Strand Algebras for DNA Computing

Luca Cardelli

Microsoft Research

Abstract

We present a process algebra for DNA computing, and we discuss compilation of various other formal systems into the algebra, and compilation of the algebra into DNA structures.

1 Introduction

DNA technology is reaching the point where one can envision automatically compiling high-level formalisms to DNA computational structures [16]. Examples so far include the 'manual compilation' of automata and Boolean networks, where some impressive demonstrations have been carried out [1][8][13][14]. Typically one considers sequential or functional computations, realized by massive numbers of molecules; we should strive, however, to take more direct advantage of massive concurrency 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 [15]: 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 low-level combinatorial algebra, as a paradigm for compiling expressions of arbitrary complexity to 'assembly language' DNA combinators.

Third is our original motivation: translating heterogeneous collections of interacting automata [4] to molecular structures. How to do that was initially unclear, because one must choose some suitable 'programmable matter' (such as DNA) as a substrate, but must also come up with compositional protocols for interaction of the components that obey the high-level semantics of the language. We show a solution to this problem in Section 5.1.4, based on the combinatorial DNA algebra. The general issue there 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. What can be implemented directly in DNA, is instead a *join* primitive, based on [15]: this is a powerful operator, widely studied in concurrency theory [7], which can indirectly provide an implementation of external choice. The DNA algebra supporting the translation is built around joins operators.

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 [2], 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 it may enable additional reactions). The two

relations of mixing and reaction, are connected by a rule that says that the solution is well mixed. It is also useful to consider the symmetric and transitive closure, $\rightarrow^*$, representing sequences of reactions. 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 $\rightarrow$ describing whole-system transitions, one can generate a continuous-time Markov chain representing the kinetics of the system.

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

1.1–1 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
$P \equiv Q \quad \Rightarrow \quad P + R \equiv Q + R$	in context
$P + Q \equiv Q + P; \quad P + (Q + R) \equiv (P + Q) + R; \quad P + 0 \equiv P$	diffusion
$P \rightarrow Q \quad \Rightarrow \quad P + R \rightarrow Q + R$	dilution
$P \equiv P',\ P' \rightarrow Q',\ Q' \equiv Q \quad \Rightarrow \quad P \rightarrow Q$	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: $H + H \rightarrow H_2$; $H + O \rightarrow OH$; $H_2 + O \rightarrow H_2O$; $H + OH \rightarrow H_2O$. 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 $H + O + H \rightarrow\rightarrow H_2O$, that is we can produce water molecules in two steps (and by two different paths), and that $H+H+H+H+O+O \rightarrow^* H_2O + H_2O$. Chemical evolution is therefore encoded in the two relations of mixing and reaction: a solution P can evolve to a solution Q iff $\langle P,Q\rangle \in \rightarrow^*$. Algebra is about equations, but 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 [10]. The complexity of these derived equational theories varies with the algebra. A simple instance here is the equation P + 0 = P, which requires verifying that in our definition of $\rightarrow$ 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 *bio*chemistry more directly [6][12] or, as in this paper, that can suit DNA computing.

2 Strand Algebras

By a *strand algebra* we mean a process algebra [10] where the main components represent DNA strands, DNA gates, and their interactions. We begin with a nondeterministic algebra, and we discuss a stochastic variant in Section 4. Our strand algebras may look very similar to either chemical reactions, or Petri nets, or multiset-rewriting systems. The difference 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 (*strands* and *gates*), and only two combinators: *parallel (concurrent) composition* P|P, and *populations* P*. An inexhaustible population P* has the property that P* = P|P*; that is, there is always one more P that can be taken from the population.

2.1.1 Syntax

The set $\mathcal{P}$ is the set of finite trees P generated by the syntax below. We freely use parentheses when representing these trees linearly as strings. The set $\mathcal{S}$ is a countable set of distinct *strands* x.

2.1–1 Syntax

$P ::= x \vdots [x_1,..,x_n].[x_1',..,x_m'] \vdots 0 \vdots P_1 \mid P_2 \vdots P^*$	$n\geq 1, m\geq 0$	

A *gate* is an operator from strands to strands: $[x_1,..,x_n].[x_1',..,x_m']$ is a gate that binds strands $x_1..x_n$, produces strands $x_1'..x_m'$, and is consumed in the process. We say that this gate *joins* n strands and then *forks* m strands; see some special cases below. An inert component is indicated by 0. Strands 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*.

