# What is a Parasite?
# Defining reaction and network properties in an open ended automata chemistry

Susan Stepney[1,2], Simon Hickinbotham[1,3]

[1]York Cross-disciplinary Centre for Systems Analysis
[2]Department of Computer Science, University of York, YO10 5DD
[3]Department of Electronic Engineering, University of York, YO10 5DD
susan.stepney@york.ac.uk

## Abstract

Open-ended novelty is one of the goals of ALife. This provides challenges for analysis as the system evolves. We provide definitions for several emergent properties, such as parasitism and hypercycles, observed to emerge in an RNA world configuration of the Stringmol automata chemistry, and show how these can simultaneously be mathematically simple, capture the complexity of the processes, and be readily implementable.

## Introduction

An open-ended system exhibits continual novelty, eventually moving outside any model we build of it (Banzhaf et al., 2016). This presents the challenge of how to detect when and how this happens. It is therefore necessary to identify features of the model that can be detected in the system, and to monitor when these features change, become more common, or disappear. More challenging is to complete such an analysis via the detection of new, *emergent* properties. Typically this can only be done *a posteriori*, once forensic interrogation of the system has indicated how to do so.

RNA world systems have similar initial properties, whether *in vitro* (Koonin et al., 2017) or *in silico* (Bansho et al., 2012): the process of replication happens when two entities in the system combine, and one partner manufactures a copy of the other partner. Replication is either 'self-self' with two identical entities, or 'self-other' where non-identical entities engage in replication. This copying process can be imperfect, so mutants arise, which may be better or worse at the task of replicating, or not have a facility to replicate at all. The situation may arise where one entity may *never* copy the other; that entity is labelled a parasite.

Stepney and Hickinbotham (2020) describe how an RNA world configuration of the Stringmol automata chemistry, with mutation that produces novel replicators and parasites, moves outside its original model of mutual replication between identical molecule classes. In this configuration, the strings resemble the RNA 'replicases' of the hypothesised RNA world of prebiotic evolution (Takeuchi and Hogeweg, 2012). Novel agents, reactions, reaction networks, and properties emerge, which require an update to the model and its meta-model to capture the changes. Here we describe the definitions used to capture those particular properties and changes observed.

The consensus has been that emergent parasitism drives a well-mixed RNA world system through a process of ever increasing efficiency of replication, which can be followed by extinction as the proportion of parasites in the system increases exponentially and the number of replicators available to support the population diminishes. Recently, however, Hickinbotham et al. (2021) have observed different behaviours where spatial pattern formation, and thereby higher order selection, prevents extinction as in Takeuchi and Hogeweg (2012). Runs which survive early extinction exhibit *slower* replication times concurrent with increases in population size. These emergent changes need to be analysed in detail, yet the system model is changing as evolution progresses: initially each agent in the system is classified as a replicator, a parasite, or 'other', however, as evolution proceeds, this classification becomes insufficient – different methods of replication emerge that are not strictly pairwise; a parasite on one replicator may be a replicator when paired with a different one; and so-on. Classification needs to move from the individual *agent* level to one of identifying properties of *reactions*, and later of reaction *networks*, in order to capture and analyse the evolution of the system's dynamics.

To automate analysis, we need a precise definition of the relevant properties. These can often be defined only after a kind of event has been observed 'in the wild'. The property then needs to be formalised, and added to the analysis toolset. There are conflicting requirement in the formalisation: definitions should: (i) be simple, yet capture the property with the intended meaning; (ii) capture the complexity and intricacies of the emergent phenomenon; (iii) be readily implementable. Here we describe the formalisation developed and implemented to analyse the results reported in Hickinbotham et al. (2021). We meet the requirements by: (i) having high level definitions that clearly state the property, which are (ii) compositions of lower level definitions that capture the complexities, and (iii) provably equivalent definitions that are implementable. Although some lower

level definitions pertain only to Stringmol, the higher level definitions have analogues in RNA world that may be applied to other systems.

