcal{P}$. Let $\mathcal{Z}$ be an infinite lists of distinct signals, and $\mathcal{F}$ be the set of such $\mathcal{Z}$. Let $\mathcal{Z}_i$ be the i-th signal in the list, $\mathcal{Z}_{\geq i}$ be the list starting at the i-th position of $\mathcal{Z}$, $evn(\mathcal{Z})$ be the even elements of $\mathcal{Z}$, and $odd(\mathcal{Z})$ be the odd elements. Let $\mathcal{F}_P$ be the set of those $\mathcal{Z} \in \mathcal{F}$ that do not contain any signal that occurs in P. The unnest algorithm $\mathcal{U}(P)_\mathcal{Z}$, for $P \in n\mathcal{P}$ and $\mathcal{Z} \in \mathcal{F}_P$, is shown in Definition 6. The inner loop $\mathcal{U}(z, P)_\mathcal{Z}$ uses z as the trigger for activating the translation of P.

Definition 6. *Unnest Algorithm*

$$
\begin{array}{lcl}
\mathcal{U}(P)_{\mathcal{Z}} & \triangleq & \mathcal{Z}_0 \mid \mathcal{U}(\mathcal{Z}_0, P)_{\mathcal{Z}_{\geq 1}} \\
\mathcal{U}(z,x)_{\mathcal{Z}} & \triangleq & z.x \\
\mathcal{U}(z,?[x_1,..,x_n].P)_{\mathcal{Z}} & \triangleq & [z,x_1,..,x_n].\mathcal{Z}_0 \mid \mathcal{U}(\mathcal{Z}_0, P)_{\mathcal{Z}_{\geq 1}} \\
\mathcal{U}(z,![x_1,..,x_n].P)_{\mathcal{Z}} & \triangleq & z.[x_1,..,x_n,\mathcal{Z}_0] \mid \mathcal{U}(\mathcal{Z}_0, P)_{\mathcal{Z}_{\geq 1}} \\
\mathcal{U}(z,0)_{\mathcal{Z}} & \triangleq & z.[] \\
\mathcal{U}(z,P' \mid P'')_{\mathcal{Z}} & \triangleq & z.[\mathcal{Z}_0,\mathcal{Z}_1] \mid \mathcal{U}(\mathcal{Z}_0,P')_{evn(\mathcal{Z}_{\geq 2})} \mid \mathcal{U}(\mathcal{Z}_1,P'')_{odd(\mathcal{Z}_{\geq 2})} \\
\mathcal{U}(z,P^*)_{\mathcal{Z}} & \triangleq & (z.[\mathcal{Z}_0,z] \mid \mathcal{U}(\mathcal{Z}_0,P)_{\mathcal{Z}_{\geq 1}})^*
\end{array}
$$

For example, the translations for $?x_1.![x_2,x_3].?x_4.0$ and $?x_1.(x_2^*)$ are:

$$
\begin{array}{lcl}
\mathcal{U}(?x_1.![x_2,x_3].?x_4.0)_{\mathcal{Z}} & = & \mathcal{Z}_0 \mid [\mathcal{Z}_0,x_1].\mathcal{Z}_1 \mid \mathcal{Z}_1.[x_2,x_3,\mathcal{Z}_2] \mid [\mathcal{Z}_2,x_4].\mathcal{Z}_3 \mid \mathcal{Z}_3.[] \\
\mathcal{U}(?x_1.(x_2^*))_{\mathcal{Z}} & = & \mathcal{Z}_0 \mid [\mathcal{Z}_0,x_1].\mathcal{Z}_1 \mid (\mathcal{Z}_1.[\mathcal{Z}_2,\mathcal{Z}_1] \mid \mathcal{Z}_2.x_2)^*
\end{array}
$$

In $?x_1.(x_2^*)$, activating x_1 once causes a linear production of copies of x_2. For an exponential growth of the population one should change $\mathcal{U}(z,P^*)_{\mathcal{Z}}$ to produce $(z.[\mathcal{Z}_0,z,z] \mid \mathcal{U}(\mathcal{Z}_0,P')_{\mathcal{Z}_{\geq 1}})^*$.

In the nested algebra we can easily solve systems of recursive definitions; for example, this recursive definition of X and Y processes:

$$
\begin{array}{lcl}
X & = & (?x_1.X \mid !x_2.Y) \\
Y & = & ?x_3.(X \mid Y)
\end{array}
$$

can be rewritten systematically as the following single term, where the X,Y 'processes' are replaced by ordinary signals:

$$
\begin{array}{l}
(?X.(?x_1.X \mid !x_2.Y))^* \mid \\
(?Y.?x_3.(X \mid Y))^*
\end{array}
$$

As an example of an $n\mathcal{P}$ program, consider a coffee vending machine controller, *Vend*, that accepts two coins for coffee. An *ok* is given after the first *coin* and then either a second *coin* (for coffee) or an *abort* (for refund) is accepted:

$$
\begin{array}{lcl}
\mathit{Vend} & = & ?\mathit{coin}.![\mathit{ok},\mathit{mutex}].(\mathit{Coffee} \mid \mathit{Refund}) \\
\mathit{Coffee} & = & ?[\mathit{mutex},\mathit{coin}].!\mathit{coffee}.(\mathit{Coffee} \mid \mathit{Vend}) \\
\mathit{Refund} & = & ?[\mathit{mutex},\mathit{abort}].!\mathit{refund}.(\mathit{Refund} \mid \mathit{Vend})
\end{array}
$$

Each Vend iteration spawns two branches, *Coffee* and *Refund*, waiting for either coin or abort. The branch not taken in the mutual exclusion is left behind; this could skew the system towards one population of branches. Therefore, when the *Coffee* branch is chosen and the system is reset to *Vend*, we also spawn another *Coffee* branch to dynamically balance the *Refund* branch that was not chosen; conversely for *Refund*.