2.1–2 Explanation of the Syntax and Abbreviations

x		is a *strand*	0	is *inert*
$x_1.x_2$	$\stackrel{\text{def}}{=} [x_1].[x_2]$	is a *sequence gate*	$P \mid P$	is *parallel composition*
$x.[x_1,..,x_m]$	$\stackrel{\text{def}}{=} [x].[x_1,..,x_m]$	is a *fork gate*	P^*	is an *unbounded population*
$[x_1,..,x_n].x$	$\stackrel{\text{def}}{=} [x_1,..,x_n].[x]$	is a *join gate*		

2.1.2 Semantics

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 [2]:

2.1–3 Mixing

$P \equiv P$	equivalence	$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	in context
$P \equiv Q \Rightarrow Q \equiv P$		$P \equiv Q \Rightarrow P^* \equiv Q^*$	
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$			
		$P^* \equiv P^* \mid P$	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 $\rightarrow$.

2.1–4 Reaction

$x_1 \mid .. \mid x_n \mid [x_1,..,x_n].[x_1',..,x_m'] \rightarrow x_1' \mid .. \mid x_m'$	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 strand. Because of the associativity of parallel composition under ≡, and of the well-mixing rule, it does not matter how the left or right hand side of the gate rule associates, although one should assume a fixed association for the formal rule.

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

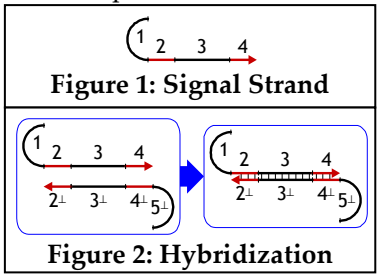

Figure 1: Signal Strand

Figure 2: Hybridization

It is important to note that there is a duality between strands and gates: strands can interact with gates but strands cannot interact with strands, and gates cannot interact with gates. As we shall see, in the DNA implementation the input part of a gate is the Watson-Crick dual of the corresponding strand. This duality need not be exposed in the syntax: it is implicit in the separation between strands and gates, so we use the same x_1 both for the 'positive' free strand and for the complementary 'negative' gate inputs in a reaction like $x_1 \mid x_1.x_2 \rightarrow x_2$.

3 DNA Semantics

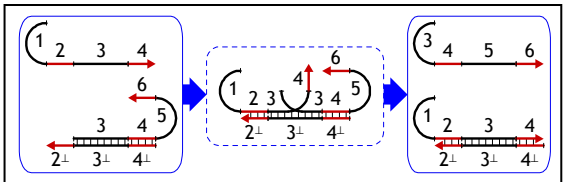
Figure 3: Transducer

In this section we provide a DNA implementation of the combinatorial strand algebra. Given a representation of strands and gates, it is then a simple matter to represent any strand algebra expression as a DNA system, since 0, P|P, and P* are just assemblies of strands and gates.

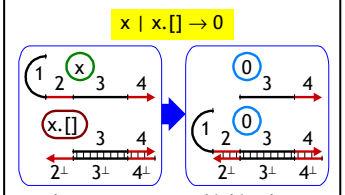

Figure 4: Annihilation

Following [15], we represent a strand x in the strand algebra as a DNA *signal strand* with four regions 1,2,3,4 (Figure 1): 1 = *history*, 2 = *toehold*, 3 = *body*, 4 = *next toehold*. The history is accumulated during previous interactions (and might even by hybridized), and is not part of signal identity: DNA strands with different histories should behave the same. Hence, x denotes an equivalence class of strands with different histories. A toehold is a region that causes interaction with gates.

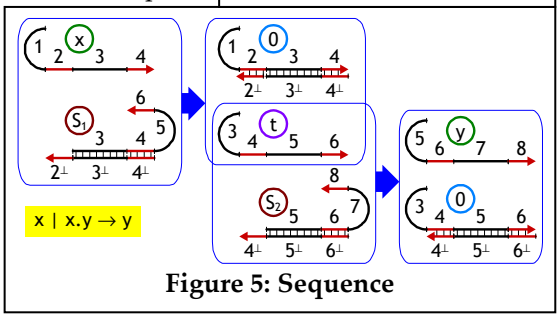

Figure 5: Sequence

A strand $5^\perp,4^\perp,3^\perp,2^\perp = (2,3,4,5)^\perp$ is *Watson-Crick complementary* to 2,3,4,5 and, as in Figure 2, can partially hybridize with 1,2,3,4. For two abstract strands 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 regions [9]. In our algebra we assume that all strands are 'positive', that is, they are encoded in such a way that none is complementary to another: the only 'negative' strands occur in the input region of gates.

