Strand Algebras for DNA Computing

Luca Cardelli

Microsoft Research

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.

1 Introduction

DNA technology is reaching the point where one can envision automatically compiling high-level formalisms to DNA computational structures [18]. Examples so far include the 'manual compilation' of automata and Boolean networks, where some impressive demonstrations have been carried out [1][8][15][16]. 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 [17]: 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. In DNA we can instead implement a *join* primitive, based on [17]: this is a powerful operator, widely studied in concurrency theory [7][13], which can indirectly provide an implementation of external choice. The DNA algebra supporting the translation is built around the 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 [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 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. It is also useful to consider the symmetric and transitive closure, $\to^*$, 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 $\to$ describing whole-system transi-

tions, 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
$P \equiv Q \Rightarrow P + R \equiv Q + R$	in context
$P + Q \equiv Q + P;\quad P + (Q + R) \equiv (P + Q) + R;\quad P + 0 \equiv P$	diffusion
$P \to Q \Rightarrow P + R \to Q + R$	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 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 [11]. 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 $\to$ 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 *bio*chemistry more directly [6][14] or, as in this paper, that can suit DNA computing.

2 Strand Algebras

By a *strand algebra* we mean a process algebra [11] 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 (*signals* and *gates*), and only two combinators: *parallel (concurrent) composition* P | Q, 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.

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 are taken from a countable set.

2.1–1 Syntax

$P ::= x \mid [x_1,..,x_n].[x'_1,..,x'_m] \mid 0 \mid P_1 \mid P_2 \mid P^*$	$n\geq 1,\ m\geq 0$

A *gate* is an operator from signals to signals: $[x_1,..,x_n].[x'_1,..,x'_m]$ is a gate that binds signals $x_1..x_n$, produces signals $x'_1,..,x'_m$, 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*.

2.1–2 Explanation of the Syntax and Abbreviations

x		is a *signal*	0	is *inert*
$x_1.x_2$	$\stackrel{def}{=} [x_1].[x_2]$	is a *transducer gate*	$P_1 \mid P_2$	is *parallel composition*
$x.[x_1,..,x_m]$	$\stackrel{def}{=} [x].[x_1,..,x_m]$	is a *fork gate*	P*	is an *unbounded population*
$[x_1,..,x_n].x$	$\stackrel{def}{=} [x_1,..,x_n].[x]$	is a *join gate*		

The relation $\equiv \subseteq \mathcal{P}x\mathcal{P}$, 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 $\rightarrow \subseteq \mathcal{P}x\mathcal{P}$, 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 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 $\rightarrow x_3 \mid x_1.x_2$
$X \mid ([X,x_1].[x_2,X])^*$ a catalytic system ready to transform multiple x_1 to x_2, with catalyst X

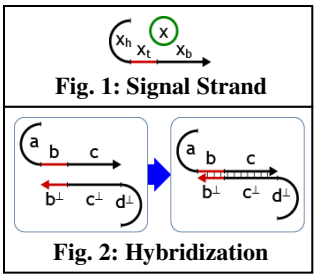

Fig. 1: Signal Strand

Fig. 2: Hybridization

There is a duality between signals and gates: 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. This 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

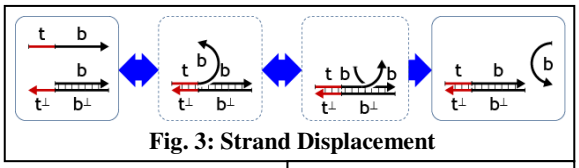

Fig. 3: Strand Displacement

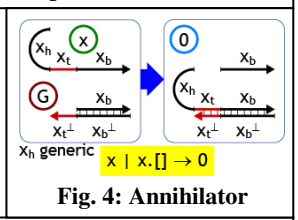

Fig. 4: Annihilator

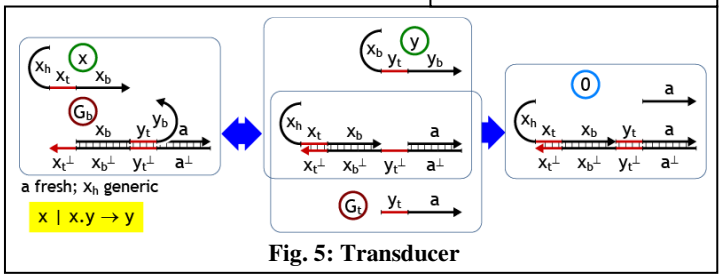

Fig. 5: Transducer

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.