5 Stochastic Strand Algebra

Stochastic strand algebra is obtained by assigning stochastic rates to gates, and by dropping the unbounded populations, P^*, which do not have a well-defined stochastic meaning; instead we write P^k as an abbreviation for k parallel copies of P, with $P^0 = 0$.

We assume that all gates with the same number n of inputs have the same stochastic rate g_n, collapsing all the gate parameters into a single effective parameter (the binding strengths of all toeholds of the same length are comparable [25]). Therefore, in this section $[x_1,..,x_n].[y_1,..,y_m]$ represents stochastic gates of rate g_n. Although

gate rates are fixed for each n, we can vary population sizes in order to differentiate macroscopic rates. We add two new flavors of gates: curried gates and persistent gates.

The curried gates described in Figure 11, when extended to multiple inputs and outputs in the style of Figure 9, imply that there is an extension of strand algebra with gates of the form

$$G \ ::= \ 0 \mid [x_1, .., x_n].[y_1, .., y_m, G]$$

where one of the outputs is another gate. In the combinatorial strand algebra, this extension can be encoded by setting, e.g., $x.y.z = [x].[[y].[z, 0]] \triangleq x.w \mid [w, y].z$ for a fresh w. But stochastically the encoding does not preserve rate behavior; therefore, in this stochastic strand algebra we take curried gates as primitive, implemented as in Figure 11: they will play a special role below.

We also add *persistent gates* $G^{=}$ (with $0^{=} = 0$). These gates have the same stochastic rate g_n as G, but are not consumed when used. Through them we can recover the role of P^* for implementing unbounded recursion, by using $G^{=k} \triangleq (G^{=})^k$ as a constant gate population of size k. Persistent gates can be encoded, up to an arbitrary precision, by regular gates that are buffered so that they are automatically replenished whenever used. We discuss the approximate implementation below, but for convenience we take persistent gates as primitive.

Terms of stochastic strand algebra therefore consists of strands, gates (null, curried, and persistent), and parallel composition of subsystems.

Definition 7. *Syntax of Stochastic Strand Algebra*

$$\begin{array}{lcll} G & ::= & 0 \mid [x_1, .., x_n].[y_1, .., y_m, G] & n \geq 1, m \geq 0 \\ P & ::= & x \mid G \mid G^{=} \mid P \mid P & \end{array}$$

The semantics of stochastic strand algebra is given by the continuous time Markov chain (CTMC) denoted by each term. A *state* of the Markov chain is a normalized term $P^\dagger$ of the form $x_1 \mid .. \mid x_n \mid 0 \mid G_1 \mid .. \mid G_m \mid G'^{=}_1 \mid .. \mid G'^{=}_p$, which is obtained from P by removing any repeated 0 terms and by lexicographically sorting the remaining terms. We write $P^\dagger.i$ for the i-th item in $P^\dagger$, and $P^\dagger \backslash i_1..i_n$ for the state obtained by removing the terms $P^\dagger.i_1, .., P^\dagger.i_n$.

A *labeled transition* is a 4-tuple written $l : P^\dagger \multimap^r Q^\dagger$, where l is a label (from some index set), r is a rate (a positive real), $P^\dagger$ is the initial state, and $Q^\dagger$ is the final state. A set of labeled transitions is a 4-place relation R; let $states(R)$ be the set of states that occur in R. The CTMC of a term P is obtained by defining the set of immediate transitions $Next(P)$ of P, and closing that set over further transitions to obtain the Labeled Transition Graph (LTG) Ψ of P. The labels in Ψ distinguish different ways of reaching one state from another, to properly account for multiplicity of transitions. The continuous time Markov chain $CTMC(\Psi)$ is finally obtained by collapsing all the transitions between two states and summing their rates, and also removing self-transitions. The CTMC is therefore a rate-weighted graph with states for nodes and with global transitions of the form $P^\dagger \multimap^r Q^\dagger$ with $P^\dagger \neq Q^\dagger$ for arcs. From such a graph Ψ, we can easily extract the transition matrix T of a continuous time Markov chain as normally presented, by setting $T_{ij} = r$ if $i \neq j$ and $i \multimap^r j \in \Psi$, and $T_{ii} = -\sum_{j \neq i} T_{ij}$.

Definition 8. *Labeled Transition Graph (LTG)*

A labeled transition graph is a set of transitions (quadruples) $l : P^\dagger \multimap^r Q^\dagger$ where l is from an index set, r is a positive real, and $P^\dagger$, $Q^\dagger$ are states.

Definition 9. *Continuous Time Markov Chain (CTMC) of an LTG*

Let Ψ be a labeled transition graph. Then: $CTMC(\Psi) \;=$
$\{P \rightharpoonup^r Q \;\mid\; (\exists\, l : P \rightharpoonup^s Q \in \Psi.\; P \neq Q) \;\wedge\; r = \sum r_i \;\; s.t. \;\; l_i : P \rightharpoonup^{r_i} Q \in \Psi\}$

The transition semantics simply states that each gate consumes its inputs and produces its outputs, and is itself consumed unless it is a persistent gate; the labels are finite sets of integers:

Definition 10. *Semantics of Stochastic Strand Algebra*

$Next(P) =$
$\{(\{p_1,..,p_n,q\} : P^\dagger \rightharpoonup^{g_n} Q^\dagger) \;\mid\; P^\dagger.p_i = x_i \wedge\; p_i \; distinct \; (i \in 1..n) \wedge$
$P^\dagger.q = [x_1,..,x_n].[y_1,..,y_m,G] \wedge Q = (P^\dagger \backslash p_1,..,p_n,q \mid y_1 \mid .. \mid y_m \mid G)\} \cup$
$\{(\{p_1,..,p_n,q\} : P^\dagger \rightharpoonup^{g_n} Q^\dagger) \;\mid\; P^\dagger.p_i = x_i \wedge\; p_i \; distinct \; (i \in 1..n) \wedge$
$P^\dagger.q = [x_1,..,x_n].[y_1,..,y_m,G]^= \wedge Q = (P^\dagger \backslash p_1,..,p_n \mid y_1 \mid .. \mid y_m \mid G)\}$