The basic signal transduction [15] is shown in Figure 3. This is a reaction that starts from the hybridization of toehold 2 with $2^\perp$. Then it continues with a neutral series of reactions between base pairs (*branch migration* [17]) each going randomly left or right through small exergy hills, and eventually ejecting the 3,4,5,6 strand when the branch migration randomly reaches the right end. The 3,4,5,6 strand can in principle reattach at any point along the 3,4 segment, but it has no toehold to do so easily, and would be ejected again, so this reaction is considered irreversible. Moreover, the toehold 4 has become free, and this toehold with its contiguous branch migration segment 5 can initiate another reaction (while the 4 segment in 1,2,3,4 could not).

The simple-minded interpretation of transduction is then that the strand 1,2,3,4 is removed, and the strand 3,4,5,6 is released irreversibly. The transducer itself is consumed during this process, leaving an inert residual (assuming that the history segment 1 has no activity).

Figure 4 shows the simplest gate: it inputs a strand and produces nothing that has any significant activity. In Figure 5 we implement an x.y gate as the serial composition of two transducers. The result is a sequence gate that can transform any signal x into any other signal y, through a temporary strand t that connects them. The t strand overlaps with x in 3,4 and overlaps with y in 5,6, but the strands for x and y are independent. Note that in a stochastic algebra a step like x.y would be understood on the basis of an exponential distribution, while the implementation makes two steps, and hence produces an Erlang distribution. This is a common issue in DNA encodings [15].

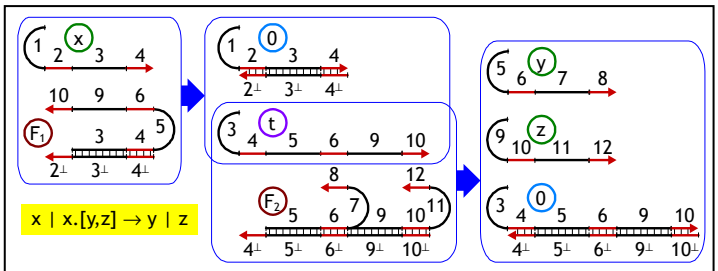

Figure 6: 2-Way Fork

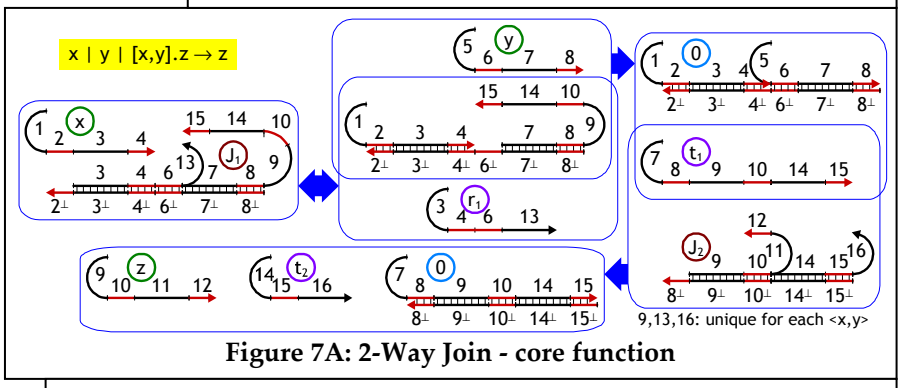

Figure 7A: 2-Way Join - core function

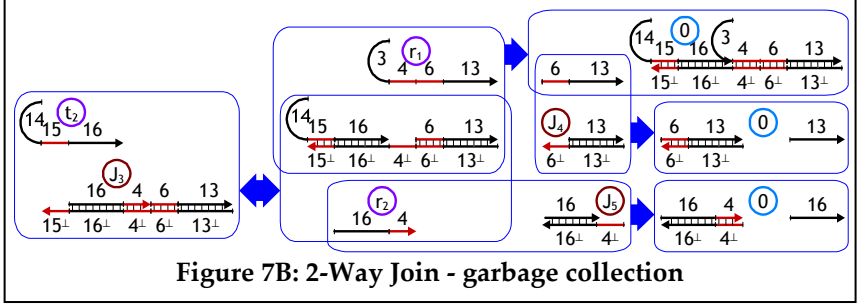

Figure 7B: 2-Way Join - garbage collection

In Figure 6 (Figure 2 in [15]), we generalize this schema to produce two outputs by releasing two strands from the second transducer. The temp strand is tailored to the task, and in particular it does not have the structure of a normal signal strand, but the inputs and outputs do. Therefore, the combined structures F_1,F_2 give us an implementation of a 2-way fork gate, x.[y,z]; the idea can be easily generalized to n-way forks.