There are many possible 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 [17], where DNA computation is based on strand displacement on loop-free structures. Other architectures are possible, like computation with hairpins [18], but have not been fully worked out. The four-segment signal structure in [17] yields a full implementation of the combinatorial strand algebra (not shown, 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 segments x_h,x_t,x_b (Figure 1): x_h = *history*, x_t = *toehold*, x_b = *binding*. A toehold is a segment that can reversibly interact with a gate: the interaction can then propagate to the adjacent binding segment. 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 *segments* (which are single-stranded sequences of bases).

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 segments. 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 segments [9][10]. We assume that all signals are made of 'positive' strands, with 'negative' strands occurring only in gates, and in particular in their input segments; this separation enables the use of 3-letter codes, that helps design independent sequences [10][20].

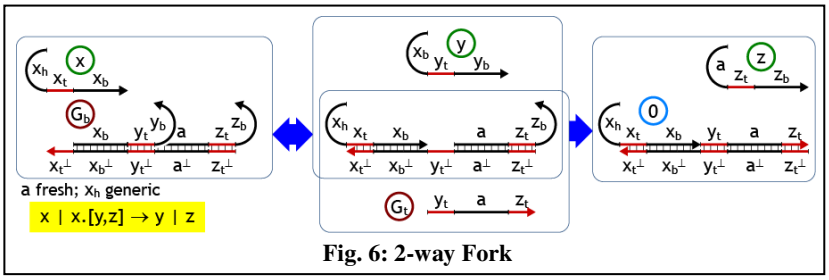

Fig. 6: 2-way Fork

The basic computational step of *strand displacement* [17] 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 segment b by a neutral series of reactions between base pairs (*branch migration* [19]) each going randomly left or right through small exergy 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 a,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. 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 reac-

tion eventually 'locks' the gate in the state where x is consumed and y is produced. The annotation 'a fresh' means that the segment '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 segment). In general, we take all gate segments to be fresh unless they are non-history segments 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. This is a common issue in DNA encodings, but its impact can be minimized [17], e.g. in this case by using a large G_t population. In Figure 6 (*cf.* Figure 2 in [17]), 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.