$LTG(P) = \bigcup_n \Psi_n$
$where \;\; \Psi_0 = Next(P) \;\; and \;\; \Psi_{n+1} = \bigcup\{Next(Q) \mid Q \in states(\Psi_n)\}$

For example, if $P = x^n \mid y^m \mid ([x,y].z)^p$, then $Next(P) = \{\{i,j,k\} : P^\dagger \rightharpoonup^{g_2} Q^\dagger \;\mid\; i \in 1..n, j \in n{+}1..n{+}m, k \in n{+}m{+}1..n{+}m{+}p, \; and \; Q = x^{n-1} \mid y^{m-1} \mid ([x,y].z)^{p-1} \mid z\}$. The first global transition in $CTMC(LTG(P))$ is then $P^\dagger \rightharpoonup^{n \times m \times p \times g_2} Q^\dagger$. Instead, if $P = x^n \mid ([x,x].y)^m$, then $Next(P) = \{\{i,j,k\} : P^\dagger \rightharpoonup^{g_2} Q^\dagger \; s.t. \; i \in 1..n, \; j \in 1..n, \; i \neq j, \; k \in n+1..n+m, \; and \; Q = x^{n-2} \mid ([x,x].y)^{m-1} \mid y\}$. The first global transition in $CTMC(LTG(P))$ is then $P^\dagger \rightharpoonup^{(n \times (n-1)/2) \times m \times g_2} Q^\dagger$. Note that in all cases the rate of the global transition is equal to the gate rate multiplied by all the possible ways of choosing inputs and gates that can react.

We next show how to approximate persistent gates $G^=$ by buffered gates $G^{(X,k)}$ for a large k, using a technique based on curried gates. (It would be possible to approximate persistent gates even with regular gates, but we would need to use an $n{+}1$-input stochastic gate to approximate an n-input persistent gate, requiring further assumptions on the relative rates of such gates.) Take:

G	$=$	$[x_1,..,x_n].[y_1,..,y_m,H]$	any non-persistent gate
G_X	$\triangleq$	$[x_1,..,x_n].[X,y_1,..,y_m,H]$	for any signal X not occurring in G
$G^{(X,k)}$	$\triangleq$	$X \mid (X.G_X)^k$	using the curried gate $X.G_X = [X].[G_X]$

Here k is the size of a buffer population for G_X, and X is the trigger that replenishes the gate G_X from the buffer (we discuss a single persistent gate, but one can similarly consider a persistent gate population that shares the same trigger). Consider a context P that does not contain X, and suppose there is a reaction between $G^=$ and signals in P, with global transition:

$(G^= \mid P)^\dagger \rightharpoonup^{p \times g_n} (G^= \mid Q)^\dagger$ with $Q = y_1 \mid .. \mid y_m \mid H \mid P^\dagger \backslash x_1,..,x_n$

where p is the number of possible such reactions, and $P^\dagger \backslash x_1,..,x_n$ stands for the appropriate index-based removal of the input signals. Let us now replace $G^=$ with $G^{(X,k)}$. By interaction on X, which produces the only possible reaction out of $G^{(X,k)}$, a first global transition is:

$(G^{(X,k)} \mid P)^\dagger = (X \mid (X.G_X)^k \mid P)^\dagger \rightharpoonup^{k \times g_1} ((X.G_X)^{k-1} \mid G_X \mid P)^\dagger$

The previous reaction between $G^=$ and P now leads to a reaction between G_X and P:

$((X.G_X)^{k-1} \mid G_X \mid P)^\dagger \rightharpoonup^{p \times g_n} ((X.G_X)^{k-1} \mid X \mid Q)^\dagger = (G^{(X,k-1)} \mid Q)^\dagger$

For arbitrarily large k, the combined rate of those two global transitions approximates $p \times g_n$, and the further reductions involving $G^{(X,k-1)}$ approximate those involving $G^{(X,k)}$, the only difference being the arbitrarily large interaction on X. Therefore,

the transition $(G^{=} \mid P)^{\dagger} \multimap^{p \times g_n} (G^{=} \mid Q)^{\dagger}$ is approximated to an arbitrary precision by the transition $(G^{(X,k)} \mid P)^{\dagger} \multimap^{p \times g_n} (G^{(X,k)} \mid Q)^{\dagger}$. In this sense $G^{(X,k)}$ approximates $G^{=}$.

For a chain of transitions, the initial k must be large enough to support the whole chain, and if the chain is unbounded then the approximation will eventually fail; that is, the buffer will run out. However, it is possible to periodically replenish the buffer by external intervention: this can cause a major rate change, but only on the X reactions that are arbitrarily fast, and *does not disturb the rest of the system*, or more precisely, disturbs the interactions on G_X arbitrarily little. The perturbation on G_X depends the size of k and on the frequency of refills; the latter cannot be bounded and requires monitoring of the system. Still, this technique provides a practical way of implementing unbounded computation, by periodically 'topping-up' the buffer populations. Also, in a sense, it emulates the stable conditions maintained in a bioreactor.

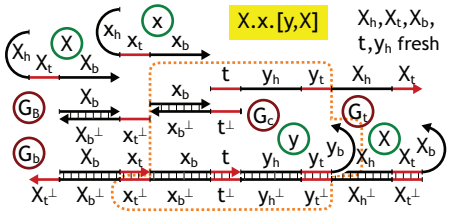

Fig. 12. Buffered Transducer

Figure 12 shows the structure of a buffered $x.y$ transducer, implemented as the curried gate $X.x.[y, X]$, where the a, b domains and related garbage collection from Figure 11 (which are generally needed to avoid interference on X) have been removed under the assumption that X is private to this gate. Compare the dashed region with the transducer from Figure 10: the extension to $n \times m$ buffered gates is uniform, as in Figure 9.