Many designs are being investigated for join [5]. The solution shown here admits the composition of joins with the same inputs, [x,y].z|[x,y].z', without crosstalk or preprocessing of the system. More economical solutions are known, assuming compiler support. Figure 7 implements a binary join operator by the composition of two 'reversible-AND' gates [15] and an intermediate fork. 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 of chemical reversibility. Given two inputs x,y, the reversible-AND gate J_1 produces a temporary output t_1, with strand r_1 providing reversibility while waiting for the second input (Figure 7A, from Figure 3 in [15]). Since t_1 has an overlap with y, we then need a further step to convert it to the desired independent output z via J_2. In the next phase (Figure 7B) we use another reversible-AND gate J_3 with inputs t_2 and r_1 to remove r_1 from the system, so that r_1 does not accumulate to slow down further join operations; this garbage collection phase happens off the critical path. Note that r_1 is eliminated only when the reversibility of J_1 is no longer needed, while the strand r_2, providing reversibility for J_3 can be eliminated right away via J_5, because by construction J_3 has both inputs available. The join gate itself is given by the structures $J_1..J_5$. The domains 9,13,16, which do not appear in the recognition regions of x,y,z, are independent parameter and are used as gate fingerprints; for example, 9 is chosen different for each different ⟨x,y⟩ pair so that J_1 and J_2 are connected without crosstalk.

This technique can be generalized to n-way join gates (logically firing only when all n inputs are available) by cascading reversible $r_1..r_{n-1}$ strands on the same J_1 backbone before committing irreversibly to the t_1 output. Alternatively, the reversibility idea suggests a way to implement a 3-way join from a 2-way join and an extra strand x_0, although this encoding 'costs' a population: $[x_1,x_2,x_3].x_4 \stackrel{def}{=} ([x_1,x_2].x_0 \mid x_0.[x_1,x_2])^* \mid [x_0,x_3].x_4$.

For the purposes of the next section, note that by adding to the $[x_1,x_2].x_3$ gate of Fig 7A a component that complements and removes r_1, thus blocking the reversibility of the first reaction, yields a 2-way *sequence* gate $x_1.x_2.x_3$ with reactions $x_1 \mid x_2 \mid x_1.x_2.x_3 \rightarrow x_2 \mid x_2.x_3 \rightarrow x_3$ (up to some initial binding and unbinding of x_1); the garbage collection of r_1 in Figure 7B is then irrelevant. Similarly, blocking the reversibility of the first reaction on a 3-way join yields a gate $x_1.[x_2,x_3].x_4$ with reactions $x_1 \mid x_2 \mid x_3 \mid x_1.[x_2,x_3].x_4 \rightarrow x_2 \mid x_3 \mid [x_2,x_3].x_4 \rightarrow x_4$. With a large enough number of blocking components one can ensure that the first reaction essentially never reverses, thus implementing purely sequential gates. Other implementations of such sequential gates are also possible.

4 Stochastic Strand Algebra

Stochastic strand algebra is obtained by assigning stochastic rates to gates, and by dropping the unbounded populations, P*. Since the binding strengths of toeholds of the same length are comparable [16], 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. Although gate rates are fixed, we can vary population sizes in order to achieve desired macroscopic rates. By the fingerprinting technique from Section 3 we can even have populations of different sizes that respond separately to the same inputs. And, as we describe below, it is possible to maintain stable population sizes, and hence to achieve desired stable rate ratios.

In this section $[x_1,..,x_n].[y_1,..,y_m]$ is a stochastic gate of rate g_n, and we write P^k for k parallel copies of P. In a global system state P, the *propensity* of a gate reaction is $(P \textit{ choose } (x_1 \mid .. \mid x_n \mid [x_1,..,x_n].[y_1,..,y_m]))\times g_n$; that is, the gate rate g_n multiplied by the number of ways of choosing out of P a multiset consisting of a gate and its n inputs. For example, if $P = x^n \mid y^m \mid ([x,y].z)^p$ with $x \neq y$, then the propensity of the first reaction in P is $n\times m\times p\times g_2$. A *global transition* from a global state P to a next global state, labeled with its propensity, has then the following form, where \ is multiset difference:

$$P \rightarrow^{(P \textit{ choose } (x_1 \mid .. \mid x_n \mid [x_1,..,x_n].[y_1,..,y_m]))\times g_n} P \setminus (x_1 \mid .. \mid x_n \mid [x_1,..,x_n].[y_1,..,y_m])) \mid y_1 \mid ... \mid y_m$$

The collection of all global transitions from P and from its successive global states forms a labeled transition graph, from which one can extract the Continuous Time Markov Chain of the system [4].