## Stringmol overview

Stringmol (Hickinbotham et al., 2016, 2012) is an automata chemistry (Dittrich et al., 2001) in which the 'molecules' are programs encoded as strings of specially designed machine code instructions (opcodes). The sequence composition determines the bind probability, execution pathway, and product(s) of the reaction. Details of the Stringmol language and execution semantics are given in Hickinbotham et al. (2012). Here we summarise the main features.

**Binding** between two strings to form reacting pairs is probabilistic, based on the strength of string matching determined by a Smith-Waterman (SW) algorithm. Any portion of one string can bind to any portion of another – binding position is determined by sequence composition. The bind regions are aligned (not concatenated) and the stringmol reaction-program starts from the SW alignment position. When two strings bind, one is designated as string1 (or the 'first' string) and the other as string2 (or the 'second' string), depending on the position of the bind.

**Mutation** happens stochastically with a fixed probability when a program executes the "copy" opcode. On mutation, a randomly chosen different symbol is written.

**Decay** also happens stochastically, removing strings from the system with a fixed uniform probability each timestep. This frees up space for new strings, and ensures species of strings must be actively reproduced to maintain their presence in the arena. It is possible for the entire arena to 'die' if the community of strings is no longer self-maintaining. Decay also ensures there are no non-terminating reactions.

**Reactions** between strings occur by executing the sequences of opcodes of strings in the reacting pair. The start point of the reaction program is the end of the bind site on string1. The program executes, with one opcode executed per timestep, using opcodes of one or both strings, depending on the sequences. For a given reaction, one opcode is executed each timestep, until either the reaction program terminates (at which point the strings unbind), or probabilistic decay occurs. If a product string is created during the reaction, it is placed in any free site in the immediate neighborhood around string1; if there is no free site, that product string is discarded. Execution and effects are purely local to the pair of strings in the reaction, and any product strings that result.

**Arena**. A stringmol chemistry operates in an abstract container, in which multiple bound pairs of strings can interact with each other simultaneously. Early Stringmol experiments use an aspatial (well-mixed) container, where any two strings could potentially interact. More recently, spatial Stringmol has been implemented in a 2D toroidal grid. Here, strings can react only if they are the Moore neighbourhood, and products are placed in free sites in the Moore neighbourhood. See Hickinbotham et al. (2021) for details.

## Classifying reaction properties

An initial Stringmol system contains 'seed replicators' that can copy strings, and can mutate. After many timesteps, new strings with new properties, new reaction types, and new reaction networks, emerge. These include emergent hypercycles, emergent parasitism, and even emergent movement. This list is by no means exhaustive: other properties are observed, and further properties may still emerge as the runs continue.

We classify certain behaviours of interest by examining the reaction products. Rather than assigning a single classification label to a reaction, it is more informative to determine whether a reaction possesses one or more *properties*. The different forms of the reaction products can be used to assign the properties to each reaction.

Some properties can be assigned by examining single reactions, but some properties require mutual behaviour, so they require examination of a network of multiple reactions.

When running Stringmol experiments (analogous to in the wild, or *in vivo*), there is probabilistic binding, mutation, and decay, and strings may be discarded if there is no space in the grid. When analysing Stringmol reactions (analogous to in the lab, on *in vitro*), binding probability is set to one, mutation is set to zero, and there is always space for product strings. The definitions here refer to what happens in the reactions during analysis, in a 'perfect' world; experimental reactions may produce mutated variants, or discard products.

## Basic components

### Strings

A string is a non-empty sequence of opcodes. Here we do not consider the specific sequence, so our basic type is the set of all strings, denoted $\mathcal{S}$. A distinguished character, $\perp \notin \mathcal{S}$, represents a 'destroyed' reagent string. It is possible to determine this has occurred in spatial Stringmol, as the site of the destroyed original reactant will become unoccupied at the end of the reaction.