In the *coarse-grain* stochastic semantics of Definition 10 we have assumed that each gate performs a single effective transition from its inputs to its output, considering g_n as the 'rate-limiting' step of a sequence of underlying chemical reactions. A *fine-grain* stochastic semantics can be given instead by directly emulating all the intermediate species and the chemical reactions that are described in the figures of Section 3. In fact, such a fine-grain semantics can be given by a formal translation into the chemical process algebra of Introduction, assigning appropriate DNA structures to the species and stochastic rates to the reactions. The fine-grain semantics is *not* stochastically equivalent to the coarse-grain semantics, because it makes finer steps. Even just as a nondeterministic semantics, it contains unbounded traces (due to reversibility) that are not present in the coarse-grain semantics. However, the fine-grain semantics can be engineered to approximate the coarse-grain semantics, therefore providing an accurate chemical (DNA) implementation of the stochastic strand algebra. These techniques have been developed in [23], where abstract chemical systems (akin to our abstract strand algebra reactions) are implemented by concrete, finer, chemical systems made of DNA molecules, in such a way that the concrete chemical systems approximate accurately the behavior of the abstract chemical systems.

6 Expressiveness – Compiling to DNA

In this section we consider four equally expressive formalisms, and relate them: stochastic strand algebra, finite stochastic chemical systems [21], interacting automata [5,7], and stochastic place-transition Petri nets. An interesting point is that, by the connection between these systems, we can implement each of them in DNA by the methods of Section 3.

6.1 Finite Stochastic Chemical Systems

A *finite stochastic chemical reaction network* (SCRN) [22] is a finite set of stochastic chemical reactions of the form:

$C = \{\rho_j : Lhs_j \rightarrow^{r_j} Rhs_j\}$ for j ranging over a finite set

where: ρ_j is a unique name for each reaction; Lhs_j has the form $X_1 + ... + X_{n_j}$ for $n_j > 0$ where X are *chemical species*; Rhs_j has the form $X_1 + ... + X_{m_j}$ for $m_j \geq 0$; and r_j is a positive real of dimension s^{-1} representing the number of reactions per seconds (the *rate* of the reaction). A *finite stochastic chemical system* (SCS) consists of an SCRN, C, together with initial conditions, P, given as a finite number of copies (*molecules*) of each species, which are understood to exist within a given volume. We write $P^\dagger$ for the lexicographical ordering of a multiset of molecules $P = X_1 + ... + X_p$. If we assume that r_j are rational numbers, then, without loss of generality, we can take them to be integer numbers, because we can scale the physical dimension of the r_j. The kinetics of an SCS is given by its continuous time Markov chain, $CTMC(LTG(C, P))$, defined similarly to Definition 10, where the states are $P^\dagger$ and the labels of the transition graph are sets of indexes in $P^\dagger$ and names of reactions:

Definition 11. *Semantics of Finite Stochastic Chemical Systems*

$$Next(C, P) =$$
$$\{(\{p_1, .., p_n, \rho\} : P^\dagger \multimap^r Q^\dagger) \mid P^\dagger.p_i = X_i \wedge\ p_i \text{ distinct } (i \in 1..n) \wedge$$
$$(\rho : X_1 + .. + X_n \rightarrow^r Y_1, .., Y_m) \in C \wedge Q = P^\dagger \backslash p_1, .., p_n + Y_1 + .. + Y_m\}$$

$$LTG(C, P) = \bigcup_n \Psi_n$$
$$\text{where } \Psi_0 = Next(C, P) \text{ and } \Psi_{n+1} = \bigcup\{Next(C, Q) \mid Q \in states(\Psi_n)\}$$

An SCS can be translated to strand algebra by mapping each n-ary reaction of rate r to a population of persistent gates of size r/g_n. Here $Parallel\{\rho_1 : P_1, ..., \rho_n : P_n\} = P_1 \mid ... \mid P_n$ (the ρ_i handle multiplicities), and $Proc(X_1 + .. + X_n) = X_1 \mid ... \mid X_n$:

Definition 12. *From SCS to Stochastic Strand Algebra*

Let (C, P) be a Finite Stochastic Chemical System:

$$Strand(C) = Parallel\{\rho : [X_1, .., X_n].[Y_1, .., Y_m]^{=r/g_n} \mid (\rho : X_1 + .. + X_n \rightarrow^r Y_1 + .. + Y_m) \in C\}$$
$$Strand(C, P) = Strand(C) \mid Proc(P)$$

Provided that, starting from rational rates, we have scaled the physical dimensions of r and g_n so that all r/g_n are integer numbers.

Theorem 1. *From SCS to Stochastic Strand Algebra*

The semantics of an SCS (C, P) and of its Stochastic Strand Algebra translation $Strand(C, P)$ coincide: $CTMC(LTG(C, P)) = CTMC(LTG(Strand(C, P)))$

Proof. By Definitions 11 and 10, let a state in $CTMC(LTG(C,P))$ be $P^\dagger$; the corresponding state in $CTMC(LTG(Strand(C,P)))$ is $(Proc(P) \mid Strand(C))^\dagger$; note that the lexicographical indexing of $P^\dagger$ remains consistent in $(Proc(P) \mid Strand(C))^\dagger$ by sorting strands before gates. By Definition 12, for each transition $\{p_1,..,p_n,\rho\} : P^\dagger \multimap^r Q^\dagger$ in $LTG(C,P)$ due to reaction $\rho : X_1+..+X_n \to^r Y_1+..+Y_m$ and molecules $P^\dagger.p_i$, there are r/g_n copies of $[X_1,..,X_n].[Y_1,..,Y_m]^=$ in $Strand(C)$, and hence r/g_n transitions $(Proc(P) \mid Strand(C))^\dagger \multimap^{g_n} (Proc(Q) \mid Strand(C))^\dagger$ in $LTG(Strand(C,P))$ due to strands $Proc(P)^\dagger.p_i$. The contribution r of $\{p_1,..,p_n,\rho\} : P^\dagger \multimap^r Q^\dagger$ to transition from $P^\dagger$ to $Q^\dagger$ in $CTMC(LTG(C,P))$ is therefore matched by the contribution $r/g_n \times g_n = r$ of $\rho : [X_1,..,X_n].[Y_1,..,Y_m]^{=r/g_n}$ to transitions from $(Proc(P) \mid Strand(C))^\dagger$ to $(Proc(Q) \mid Strand(C))^\dagger$ in $CTMC(LTG(Strand(C,P)))$.