The stochastic algebra needs in general additional gate primitives, because the encoding of a complex gates into simpler gates (which is what keeps the algebra of Section 2 minimal) may not preserve stochastic behavior. We shall soon need a gate of the form $x_0.[x_1,..,x_n].[y_1,..,y_m]$, whose DNA structure and reactions are discussed at the end of Section 3, and whose global transitions are:

$$P \rightarrow^{(P \textit{ choose } (x_0 \mid x_0.[x_1,..,x_n].[y_1,..,y_m]))\times g_1} P \setminus (x_0 \mid x_0.[x_1,..,x_n].[y_1,..,y_m])) \mid [x_1,..,x_n].[y_1,..,y_m]$$

In a stochastic system, an unbounded population like P* has little meaning because its rates are unbounded as well. In stochastic strand algebra we simply drop the P* construct. In doing so, however, we eliminate the main mechanism for iteration and recursion, and we need to find an alternative mechanism. Rather than P*, we should instead consider finite populations P^k exerting a stochastic pressure given by the size k. It is also interesting to consider finite populations that *remain* at constant size k: let's indicate them by $P^{=k}$. In particular, $P^{=1}$ represents a single catalyst molecule.

We now show that we can model populations of constant size k by using a bigger buffer population to keep a smaller population at a constant level. Take, for example, P = [x,y].z, and define:

$$P^{=k} \stackrel{def}{=} [x,y].[z,X]^k \mid (X.[x,y].[z,X])^{f(k)} \qquad \text{for a fresh strand } X$$

Here $f(k)$ is the size of a large-enough buffer population. A global transition of $P^{=k}$ in context Q (with Q not containing other copies of those gates) is $(Q|P^{=k}) \rightarrow^{((Q|P^{=k}) \ choose \ (x|y|[x,y].[z,X]))\times g_2} (Q\backslash(x|y) \mid [x,y].[z,X]^{k-1} \mid z \mid X \mid (X.[x,y].[z,X])^{f(k)})$. For a large enough $f(k)$, the propensity of a next reaction on gate X.[x,y].[z,X] can be made arbitrarily large, so that the two global transitions combined approximate $(Q|P^{=k}) \rightarrow^{((Q|P^{=k}) \text{ choose } (x|y|[x,y].[z,X]))\times g_2} (Q\backslash(x|y) \mid [x,y].[z,X]^{k} \mid z \mid (X.[x,y].[z,X])^{f(k)-1})$, where the gate population is restored at level k, and the buffer population decreases by 1. We have shown that the reaction propensity in $(Q|P^{=k})$ can be made arbitrarily close to the reaction propensity in $(Q|P^{k})$, but with the gate population being restored to size k. Moreover, it is possible to periodically replenish the buffer by external intervention *without disturbing the system* (except for the arbitrarily fast reaction speed on X). This provides a practical way of implementing recursion and unbounded computation, by 'topping-up' the buffer populations, without a notion of unbounded population.

It is possible to formulate a formal translation from the stochastic strand algebra to the chemical algebra of Introduction, by following the figures of Section 3 (considering branch migration as a single reaction). This chemical semantics of the strand algebra does not exactly match the global transition semantics given above, because for example a single reaction $x|x.y \rightarrow y$ is modeled by two chemical reactions; however, as in [15], it can approximate it.

5 Compiling to Strand Algebra

We give examples of translating other formal languages to strand algebra, in particular translating interacting automata. The interesting point is that by these translations we can map all those formal languages to DNA, by the methods in Section 3.

5.1.1 Finite Stochastic Reaction Networks

We summarize the idea of [15], which shows how to encode with approximate dynamics a stochastic chemical system as a set of DNA strands and gates. A unary reaction $A \rightarrow C_1+..+C_n$ is represented as $(A.[C_1,..,C_n])^*$. A binary reaction $A+B \rightarrow C_1+..+C_n$ is represented as $([A,B].[C_1,..,C_n])^*$. The initial solution, e.g. A+A+B, is represented as $A|A|B$ and composed with the populations representing the reactions. For stochastic chemistry, one must replace the unbounded populations with large but finite populations whose sizes and rates are calibrated to provide the desired chemical rates. Because of technical constraints on realizing the rates, one may have to preprocess the system of reactions [15].

5.1.2 Petri Nets

Consider a place-transition net with places x_i; then, a transition with incoming arcs from places $x_1..x_n$ and outgoing arcs to places $x_1'..x_m'$ is represented as $([x_1,..,x_n].[x_1',..,x_m'])^*$. The initial marking $\{x_1, .., x_k\}$ is represented as $x_1|..|x_k$. The idea is obviously similar to the translation of chemical networks, because those can be represented as (stochastic) Petri nets. Conversely (thanks to Cosimo Laneve for pointing this out), a strand can be represented as a marked place in a Petri net, and a gate $[x_1,..,x_n].[x_1',..,x_m']$ as a transition with an additional marked 'trigger' place on the input that makes it fire only once; then, P* can be represented by connecting the transitions of P to refresh the trigger places. Therefore, strand algebra is equivalent to Petri nets. Still, the algebra provides a compositional language for describing such nets, where the gates/transitions are consumed resources.