We define the set of all (possibly destroyed) strings:

$$\mathcal{S}_\perp = \{\perp\} \cup \mathcal{S} \tag{1}$$

We use uppercase letters, and primed variants, to represent individual (possibly destroyed) strings[1]: $A, B, C, \ldots, A', B', C', \ldots \in \mathcal{S}_\perp$. We use $L \in \mathcal{S}^*$ to represent a possibly empty list of (non-destroyed) strings.

[1] In the following definitions, there is no implication that different names to refer to distinct strings, unless a constraint is stated explicitly. In the examples of particular reactions, however, different names do refer to distinct strings.

## Binding

The binding strength for two strings is calculated using a Smith-Waterman style algorithm. The strongest binding site is chosen; if there are multiple equally strong sites discovered, binding occurs at the site with this value that is nearest the start of the program. The strings bind at the chosen site, with a probability based on the strength.

The designation of string1 and string2 depends on the lengths of the ends of the strings before the bind site. If the bind is 'asymmetric' (a shorter end on one string than the other), then the string with the shorter end before the bind site is the first partner. If the bind is 'symmetric' (the same length ends on both strings), then the first partner is chosen randomly. This leads to three possibilities for the first partner when attempting to bind strings $A$ and $B$: (i) no bind, $A$ and $B$ cannot react; (ii) binding site such that $A$ is always string1 and $B$ is always string2; (iii) binding site such that either $A$ or $B$ can be string1. However, it is more convenient in terms of definitions to look at the binding part of a reaction attempt from the perspective of a given one of the pair being string1. The binding part of a reaction attempt between strings $A$ and $B$ has different possibilities:

1. A bind happens, and a reaction occurs. We write $A \gg B$ to indicate a bound pair where $A$ is the *first* partner and $B$ is the *second* partner.
2. A binding is not possible with $A$ as the first partner; we write $A \blacktriangleright B$ to say that $A$ cannot bind as first string.
3. No binding is possible, so no reaction occurs; we have both $A \blacktriangleright B$ and $B \blacktriangleright A$.

## Reactions

Once string1 is determined the reaction starts, with first and second reactant strings $[A, B]$. The program of string1 starts to execute. Each timestep, one opcode is executed, until either the reaction program terminates, or probabilistic decay occurs. If the reaction terminates, we have a final state of potentially changed or destroyed reactants, plus a possibly empty list of product strings: $[A, B] \rightarrow [A', B'] + L$. Each product string in $L$ is placed in an unoccupied cell in $A$'s Moore neighbourhood as it is produced, or, if there are no free spaces at that time, it is discarded.

If the reaction decays, both reactant strings are destroyed; any products produced up to that point have already been placed in free locations: $[A, B] \rightarrow [\perp, \perp] + L$.

We use the following notation:

$$\_ \gg \_ : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}_\perp^2 \times \mathcal{S}^* \quad (2)$$

$$A \gg B \rightarrow [A', B'] + L \quad (3)$$

$A$ and $B$ are the input *reactants*. $A'$ and $B'$ are the *output* reactants; reactants may be changed by the reaction; $L$ is the list of reaction *products*[2]. We use the asymmetric operator $\gg$ as a visual indication that $A$ is string1 and $B$ is string2 in such a definition.

The product list $L$ may be empty (nothing produced) $[\,]$, a single string $[C]$, or multiple strings $[C, D, \ldots]$. In our experiments, we observe only a few reactions that result in two products, and a very few with three products. The majority of reactions result in zero or one product strings.

Note that the reaction operation in Stringmol is non-commutative: $A \gg B \neq B \gg A$; which string is the designated first string[3] matters, as the program execution starts on that string (the execution path may move back and forth between strings as the program executes, but it starts on the first string). This leads to some difference and subtleties in the definition of various Stringmol reaction properties that do not occur in natural chemistry, since here definitions have to include the concept of first and second string.

Additionally, in spatial Stringmol, we can distinguish output reactants from products in the after state, because strings are located in particular grid positions, and reactants do not move. This leads to some distinctions in the following definitions that are not made in the earlier aspatial Stringmol.

## Properties

We use the following notation to define some properties of reactions.