Therefore, we can translate Finite Stochastic Chemical Systems to Stochastic Strand Algebra while preserving their kinetics. With the further assumption that stochastic strand algebra can be correctly implemented in DNA, we can then translate any abstract chemical system to a concrete DNA-based chemical system, at least up to the correct implementation of a fine-grain stochastic semantics: see [23].

We have also shown that stochastic strand algebra is at least as expressive as SCS (in fact, it is no more expressive, as discussed later). We have a particular use in mind for Theorem 1; that is, as a stepping stone for the molecular implementation of interacting automata, as discussed next.

6.2 Interacting Automata

Interacting automata [5,7] (a stochastic subset of CCS [15]) are finite state automata that interact with each other over synchronous stochastic channels. An interaction can happen when two automata choose the same channel c_r, with rate r, one as input ($?c_r$) and the other as output ($!c_r$). As a molecular analogy, these automata represent stateful molecules that 'collide' pairwise on complementary exposed surfaces (channels) and change states as a result of the collision. Figure 13 shows two such automata systems E_1 and E_2, where each diagram represents a population of identical automata interacting with each other (and in general with other populations). For example, in system E_1, two automata in states A_1 and B_1 can either collide on channel a at rate r by complementary interactions $!a$ and $?a$, and each move to state A_1, or ($\oplus$) collide on channel b at rate s and each move to state B_1 (see [4] for many other examples).

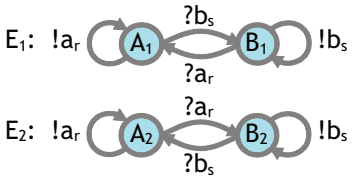

Fig. 13. Interacting Automata

In general, a system of interacting automata is given by a system E of *equations* of the form $X = M$, where X is a *species* (an automaton state) and M is a *molecule* (representing the transition table from state X). M has the form $\pi_1.P_1 \oplus \ldots \oplus \pi_n.P_n$, where $\oplus$ is stochastic choice among possible interactions, P_i are multisets of resulting species, and π_i are either delays τ_r, inputs $?c_r$, or outputs $!c_r$ on channel c at rate r.

$$E_1: \quad A_1 = !a_r.A_1 \oplus ?b_s.B_1 \qquad E_2: \quad A_2 = !a_r.A_2 \oplus ?a_r.B_2$$
$$\qquad\quad B_1 = !b_s.B_1 \oplus ?a_r.A_1 \qquad\qquad\quad B_2 = !b_s.B_2 \oplus ?b_s.A_2$$

Figure 14 shows an example of a 3-state automaton that exhibits an interesting kinetics: a Gillespie simulation of 1500 such automata (with given initial conditions, and $r = 1.0$) produces an oscillation in the number of automata in a given state A, B, or C. The corresponding equation system and its translation to strand algebra (described in Definition 15) are:

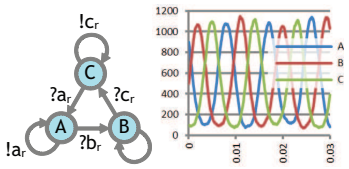

Fig. 14. Oscillator

$$\begin{array}{ll} A = !a_r.A \oplus ?b_r.B & ([A,B].[B,B])^{=r/g2} \mid \\ B = !b_r.B \oplus ?c_r.C & ([B,C].[C,C])^{=r/g2} \mid \\ C = !c_r.C \oplus ?a_r.A & ([C,A].[A,A])^{=r/g2} \mid \\ A^{900} \mid B^{500} \mid C^{100} & A^{900} \mid B^{500} \mid C^{100} \end{array}$$

The continuous time Markov chain semantics of [5] prescribes the possible transitions $\rightharpoonup^p$ between automata states, with their propensities p. For example, consider the systems E_1 and E_2 with initial conditions $A_i^n \mid B_i^m$; that is, with n automata in state A_i and m in state B_i. In E_1 on channel a_r the propensity of the transition to $A_1^{n+1} \mid B_1^{m-1}$ is $n \times m \times r$, which is the number of possible binary interactions multiplied by the channel rate. In E_2 on channel a_r, with two symmetric ?/! ways for A_2 to collide with A_2, the propensity of the transition to $A_2^{n-1} \mid B_2^{m+1}$ is $2 \times (n \; choose \; 2) \times r = n \times (n-1) \times r$, which is again the number of possible binary interactions multiplied by the channel rate:

$$\begin{array}{lll} A_1^n \mid B_1^m : (on\ a_r) & A_1^n \mid B_1^m \rightharpoonup^{n \times m \times r} & A_1^{n+1} \mid B_1^{m-1} \\ \qquad\qquad (on\ b_s) & A_1^n \mid B_1^m \rightharpoonup^{n \times m \times s} & A_1^{n-1} \mid B_1^{m+1} \end{array}$$

$$\begin{array}{lll} A_2^n \mid B_2^m : (on\ a_r) & A_2^n \mid B_2^m \rightharpoonup^{n \times (n-1) \times r} & A_2^{n-1} \mid B_2^{m+1} \\ \qquad\qquad (on\ b_s) & A_2^n \mid B_2^m \rightharpoonup^{m \times (m-1) \times s} & A_2^{n+1} \mid B_2^{m-1} \end{array}$$

Subsequent transitions from the new states are computed in the same way.