5.1.3 Finite State Automata and Transducers

A state in an FSA is represented as a strand X. The transition matrix of the FSA is represented as a set of terms $([X,x].X')^*$ in parallel, where X is the current state, x is from the input alphabet, and X' is the next state. For multiple transitions from the same state there will be multiple occurrences with the same X and different x_i. For nondeterministic transitions there will be multiple occurrences of the same X and x. The initial state X_0 is placed in parallel with those terms. For a transducer, the terms

have the form ([X,x].[X′,x′])*, where x′ is taken from the output alphabet. An encoding of input strings $x_1.x_2.x_3...$ as sequential inputs can be obtained by cascading binary joins.

5.1.4 Interacting Automata

Interacting automata [4] are finite state automata that interact with each other over synchronous stochastic channels (that is, they 'collide' pairwise and change states); see [3] for many examples. They can be faithfully emulated in stochastic strand algebra by generating a binary join gate for each possible collision, and by choosing stable population sizes that produce the prescribed rates. The translation can cause an n^2 expansion of the representation [4].

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* of the form $\pi_1;P_1 \oplus \ldots \oplus \pi_n;P_n$, where $\oplus$ is stochastic choice, P_i are multisets of resulting species, and π_i are either delays τ_r, inputs $?c_r$, or outputs $!c_r$ on a channel c at rate r. Complementary $!c_r$ / $?c_r$ prefixes specify collisions at rate r. For example, in E_1 an A automaton can collide with a B automaton on channel a, resulting in two A automata:

E_1: $A = !a_r.A \oplus ?b_s.B$, $B = !b_s.B \oplus ?a_r.A$ $\qquad$ E_2: $A = !a_r.A \oplus ?a_r.B$, $B = !b_s.B \oplus ?b_s.A$

With initial conditions $A^n|B^m$ (that is, n automata in state A and m in state B), the Continuous Time Markov Chain semantics of [4] prescribes the propensities for the transitions out of $A^n|B^m$ (in E_2 there are two ?/! ways for A to collide with A, hence the rate there is $2\times(n \text{ choose } 2)\times r = n\times(n-1)\times r$):

E_1: by a: $A^n|B^m \rightarrow^{n\times m\times r} A^{n+1}|B^{m-1}$; by b: $A^n|B^m \rightarrow^{n\times m\times s} A^{n-1}|B^{m+1}$ $\qquad$ E_2: by a: $A^n|B^m \rightarrow^{n\times(n-1)\times r} A^{n-1}|B^{m+1}$; by b: $A^n|B^m \rightarrow^{m\times(m-1)\times s} A^{n+1}|B^{m-1}$

Subsequent transitions are computed in the same way.

The translation of interacting automata to strand algebra is as follows. E.X.i denotes the i-th summand of the molecule associated to X in E; ⟪...⟫ and ∪ denote multisets and multiset union to correctly account for multiplicity of interactions; and *Parallel*(S) is the parallel composition of the elements of multiset S. *Strand*(E) is then the translation of a system of equations E, using the stable buffered populations $P^{=k}$ described in Section 4:

$$\begin{aligned} \mathit{Strand}(E) = \mathit{Parallel}(& \langle\!\langle\ (X.[P])^{=r/g_1}\ \mathit{s.t.}\ \exists i.\ E.X.i = \tau_r;P\ \rangle\!\rangle \cup \\ & \langle\!\langle\ ([X,Y].[P,Q])^{=r/g_2}\ \ \mathit{s.t.}\ X{\neq}Y \text{ and } \exists i,j,c.\ E.X.i = ?c_r;P \text{ and } E.Y.j = !c_r;Q\ \rangle\!\rangle \cup \\ & \langle\!\langle\ ([X,X].[P,Q])^{=2r/g_2}\ \ \mathit{s.t.}\ \exists i,j,c.\ E.X.i = ?c_r;P \text{ and } E.X.j = !c_r;Q\ \rangle\!\rangle\ \ ) \end{aligned}$$

The E_1,E_2 examples above, in particular, translate as follows:

S_1: $([B,A].[A,A])^{=r/g_2}\ |\ ([A,B].[B,B])^{=s/g_2}$ $\qquad$ S_2: $([A,A].[B,A])^{=2r/g_2}\ |\ ([B,B].[A,B])^{=2s/g_2}$