We define properties of single reactions:

$$\mathsf{prop\text{-}name}(A \gg B) \triangleq A \gg B \rightarrow [A', B'] + L \quad (4)$$

This can be read as: 'the reaction of string1 $A$ with string2 $B$ establishes the property $\mathsf{prop\text{-}name}(A \gg B)$'. The property holds (is true) if the reaction produces the strings $[A', B'] + L$. It is false either if the result is other than $[A', B'] + L$, or if $A$ cannot bind as first string to $B$ (that is, if $A \blacktriangleright B$).

We also define properties of two or more reactions involving two strings, some with one string as the first string, some with the other as first string. Basic properties are combined

[2]How to read these definitions: Eqn.2 shows the type of the reaction function $\gg$. It is an infix function $[\_ \gg \_]$ that takes two arguments $[\mathcal{S} \times \mathcal{S}]$; these are the reactant stringmols comprising the first stringmol and the second stringmol. It returns two lists (sometimes written as one concatenated list); the first of which has two stringmols $[\mathcal{S}_\perp^2]$, either of which might be 'destroyed' (these are the resulting reactant stringmols), the second of which $[\mathcal{S}^*]$ is a list of product stringmols (which may be empty). Eqn.3 shows a generic instance of the reaction: the reactants are first string $A$ and second string $B$; the result is the (possibly changed or destroyed) output reactants $A'$ and $B'$ and the (possibly empty) list of products $L$.

[3]The terms 'first' and 'second' are used, because the code starts executing on the first partner, string1. The original hand-designed replicator string has all the executing code on the first partner, which was therefore called 'active', and all the copied code on the second partner partner, which was therefore called 'passive'. As the system evolves, code may execute on either or both strings; we have changed the original terminology, which had become unhelpful.

to produce more complex properties, by using prop-name as a predicate in further definitions. For these, we write:

$$\text{prop-name}(A \Diamond B) \triangleq \Phi \tag{5}$$

where $\Phi$ is a combination of other reaction properties. This can be read as: 'the reaction of $A$ and $B$ (in one of possibly multiple different ways) establishes the property prop-name$(A \Diamond B)$'. The property holds (is true) if the combination of properties $\Phi$ on the RHS is true. We use the symmetric operator $\Diamond$ as a visual indication that sometimes $A$ and sometimes $B$ is string1 in the components of such a definition.

The definition of a property here does not imply that it need be observed in experiments; some are defined to be helpful in other definitions; some are counterfactual properties, explicitly required *not* to (be able to) occur in certain classification cases.

## Properties of individual reactions

Some properties can be identified by studying a single reaction only. They are used as the building-blocks to identifying more complex properties formed from networks of molecular interactions.

### React and no-bind properties

For convenience, we define the two binding possibilites as properties.

*Definition – react*: a reaction occurs with $A$ as string1:

$$\text{react}(A \succcurlyeq B) \triangleq A \succcurlyeq B \rightarrow [A', B'] + L \tag{6}$$

*Definition – no-bind*: a reaction cannot occur as $A$ cannot bind as string1:

$$\text{no-bind}(A \blacktriangleright B) \triangleq A \blacktriangleright B \tag{7}$$

These two cases exhaust the possibilities: either $A$ can bind as string1 with $B$ and react, or it cannot bind as string1:

$$\text{react}(A \succcurlyeq B) \equiv \neg\, \text{no-bind}(A \blacktriangleright B) \tag{8}$$

### Self-preserving and self-modifying properties

The hand-designed replicator in Stringmol simply copies string $B$, not changing either of the reactants. As evolution proceeds, program execution gets more complicated, and the reactants may be modified during program execution (self modifying code), even during analysis where the mutation-on-copy rate is set to zero. We define the cases where the reactants are preserved or changed.

*Definition – Self-preserving reaction*: a reaction where neither reactant is changed:

$$\text{self-pres}(A \succcurlyeq B) \triangleq A \succcurlyeq B \rightarrow [A, B] + L \tag{9}$$

Examples of the self-pres$(A \succcurlyeq B)$ property:

$$A \succcurlyeq B \rightarrow [A, B] \tag{10}$$
$$A \succcurlyeq B \rightarrow [A, B, C] \tag{11}$$

A self-preserving reaction makes no changes to the reactants: it might produce no products (eqn.10), or a product string (eqn.11).