Interacting automata can be emulated in stochastic strand algebra, preserving their possible transitions and propensities. This is achieved by a translation that generates a binary join gate for each possible collision, choosing stable population sizes that produce the prescribed rates. This translation can cause a quadratic expansion of the representation [5], meaning that the translation is non-trivial, and that interacting automata provide a more compact representation of a CTMC than either strand algebra or chemical reactions. Achieving a linear translation is an open problem, which likely requires an extension of the strand algebra. For example, if we had a suitable *choice* operator in strand algebra, then the choice operator $\oplus$ of interacting automata could be translated linearly to it, instead of by quadratically expanding all possible interactions due to different choices. It is not clear, though, how such a choice operator could be defined.

We recall the definitions of syntax and stochastic semantics from [5], where $M.i$ is the i-th summand in M, $E.X$ is the M associated to X in E, and $E.X.i$ refers to the summand $(E.X).i$:

Definition 13. *Syntax of Interacting Automata*

E	::=	$0 \mid X = M, E$	*Reagents*	*(a list of reagents $X = M$)*
M	::=	$0 \mid \pi.P \oplus M$	*Molecule*	*(a list of interactions $\pi.P$)*
P	::=	$0 \mid X \mid P$	*Solution*	*(a list of species X)*
π	::=	$\tau_r \mid ?a_r \mid !a_r$	*Interaction*	*(delay, input, output)*

Definition 14. *Semantics of Interacting Automata*

$$Next(E,P) =$$
$$\{(\{m.X.i\} : P^\dagger \multimap^r Q^\dagger) \mid P^\dagger.m = X \wedge E.X.i = \tau_r.R \wedge Q = (P^\dagger \backslash m \mid R)\} \cup$$
$$\{(\{m.X.i, n.Y.j\} : P^\dagger \multimap^r Q^\dagger) \mid P^\dagger.m = X \wedge P^\dagger.n = Y \wedge m \neq n \wedge$$
$$E.X.i = ?a_r.R \wedge E.Y.j = !a_r.S \wedge Q = (P^\dagger \backslash m, n \mid R \mid S)\}$$

$$LTG(E,P) = \bigcup_n \Psi_n$$
$$\text{where } \Psi_0 = Next(E,P) \text{ and } \Psi_{n+1} = \bigcup\{Next(E,Q) \mid Q \in states(\Psi_n)\}$$

Consider now the following translation from interacting automata to stochastic strand algebra. The labeling accounts for multiplicities; a label is either a singleton $\langle X.i \rangle$ or an ordered pair $\langle X.i, Y.j \rangle$, where $X.i$ are also ordered pairs.

Definition 15. *From Interacting Automata to Stochastic Strand Algebra*

$$Strand(E) = Parallel($$
$$\{(\langle X.i \rangle : X.[P]^{=r/g_1}) \mid E.X.i = \tau_r.P\} \cup$$
$$\{(\langle X.i, Y.j \rangle : [X,Y].[P,Q]^{=r/g_2}) \mid X \neq Y \wedge E.X.i = ?a_r.P \wedge E.Y.j = !a_r.Q\} \cup$$
$$\{(\langle X.i, X.j \rangle : [X,Y].[P,Q]^{=2r/g_2}) \mid E.X.i = ?a_r.P \wedge E.X.j = !a_r.Q\})$$
$$Strand(E,P) = Strand(E) \mid Proc(P)$$

The examples above, in particular, translate as follows, with $P_1 = Strand(E_1)$ and $P_2 = Strand(E_2)$:

$$P_1 = [B_1, A_1].[A_1, A_1]^{=r/g_2} \mid [A_1, B_1].[B_1, B_1]^{=s/g_2} \qquad P_2 = [A_2, A_2].[B_2, A_2]^{=2r/g_2} \mid [B_2, B_2].[A_2, B_2]^{=2s/g_2}$$

Initial automata states are translated identically into initial signals and placed in parallel. As described in Section 5, a strand algebra transition from global state $A^n \mid B^m \mid [A,B].[C,D]^{=p}$ has propensity $n \times m \times p \times g_2$, and from $A^n \mid [A,A].[C,D]^{=p}$ has propensity $(n \; choose \; 2) \times p \times g_2$. From the same initial conditions $A^n \mid B^m$ as in the automata, we then obtain the global strand algebra transitions:

$$A_1^n \mid B_1^m \mid P_1 \multimap^{n \times m \times r/g_2 \times g_2} A_1^{n+1} \mid B_1^{m-1} \mid P_1$$
$$A_1^n \mid B_1^m \mid P_1 \multimap^{n \times m \times s/g_2 \times g_2} A_1^{n-1} \mid B_1^{m+1} \mid P_1$$

$$A_2^n \mid B_2^m \mid P_2 \multimap^{(n \times (n-1))/2 \times 2r/g_2 \times g_2} A_2^{n-1} \mid B_2^{m+1} \mid P_2$$
$$A_2^n \mid B_2^m \mid P_2 \multimap^{(m \times (m-1))/2 \times 2s/g_2 \times g_2} A_2^{n+1} \mid B_2^{m-1} \mid P_2$$

which have the same propensities and corresponding states as the interacting automata transitions. In general, we obtain that interacting automata can be translated faithfully:

Theorem 2. *From Interacting Automata to Stochastic Strand Algebra*

The semantics of interacting automata and of their translation to stochastic strand algebra coincide: $CTMC(E,P) = CTMC(Strand(E,P))$.