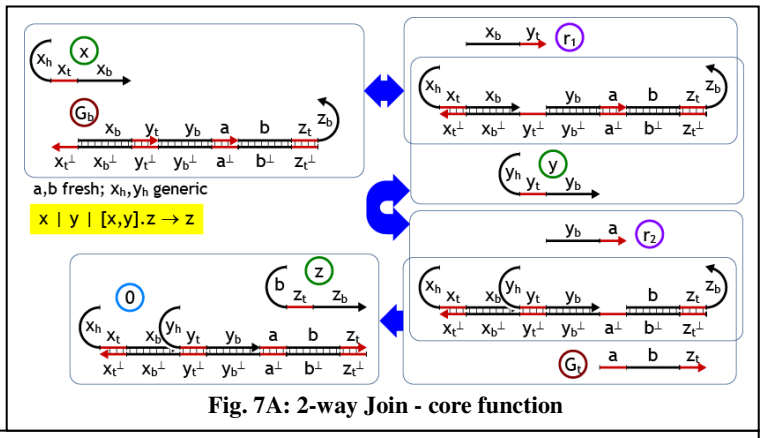

Fig. 7A: 2-way Join - core function

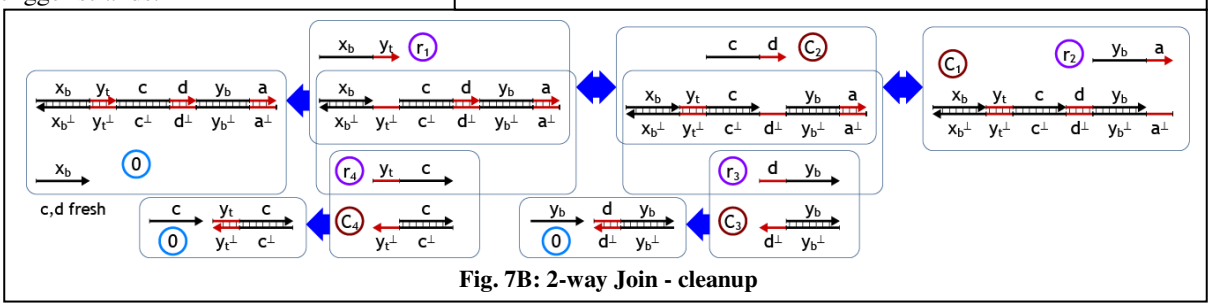

Fig. 7B: 2-way Join - cleanup

Many designs have been investigated for join gates [5]. The solution shown in Figure 7 admits the coexistence of joins with the same inputs, [x,y].z | [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 of chemical reversibility. Given two inputs x,y, a 'reversible-AND' G_b backbone releases two strands r_1,r_2, with r_1 providing reversibility while waiting for y (*cf.* Figure 3 in [17]); the trigger G_t finally irreversibly releases the output z (or outputs). In a cleanup phase (Figure 7B), off the critical path, we use a similar reversible-AND 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 reversible-ANDs. (Note that without the extra c,d segments, a strand y_t,y_b = y would be released.) This gate structure can be easily generalized to 3-way and higher join gates by cascading more inputs on the G_b backbone. Alternatively, we can implement a 3-way join from 2-way joins and an extra signal x_0, but this encoding 'costs' a population: $[x_1,x_2,x_3].x_4 \stackrel{\text{def}}{=} ([x_1,x_2].x_0 \mid x_0.[x_1,x_2])^* \mid [x_0,x_3].x_4$.

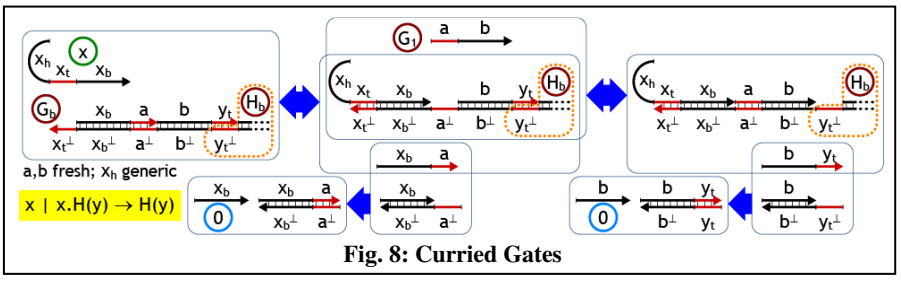

Fig. 8: Curried Gates

This completes the implementation of strand algebra in DNA. For the purposes of the next section, however, it is useful to consider also *curried gates* (gates that produce gates). Figure 8 shows a gate x.H(y) that accepts a signal x and activates the backbone H_b of 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 | x.y.z → y.z. (The extra a,b segments prevent the release of a strand x_b,y_t that would interfere with r_1 of [x,y].z; see Figure 7A.) This implies that there is an extension of strand algebra with gates of the form $G ::= [x_1,..,x_n].[x'_1,..,x'_m] \vdots [x_1,..,x_n].G$; this extension can be translated back to the basic strand algebra, e.g. by setting x.y.z = x.w | [w,y].z for a fresh w, but a direct implementation of curried gates is also available.

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 [18], 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. Moreover, 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:

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

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]. We shall soon use also a curried gate of the form x.G, whose DNA structure is discussed in Section 3, and whose global transitions are:

$$P \quad \rightarrow^{(P \textit{ choose } (x \mid x.G)) \times g_1} \quad P \backslash (x \mid x.G) \mid G$$

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:

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

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 $(Q \mid P^{=k}) \rightarrow^{((Q \mid P^{=k}) \textit{ choose } (x|y|[x,y].[z,X])) \times g_2} (Q \backslash (x \mid y) \mid ([x,y].[z,X])^{k-1} \mid z \mid X \mid (X.[x,y].[z,X])^{f(k)})$. 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 $(Q \mid P^{=k}) \rightarrow^{((Q \mid P^{=k}) \text{ choose } (x|y|[x,y].[z,X])) \times g_2} (Q \backslash (x \mid y) \mid ([x,y].[z,X])^k \mid z \mid (X.[x,y].[z,X])^{f(k)-1})$, where the gate population is restored at level k, and the buffer population decreases by 1. We have shown that the reaction propensity in $(Q \mid P^{=k})$ can be made arbitrarily close to the reaction propensity in $(Q \mid P^k)$, 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. The construction of a stable population $([x,y].z)^{=k}$ can be carried out also in the basic stochastic algebra without curried gates, but it then requires balancing the rate of a ternary gate against the desired rate of a binary gate.