*Definition – Self-modifying reaction*: a reaction where at least one reactant is changed:

$$\text{self-mod}(A \succcurlyeq B) \triangleq A \succcurlyeq B \rightarrow [A', B'] + L \quad \text{where } A \neq A' \vee B \neq B' \tag{12}$$

Examples of the self-mod$(A \succcurlyeq B)$ property:

$$A \succcurlyeq B \rightarrow [A', B] \tag{13}$$
$$A \succcurlyeq B \rightarrow [A, B', C] \tag{14}$$
$$A \succcurlyeq B \rightarrow [\bot, B, A] \tag{15}$$
$$A \succcurlyeq B \rightarrow [\bot, \bot] \tag{16}$$

A self-modifying reaction might modify string1 (eqns.13,15), string2 (eqn.14), or both reactant strings (eqn.16). It might also produce a product (eqns.14,15), which might contain a (copy of) the modified string (eqn.15). Modifications include destruction of either or both reactants (eqns.15,16).

Any given reaction is either self preserving or self modifying, but not both:

$$\text{react}(A \succcurlyeq B) \equiv \text{self-pres}(A \succcurlyeq B) \ \text{XOR}\ \text{self-mod}(A \succcurlyeq B) \tag{17}$$

Because the position of a string in spatial Stringmol is explicit and unchanging, it is possible to detect the difference between two reactions with identical output reactants and products, such as the following pair of a self-preserving (eqn.18) and a self-modifying (eqn.19) reaction:

$$A \succcurlyeq B \rightarrow [A, B] + [C, D] \tag{18}$$
$$A \succcurlyeq B \rightarrow [C, D] + [A, B] \tag{19}$$

This distinguishability has consequences in a spatial Stringmol system: the output reactants stay in the same location, even if modified, whereas any newly created product strings, the $L$, are placed in adjacent locations. So the position of a string on the grid is fixed; a string can be changed only via a reaction, and a particular string can 'move' only if it is copied (or recreated in another way) into another grid position. (In aspatial well-mixed Stringmol systems, the difference has no direct consequences.)

### Properties of the products

**Product.** Most reactions of interest result in product string(s); we call these product reactions.

*Definition – product reaction*: a reaction where at least one product is formed:

$$\textsf{product}(A \gtrdot B) \triangleq A \gtrdot B \rightarrow [A', B'] + L \quad \text{where } L \neq [\,] \tag{20}$$

**No product.** A few reactions of interest do not result in any products. There may be a change to the reactants.

*Definition – no product reaction*: a reaction that yields no product strings:

$$\textsf{no-product}(A \gtrdot B) \triangleq A \gtrdot B \rightarrow [A', B'] \tag{21}$$

Any given reaction is either a product or no-product reaction, but not both:

$$\textsf{react}(A \gtrdot B) \equiv \textsf{product}(A \gtrdot B) \ \textsc{xor}\ \textsf{no-product}(A \gtrdot B) \tag{22}$$

**Null reaction.** If a reaction is both self-preserving (hence not self-modifying), and makes no-product, it doing nothing except consuming execution time.

*Definition – null reaction*: a reaction where the reactants do not change, and there are no products:

$$\textsf{null}(A \gtrdot B) \triangleq \textsf{self-pres}(A \gtrdot B) \wedge \textsf{no-product}(A \gtrdot B) \tag{23}$$

It is important for understanding the evolutionary dynamics of an experimental run to include such cases where apparently 'nothing happens'. A no-bind occupies one timestep, for the (failed) bind attempt. A null reaction occupies at least two timesteps: one for the bind, and minimally, one to execute the 'end reaction' opcode. It may occupy many more timesteps, executing opcodes to no effect, and we see such behaviour evolve as a response to parasites (Hickinbotham et al., 2021). Bound strings are unavailable for binding to other strings: this can prevent them from being copied and reduce their ability to fix in the system.