Initial automata states are translated identically to initial strands and placed in parallel. As described in Section 4, a strand algebra transition from global state $A^n|B^m|([A,B].[C,D])^{=p}$ has propensity $n\times m\times p\times g_2$, and from $A^n|([A,A].[C,D])^{=p}$ has propensity (n *choose* 2)$\times p\times g_2$. From the same initial conditions $A^n|B^m$ as in the automata, we then obtain the global transitions:

$A^n|B^m|S_1 \rightarrow^{n\times m\times r/g_2\times g_2} A^{n+1}|B^{m-1}|S_1'$ $\qquad$ $A^n|B^m|S_2 \rightarrow^{(n\times(n-1))/2\times 2r/g_2\times g_2} A^{n-1}|B^{m+1}|S_2'$

$A^n|B^m|S_1 \rightarrow^{n\times m\times s/g_2\times g_2} A^{n-1}|B^{m+1}|S_1''$ $\qquad$ $A^n|B^m|S_2 \rightarrow^{(m\times(m-1))/2\times 2s/g_2\times g_2} A^{n+1}|B^{m-1}|S_2''$

which are equivalent to the interacting automata transitions. Here S_i',S_i'' are systems where a buffer has lost one element, but where the active gate populations remain at the same level as in S_i. We have shown that the stochastic behavior of interacting automata is preserved by their translation to strand algebra, assuming that the buffers are not depleted.

6 Nested Strand Algebra

The purpose of this section is to allow nesting of join/fork operators in strand algebra, so that natural compound expressions can be written. We provide a uniform translation of this extended language back to $\boldsymbol{\mathcal{P}}$, as a paradigm for the compilation of high(er) 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:

$$?x_1.!x_2.?x_3 \quad \stackrel{\text{def}}{=} \quad x_1.[x_2,x_0] \mid [x_0,x_3].[]$$

where the right hand side is a set of $\boldsymbol{\mathcal{P}}$ 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$\boldsymbol{\mathcal{P}}$ admits such nesting of operators in general. The main change from the combinatorial $\boldsymbol{\mathcal{P}}$ 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:

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

The mixing relation is the same as in $\boldsymbol{\mathcal{P}}$. The reaction relation is modified only in the gate rule:

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

We now show how to compile n$\boldsymbol{\mathcal{P}}$ to $\boldsymbol{\mathcal{P}}$. Let $\mathcal{X}$ be an infinite lists of distinct strands, and $\mathfrak{F}$ be the set of such $\mathcal{X}$'s. Let $\mathcal{X}_i$ be the i-th strand in the list, $\mathcal{X}_{\geq i}$ be the list starting at the i-th position of $\mathcal{X}$, $evn(\mathcal{X})$ be the even elements of $\mathcal{X}$, and $odd(\mathcal{X})$ be the odd elements. Let $\mathfrak{F}_P$ be the set of those $\mathcal{X} \in \mathfrak{F}$ that do not contain any strand that occurs in P. The unnest algorithm $U(P)_{\mathcal{X}}$, for $P \in n\boldsymbol{\mathcal{P}}$ and $\mathcal{X} \in \mathfrak{F}_P$, is shown in Table 9.1–1. The inner loop $U(X,P)_{\mathcal{X}}$ uses X as the trigger for the translation of P.

6.1–1 Unnest Algorithm

$U(P)_{\mathcal{X}}$	$\stackrel{\text{def}}{=}$	$\mathcal{X}_0 \mid U(\mathcal{X}_0,P)_{\mathcal{X}_{\geq 1}}$
$U(X, x)_{\mathcal{X}}$	$\stackrel{\text{def}}{=}$	$X.x$
$U(X, ?[x_1,..,x_n].P)_{\mathcal{X}}$	$\stackrel{\text{def}}{=}$	$[X,x_1,..,x_n].\mathcal{X}_0 \mid U(\mathcal{X}_0,P)_{\mathcal{X}_{\geq 1}}$
$U(X, ![x_1,..,x_n].P)_{\mathcal{X}}$	$\stackrel{\text{def}}{=}$	$X.[x_1,..,x_n,\mathcal{X}_0] \mid U(\mathcal{X}_0,P)_{\mathcal{X}_{\geq 1}}$
$U(X, 0)_{\mathcal{X}}$	$\stackrel{\text{def}}{=}$	$X.[]$
$U(X, P' \mid P'')_{\mathcal{X}}$	$\stackrel{\text{def}}{=}$	$X.[\mathcal{X}_0,\mathcal{X}_1] \mid U(\mathcal{X}_0,P')_{evn(\mathcal{X}_{\geq 2})} \mid U(\mathcal{X}_1,P'')_{odd(\mathcal{X}_{\geq 2})}$
$U(X, P^*)_{\mathcal{X}}$	$\stackrel{\text{def}}{=}$	$(X.[\mathcal{X}_0,X] \mid U(\mathcal{X}_0,P)_{\mathcal{X}_{\geq 1}})^*$

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

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

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 $U(X,P^*)_{\mathcal{X}}$ to produce $(X.[\mathcal{X}_0,X,X] \mid U(\mathcal{X}_0,P')_{\mathcal{X}_{\geq 1}})^*$.