We should note that the stochastic strand algebra is a convenient abstraction, but the correspondence with the DNA semantics of Section 3 is not direct. More precisely, 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 strand displacement as a single reaction). Such a chemical semantics does not exactly match the global transition semantics given above, because for example a single reaction $x \mid x.y \rightarrow y$ is modeled by two chemical reactions. It is possible to define a chemical semantics that approximates the global transition semantics, by using the techniques discussed in [17], but this topic requires more attention that we can provide here.

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 [17], which shows how to encode with approximate dynamics a stochastic chemical system as a set of DNA signals 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 $A \mid A \mid B$ 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 [17].

5.1.2 Petri Nets

Consider a place-transition Petri Net [13] 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 $x_1 \mid .. \mid x_k$. The idea is similar to the translation of chemical networks: those can be represented as (stochastic) Petri nets. Conversely (thanks to Cosimo Laneve for pointing this out), a signal 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

We assume a single copy of the FSA and of the input string. An FSA state is represented as a signal X. The transition matrix is represented as a set of terms $([X,x].[X',\tau])^*$ in parallel, where X is the current state, x is from the input alphabet, X' is the next state, and τ is a fixed signal used to synchronize with the input string. For nondeterministic transitions there will be multiple occurrences of the same X and x. The initial state $X_0 \mid \tau$ is placed in parallel with those terms. An input string $x_1,x_2,x_3...$ is then encoded as $\tau.[x_1,y_1] \mid [y_1,\tau].[x_2,y_2] \mid [y_2,\tau].[x_3,y_3] \mid ...$ for fresh $y_1,y_2,y_3...$.

5.1.4 Interacting Automata

Interacting automata [4] (a stochastic subset of CCS [11]) 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 ($!c_r$). Intuitively, these automata 'collide' pairwise on complementary exposed surfaces (channels) and change states as a result of the collision. Figure 9 shows two such automata, where each diagram represents a population of identical automata interacting with each other and with other populations (see [3] for many examples). Interacting automata 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

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 a channel c at rate r. For example, in an E_1 population, an automaton in state A_1 can collide by $!a_r$ with an automaton in state B_1 by $?a_r$, resulting in two automata in state A_1:

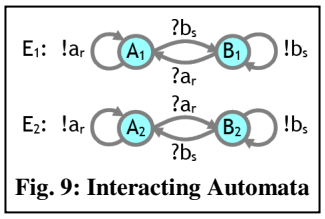

Fig. 9: Interacting Automata

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

With initial conditions $A_i^n \mid B_i^m$ (that is, n automata in state A_i and m in state B_i), the Continuous Time Markov Chain semantics of [4] prescribes the *propensities* of the interactions. On channel a_r, in E_1 the propensity is n×m×r, while in E_2, with two symmetric ?/! ways for A_2 to collide with A_2, the propensity is 2×(n choose 2)×r = n×(n-1)×r:

$$A_1^n \mid B_1^m : (\text{on } a_r) \quad A_1^n \mid B_1^m \rightarrow^{n\times m\times r} A_1^{n+1} \mid B_1^{m-1} \qquad A_2^n \mid B_2^m : (\text{on } a_r) \quad A_2^n \mid B_2^m \rightarrow^{n\times(n-1)\times r} A_2^{n-1} \mid B_2^{m+1}$$
$$(\text{on } b_s) \quad A_1^n \mid B_1^m \rightarrow^{n\times m\times s} A_1^{n-1} \mid B_1^{m+1} \qquad (\text{on } b_s) \quad A_2^n \mid B_2^m \rightarrow^{m\times(m-1)\times s} A_2^{n+1} \mid B_2^{m-1}$$

Subsequent transitions are computed in the same way. One can also mix E_1,E_2 populations (not shown).