Equivalently, we can say that a null reaction is a reaction that is neither self-modifying, nor makes a product:

$$\textsf{null}(A \gtrdot B) \equiv \textsf{react}(A \gtrdot B) \wedge \neg\, \textsf{self-mod}(A \gtrdot B) \wedge \neg\, \textsf{product}(A \gtrdot B) \tag{24}$$

**New products.** Products may be strings different from the original reactants. Simply having $A$ or $B$ be changed by the reaction is not considered to be a new product reaction, rather, that is classed as a self-modifying reaction.

*Definition – new product reaction*: a reaction that yields a product string that is different from either reactant string[4]:

$$\textsf{new-prod}(A \gtrdot B) \triangleq A \gtrdot B \rightarrow [A', B'] + L \quad \text{where } L \setminus \{A, B\} \neq [\,] \tag{25}$$

[4]The operator $\_ \setminus \_ : \mathcal{S}^* \times \mathbb{P}\mathcal{S} \rightarrow \mathcal{S}^*$ takes a list of strings (here, $L$), and a set of strings (here, $\{A, B\}$), and results in a list that has had all the elements in the set removed from it. It can be thought of as a list version of set difference. So the definition says that there is least one string in list $L$ that is neither $A$ nor $B$; hence, a 'new' product.

Examples of the $\textsf{new-prod}(A \gtrdot B)$ property :

$$A \gtrdot B \rightarrow [A, B, C] \tag{26}$$
$$A \gtrdot B \rightarrow [A, B', C, D] \tag{27}$$
$$A \gtrdot B \rightarrow [A', B, B, C] \tag{28}$$

A new product reaction might make a single new product (eqn.26), or multiple new products (eqn.27). It might additionally make copies of reactants (eqn.28). It is self-preserving (eqns.26) or self-modifying (eqn.27,28).

## Replicator properties

Replication has occurred when there are more instances of one (or both) of the reactant strings after the reaction than before. This can happen if one of the reactant strings is copied, and the original is not modified, or if one of the reactant strings is copied multiple times, and the original modified, or even if one of the reactants is modified to be a copy of the other with no product produced. The key property is that there are more instances of the given reactant string after the reaction than before.

We first define two cases, where string2 or string1 is the string that is replicated.

*Definition – string2 replication reaction*: a reaction where the second string is replicated: there are more copies of the second string after the reaction than before[5]:

$$\textsf{repl2}(A \gtrdot B) \triangleq A \gtrdot B \rightarrow [A', B'] + L \quad \text{where } \#B \,\textsf{in}\, [A, B] < \#B \,\textsf{in}\, [A', B'] + L \tag{29}$$

*Definition – string1 replication reaction*: a reaction where the first string is replicated: there are more copies of the first string after the reaction than before.

$$\textsf{repl1}(A \gtrdot B) \triangleq A \gtrdot B \rightarrow [A', B'] + L \quad \text{where } \#A \,\textsf{in}\, [A, B] < \#A \,\textsf{in}\, [A', B'] + L \tag{30}$$

Examples of the $\textsf{repl2}(A \gtrdot B)$ property:

$$A \gtrdot B \rightarrow [A, B, B] \tag{31}$$
$$A \gtrdot B \rightarrow [A, B', B, B] \tag{32}$$
$$A \gtrdot B \rightarrow [A', B, B, B, B, C] \tag{33}$$
$$A \gtrdot B \rightarrow [B, B', B] \tag{34}$$
$$A \gtrdot B \rightarrow [A, B, B, A] \tag{35}$$

The repl2 reaction might make a single (eqns.31,32,34,35) or multiple (eqn.33) copies of string2. It is self-preserving (eqns.31,35) or self-modifying (eqns.32,33,34). It might

[5]The operator $\#\_\,\textsf{in}\,\_ : \mathcal{S} \times \mathcal{S}^*_\perp \rightarrow \mathbb{N}$ counts the number of times a string occurs in a list of strings.

also produce a new product (eqn.33). If might replicate both reactants (eqn.35).