In the nested algebra we can also easily solve systems of recursive definitions; for example: 'X = $(?x_1.X \mid !x_2.Y)$ and Y = $?x_3.(X \mid Y)$' can be written as: '$(?X.(?x_1.X \mid !x_2.Y))^* \mid (?Y.?x_3.(X \mid Y))^*$'.

7 Contributions and Conclusions

We have introduced strand algebra, a formal language based on a simple relational semantics that is equivalent to place-transition Petri nets (in the current formulation), but allows for compositional de-

scriptions where each component maps directly to DNA structures. Strand algebra connects a simple but powerful class of DNA system to a rich set of techniques from process algebra for studying concurrent systems. Within this framework, it is easy to add operators for new DNA structures, or to map existing operators to alternative DNA implementations. We show how to use strand algebra as an intermediate compilation language, by giving a translation from a more convenient syntax. We also describe a stochastic variant, and a technique for maintaining stable buffered populations to support indefinite and unperturbed stochastic computation.

Using strand algebra as a stepping stone, we describe a DNA implementation of interacting automata that preserves stochastic behavior. Interacting automata (a stochastic subset of CCS [10]) are about the simplest process algebra in the literature. Hopefully, more advanced features of process algebra 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.

I would like to acknowledge the Molecular Programming groups at Caltech for invaluable discussions and corrections. In particular, the join gate design of Figure 7 evolved through extended discussions with David Soloveichik and Erik Winfree.

References

[1] Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, E. Shapiro. **Programmable and Autonomous Computing Machine made of Biomolecules.** Nature, Vol 414, 22 November 2001.

[2] G. Berry, G. Boudol. **The Chemical Abstract Machine**. Proc. 17th POPL, ACM, 81-94, 1989.

[3] L. Cardelli: **Artificial Biochemistry**. In: A. Condon, D. Harel, J.N. Kok, A. Salomaa, E.Winfree (Eds.). Algorithmic Bioprocesses. Springer, April 2009. <http://lucacardelli.name/Papers/Artificial%20Biochemistry.pdf>

[4] L. Cardelli: **On Process Rate Semantics**. Theoretical Computer Science 391(3) 190-215, 2008.

[5] L. Cardelli, L. Qian, D. Soloveichik, E. Winfree. Personal communications.

[6] V. Danos, C. Laneve. **Formal molecular biology**. Theoretical Computer Science 325(1) 69-110. 2004.

[7] 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.

[8] M. Hagiya. **Towards Molecular Programming.** In G. Ciobanu, G. Rozenberg, (Eds.) Modelling in Molecular Biology. Springer 2004.

[9] L. Kari, S. Konstantinidis, P. Sosík. **On Properties of Bond-free DNA Languages**. Theoretical Computer Science 334(1-3), 131-159, 2005.

[10] R. Milner. **Communicating and Mobile Systems: The π-Calculus**. Cambridge University Press, 1999.

[11] L. Qian, E. Winfree. **A Simple DNA Gate Motif for Synthesizing Large-scale Circuits.** Proc. 14th International Meeting on DNA Computing. 2008.

[12] 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.

[13] 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.

[14] G. Seelig, D. Soloveichik, D.Y. Zhang, E. Winfree. **Enzyme-Free Nucleic Acid Logic Circuits.** Science, Vol 314, 8 December 2006.

[15] D. Soloveichik, G. Seelig, E. Winfree. **DNA as a Universal Substrate for Chemical Kinetics** Proc. DNA14.

[16] P. Yin, H.M.T. Choi, C.R. Calvert, N.A. Pierce. **Programming Biomolecular Self-assembly Pathways.** Nature, 451:318-322, 2008.

[17] B. Yurke, A.P. Mills Jr. **Using DNA to Power Nanostructures**, Genetic Programming and Evolvable Machines archive 4(2), 111 - 122, Kluwer, 2003.

[18] D. Y. Zhang, A. J. Turberfield, B. Yurke, E. Winfree. **Engineering Entropy-driven Reactions and Networks Catalyzed by DNA**. Science, 318:1121-1125, 2007.