Proof. We blur over the syntactic differences between "|" in interacting automata and strand algebra expression, and "+" in chemical expression, assuming they are converted as appropriate. The following translation $Ch(E,P)$ from interacting automata (E,P) to Finite Stochastic Chemical Systems $Ch(E),P$ is such that $CTMC(Ch(E,P))$ $= CTMC(E,P)$ ([5], Theorem 3.4-2):

$Ch(E) =$

$\{(\langle X.i\rangle : X \to^r P) \mid E.X.i = \tau_r.P\} \cup$

$\{(\langle X.i, Y.j\rangle : X + Y \to^r P + Q) \mid X \neq Y \wedge E.X.i = ?a_r.P \wedge E.Y.j = !a_r.Q\} \cup$

$\{(\langle X.i, X.j\rangle : X + X \to^{2r} P + Q) \mid E.X.i = ?a_r.P \wedge E.X.j = !a_r.Q\}$

By Theorem 1 $CTMC(Strand(Ch(E,P)) = CTMC(Ch(E,P))$. Therefore we obtain that $CTMC(Strand(Ch(E,P)) = CTMC(E,P)$; that is, there is a translation $Strand(Ch(E,P))$ from interacting automata to stochastic strand algebra that preserves the CMTC semantics. By composing $Strand(C,P)$ from Definition 12 with $Ch(E,P)$ form above, we obtain $Strand(E,P)$ as in Definition 15.

6.3 Petri Nets

A place-transition Petri net (the simplest kind of Petri nets [18]) has *places* marked with a number of *tokens*, and *transitions* between n incoming and m outgoing places that *fire* when all the incoming places have at least one taken, which is removed, and a token is then added to each outgoing place. Consider a place-transition Petri Net with places x_i; then, a transition with incoming arcs from places $x_1..x_n$ and outgoing arcs to places $y_1..y_m$ is represented in the combinatorial strand algebra as $([x_1,..,x_n].[y_1..y_m])^*$, where an unbounded population of gates ensures that the transition can fire repeatedly. The initial token marking $x_1,..,x_k$ (a multiset of places) is represented as $x_1 \mid .. \mid x_k$. Conversely, a signal in strand algebra can be represented as a marked place in a Petri net, and a gate $[x_1,..,x_n].[y_1..y_m]$ as a transition with an additional marked 'one-shot' place on the input that makes it fire only once; then, P^* can be represented by connecting the transitions of P to refresh the one-shot places (this was suggested by Cosimo Laneve). Therefore, the combinatorial strand algebra is equivalent to place-transition Petri nets, with strand algebra providing a compositional language for describing such nets.

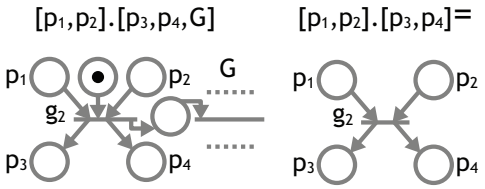

Fig. 15. Stochastic Petri Nets

We are interested here specifically in the translation from stochastic strand algebra to stochastic nets (which are just nets with rates on transitions and with an induced CTMC semantics). In this case, we can translate signals as marked places, as above, and gates as transitions with associated rates. In the stochastic version we do not have P^* to handle any more: just multisets of gates, but including persistent and curried gates. Figure 15 shows the translation by two general examples: on the left, a curried gate with a one-shot place, where the continuation gate G (if any) is activated

by a connecting place; on the right, a persistent gate as a normal transition, where a continuation gate (not shown) can be handled as on the left.

6.4 Equivalences

We have seen that interacting automata can be represented in stochastic strand algebra (Section 6.2) via SCS (Section 6.1), and that stochastic strand algebra can be represented as stochastic place-transition nets (Section 6.3). The latter can be easily represented as SCS where each transition corresponds to a chemical reaction [23,7]. Finally, SCS can be represented as interacting automata [5].

Therefore, all these formal systems are equivalent, and by the techniques of Section 3 they can all be compiled to DNA. Moreover, the compilation technique is uniform, through the common connection to strand algebra.

The computational power of this class of systems is well characterized. For the nondeterministic case, many questions about reachable configurations in Petri nets are decidable [18], and therefore this class of systems is not Turing-complete. Still, this class is widely used and studied, due to the analysis of concurrency that come from the theories of Petri nets and Process Algebra, and somewhat separately from the theory of Distributed Systems. As for the latter, a computation in this class can be seen as a *population protocol* by defining a predicate over configurations that always becomes stable under fair executions [1]; the predicates that are stably computable this way coincide with the predicates over initial states expressible in Presburger arithmetic, which is again a decidable class. The fairness assumption of population protocols matches abstractly the probabilistic fairness implicit in stochastic systems, and the assumption that all agents can interact match the chemical assumption of well-mixing. It has been shown that Turing machines can be emulated up to an arbitrarily small error [1,22], and these results have been transferred to Process Algebra [27].

7 Contributions and Conclusions

We have introduced strand algebras: formal languages based on a simple relational semantics that are equivalent to place-transition Petri nets, but allow for compositional descriptions where each component maps directly to DNA structures. Strand algebras connect a simple but powerful class of DNA system to a rich set of techniques from process algebra for studying concurrent systems: such techniques will be required in the verification and optimization of DNA-based computing architectures, where many subtle interactions may occur. Detailed analysis of system kinetics requires the introduction of quantitative parameters; to this end we have described a stochastic strand algebra and a technique for maintaining stable buffered populations to support indefinite and unperturbed computation in a stochastic setting. Using strand algebra as a stepping stone, we have described a DNA implementation of interacting automata that preserves stochastic behavior. Interacting automata are one of the simplest process algebras in the literature. Hopefully, more advanced process algebra operators will eventually be implemented as DNA structures, and conversely more complex DNA structures will be captured at the algebraic level, leading to more expressive concurrent languages for programming molecular systems.