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, where g_i are the gate rates of i-ary gates (we assume for simplicity that the round-off errors in r/g_i are not significant and that $r/g_i \geq 1$; otherwise one should appropriately scale the rates r of the original system):

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

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

$$P_1 = \textit{Strand}(E_1) = ([B_1,A_1].[A_1,A_1])^{=r/g_2} \mid ([A_1,B_1].[B_1,B_1])^{=s/g_2} \qquad P_2 = \textit{Strand}(E_2) = ([A_2,A_2].[B_2,A_2])^{=2r/g_2} \mid ([B_2,B_2].[A_2,B_2])^{=2s/g_2}$$

Initial automata states are translated identically into initial signals and placed in parallel. As described in Section 4, 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)×p×g_2. From the same initial conditions $A^n \mid B^m$ as in the automata, we then obtain the global strand algebra transitions:

$$A_1^n \mid B_1^m \mid P_1 \rightarrow^{n\times m\times r/g_2\times g_2} A_1^{n+1} \mid B_1^{m-1} \mid P'_1 \qquad A_2^n \mid B_2^m \mid P_2 \rightarrow^{(n\times(n-1))/2\times 2r/g_2\times g_2} A_2^{n-1} \mid B_2^{m+1} \mid P'_2$$
$$A_1^n \mid B_1^m \mid P_1 \rightarrow^{n\times m\times s/g_2\times g_2} A_1^{n-1} \mid B_1^{m+1} \mid P''_1 \qquad A_2^n \mid B_2^m \mid P_2 \rightarrow^{(m\times(m-1))/2\times 2s/g_2\times g_2} A_2^{n+1} \mid B_2^{m-1} \mid P''_2$$

which have the same propensities as the interacting automata transitions. Here P'_i,P''_i are systems where a buffer has lost one element, but where the active gate populations that drive the transitions remain at the same level as in P_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.

Figure 10 shows another example: a 3-state automaton and a Gillespie simulation of 1500 such automata with r=1.0. The equation system and its translation to strand algebra are (take, e.g., $r=g_2=1.0$):

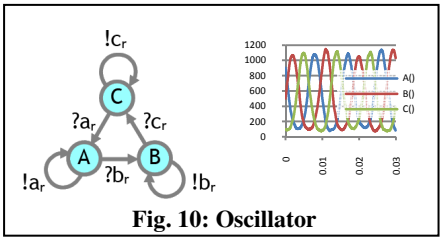

Fig. 10: Oscillator

$$\begin{aligned} A &= !a_r.A \oplus ?b_r.B & ([A,B].[B,B])^{=r/g_2} \mid \\ B &= !b_r.B \oplus ?c_r.C & ([B,C].[C,C])^{=r/g_2} \mid \\ C &= !c_r.C \oplus ?a_r.A & ([C,A].[A,A])^{=r/g_2} \mid \\ & A^{900} \mid B^{500} \mid C^{100} & A^{900} \mid B^{500} \mid C^{100} \end{aligned}$$

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 $\mathcal{P}$, 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 $\mathcal{P}$ 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:

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

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:

$?[x_1,..,x_n].P \mid x_1 \mid .. \mid x_n \;\rightarrow\; P$ 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$\mathcal{P}$ to $\mathcal{P}$. Let $\mathcal{X}$ be an infinite lists of distinct signals, and $\mathfrak{F}$ be the set of such $\mathcal{X}$'s. Let $\mathcal{X}_i$ be the i-th signal 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 signal that occurs in P. The unnest algorithm $U(P)_{\mathcal{X}}$, for $P\in n\mathcal{P}$ and $\mathcal{X}\in\mathfrak{F}_P$, is shown in Table 6.1–1. 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 | $!x_2$.Y) and Y = $?x_3$.(X | Y)' can be written as: '(?X.($?x_1$.X | $!x_2$.Y))* | (?Y.$?x_3$.(X | Y))*'.

As an example, 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:

Vend = ?coin. ![ok,mutex]. (Coffee | Refund)
Coffee = ?[mutex,coin]. !coffee. (Coffee | Vend)
Refund = ?[mutex,abort]. !refund. (Refund | Vend)

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.

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 descriptions 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 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.

I would like to acknowledge the Molecular Programming groups at Caltech for invaluable discussions and corrections. In particular, join and curried gate designs were extensively discussed with Lulu Qian, 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, 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, 2009.

[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] A. Marathe, A.E. Condon, R.M. Corn. On Combinatorial DNA Word Design. J. Comp. Biology 8(3) 201–219, 2001.

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

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

[13] W. Reisig. Petri Nets: An Introduction. Springer-Verlag, 1985.

[14] 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.

[15] 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.

[16] G. Seelig, D. Soloveichik, D.Y. Zhang, E. Winfree. Enzyme-Free Nucleic Acid Logic Circuits. Science, 314(8), 2006.

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

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

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

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