Within the strand algebra framework, beyond the specific algebras we have presented, it is easy to add operators for new DNA structures, and to map existing operators to alternative DNA implementations. We have also shown how to use strand algebra as an intermediate compilation language, by giving a translation from a more

convenient expression-based syntax. However, the goal of "compiling high-level languages to DNA", is clearly not yet fulfilled. Above the level of strand algebra we have available a number of circuit abstractions (the ones discussed in Section 6 and possibly many others), but there are no clear candidates for high-level programming languages tailored to DNA nanosystems. Below the level of strand algebra there is an increasing number of structural description languages [16] and thermodynamic analysis tools [10], but the process of reliably producing DNA nanosystems from general structural descriptions is still being worked out.

I would like to acknowledge the Molecular Programming groups at Caltech and U.Washington for invaluable discussions and corrections. In particular, join and curried gate designs were extensively discussed with Lulu Qian, David Soloveichik and Erik Winfree [8]. A preliminary version of this paper appeared as reference [6].

References

1. D. Angluin, J. Aspnes, Z. Diamadi, M.J. Fischer, R. Peralta. Computation in Networks of Passively Mobile Finite-state Sensors. Distributed Computing. Mar 2006, 235-253.
2. Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, E. Shapiro. Programmable and Autonomous Computing Machine made of Biomolecules. Nature, 414(22), 2001.
3. G. Berry, G. Boudol. The Chemical Abstract Machine. Proc. 17th POPL, ACM, 81-94, 1989.
4. L. Cardelli: Artificial Biochemistry. In: A. Condon, D. Harel, J.N. Kok, A. Salomaa, E. Winfree (Eds.). Algorithmic Bioprocesses. Springer, 2009.
5. L. Cardelli: On Process Rate Semantics. Theoretical Computer Science 391(3) 190-215, 2008.
6. L. Cardelli. Strand Algebras for DNA Computing (Preliminary version). DNA Computing and Molecular Programming, 15th International Conference, DNA 15. LNCS 5877:12-24, Springer, 2009.
7. L. Cardelli, G. Zavattaro: Turing Universality of the Biochemical Ground Form. Mathematical Structures in Computer Science, 20(1):45-73, February 2010.
8. L. Cardelli, L. Qian, D. Soloveichik, E. Winfree. Personal communications, 2009.
9. V. Danos, C. Laneve. Formal molecular biology. Theoretical Computer Science 325(1) 69-110. 2004.
10. R.M. Dirks, J.S. Bois, J.M. Schaeffer, E. Winfree, and N.A. Pierce. Thermodynamic analysis of interacting nucleic acid strands. SIAM Rev, 49, 65-88, 2007.
11. C. Fournet, G. Gonthier. The Join Calculus: a Language for Distributed Mobile Programming. In Proceedings of the Applied Semantics Summer School (APPSEM), Caminha, 9-15 September 2000.
12. M. Hagiya. Towards Molecular Programming. In G. Ciobanu, G. Rozenberg, (Eds.) Modelling in Molecular Biology. Springer, 2004.
13. L. Kari, S. Konstantinidis, P. Sosík. On Properties of Bond-free DNA Languages. Theoretical Computer Science 334(1-3) 131-159, 2005.
14. A. Marathe, A.E. Condon, R.M. Corn. On Combinatorial DNA Word Design. J. Comp. Biology 8(3) 201–219, 2001.
15. R. Milner. Communicating and Mobile Systems: The π-Calculus. Cambridge University Press, 1999.
16. A. Phillips, L. Cardelli. A Programming Language for Composable DNA Circuits. Journal of the Royal Society Interface, August 6, 2009 6:S419-S436.
17. L. Qian, E. Winfree. A Simple DNA Gate Motif for Synthesizing Large-scale Circuits. Proc. 14th International Meeting on DNA Computing. 2008.
18. W. Reisig. Petri Nets: An Introduction. Springer-Verlag, 1985.
19. A. Regev, E.M. Panina, W. Silverman, L. Cardelli, E. Shapiro. BioAmbients: An Abstraction for Biological Compartments. Theoretical Computer Science 325(1) 141-167, 2004.

20. K. Sakamoto, D. Kiga, K. Komiya, H. Gouzu, S. Yokoyama, S. Ikeda, H. Sugiyama, M. Hagiya: State Transitions by Molecules. Biosystems 52, 81–91, 1999.
21. G. Seelig, D. Soloveichik, D.Y. Zhang, E. Winfree. Enzyme-Free Nucleic Acid Logic Circuits. Science, 314(8), 2006.
22. D. Soloveichik, M. Cook, E. Winfree, J. Bruck. Computation with Finite Stochastic Chemical Reaction Networks. Natural Computing, 7:615–633. 2008.
23. D. Soloveichik, G. Seelig, E. Winfree. DNA as a Universal Substrate for Chemical Kinetics. PNAS, March 4, 2010, doi: 10.1073/pnas.0909380107.
24. O. Wolkenhauer, M. Ullah, W. Kolch, K. Cho. Modelling and simulation of intracellular dynamics: Choosing an appropriate framework. IEEE Transactions on NanoBioscience 3, 200-207. 2004.
25. P. Yin, H. M.T. Choi, C.R. Calvert, N.A. Pierce. Programming Biomolecular Self-assembly Pathways. Nature, 451:318-322, 2008.
26. B. Yurke, A.P. Mills Jr. Using DNA to Power Nanostructures, Genetic Programming and Evolvable Machines archive 4(2), 111 - 122, Kluwer, 2003.
27. G. Zavattaro, L. Cardelli. Termination Problems in Chemical Kinetics. In F.van Breugel, M.Chechik (Eds.) CONCUR 2008 - Concurrency Theory, 19th International Conference. LNCS 5201, pp 477-491, Springer 2008.
28. D.Y. Zhang, A.J. Turberfield, B. Yurke, E. Winfree. Engineering Entropy-driven Reactions and Networks Catalyzed by DNA. Science, 318:1121-1125, 2007.