The hand-designed Stringmol replicator string is designed to have the repl2$(A \gg B)$ property where string2 $B$ is replicated as in example eqn.31.

The repl1$(A \gg B)$ property, where the first string $A$ is replicated, means that $A$ can replicate itself without needing (to be able) to bind to itself, instead using some different second partner, $B$, as a kind of catalyst. This partner may be quite variable, if first string $A$ carries the copying code within itself; it may need to be a replicator if $A$ hijacks the copying code from its second partner.

A more classical 'copying machine plus copied template' model assumes all the code is located in the single machine string. In Stringmol, evolution not only allows, but seems to encourage, code execution to flip back and forth between strings, as a protection against parasites (Hickinbotham et al., 2021). The generality of the repl1 and repl2 definitions is needed here to encompass the complexity of these replication forms that arise in an evolving Stringmol system.

We combine these two kinds to define a general replication reaction.

*Definition – replication reaction*: a reaction in which string $A$ replicates string $B$: there are more copies of $B$ after the reaction than before. String $A$ is called the *replicator*.

$$\text{repl}(A \lozenge B) \triangleq \text{repl2}(A \gg B) \vee \text{repl1}(B \gg A) \tag{36}$$

The property repl$(A \lozenge B)$ can be read as '$A$ replicates $B$'. Note the different order of $A$ and $B$ in the two properties on the RHS of the definition. $B$ might be the second string, replicated by first string $A$ (the term repl2$(A \gg B)$), or it might be the first string, copying itself (a possible behaviour giving the term repl1$(B \gg A)$)[6]. We cannot tell where the replication code lies in either case and the definition does not require it to be known. Whether or not $A$ is the first string, whether or not $A$ carries the copying code, it is the 'catalyst' for $B$'s replication. So, irrespective of which string is first, we say '$A$ replicates $B$', and call $A$ the *replicator*.

The property repl$(A \lozenge B)$ is the main property of interest for replication. It is defined here in terms of the more basic repl1 and repl2, reflecting the underlying features of the Stringmol system. A different automata chemistry might well have the same effective repl$(A \lozenge B)$ property, but be based on some other replX, replY, replZ basic properties that reflect the underlying mechanisms in that other system.

## Jumper reactions

In a reaction that has the jumper property, at least one of the reactants is changed, and the products include a copy of that reactant. The effect is that the changed string has 'jumped' from its original place into a new place, resulting in an emergent 'movement' of the string (much the way gliders 'move' in the Game of Life, despite the underlying static grid).

*Definition – jumper2 reaction*: a reaction in which string2 'jumps' to a new location (in string1's Moore neighbourhood[7]): the original reactant is altered or destroyed, and a new copy is made as a product:

$$\text{jumper2}(A \gg B) \triangleq\ A \gg B \rightarrow [A', B'] + L \quad \text{where } B' \neq B \wedge B \in L \tag{37}$$

*Definition – jumper1 reaction*. a reaction in which string1 'jumps' to a new location (in its Moore neighbourhood): the original reactant is altered or destroyed, and a new copy is made as a product:

$$\text{jumper1}(A \gg B) \triangleq\ A \gg B \rightarrow [A', B'] + L \quad \text{where } A' \neq A \wedge A \in L \tag{38}$$

Examples of the jumper2$(A \gg B)$ property:

$$A \gg B \rightarrow [A, \perp, B] \tag{39}$$
$$A \gg B \rightarrow [A, B', A, B] \tag{40}$$
$$A \gg B \rightarrow [A, B', B, B] \tag{41}$$

In the simplest jumper2 reaction, $B$ jumps to a new location, and the original is destroyed (eqn.39). $A$ might also be replicated (eqn.40); $B$ might also be replicated (eqn.41).

*Definition – jumper reaction*: a reaction in which $B$ 'jumps' to a new location, either as a jumper1 or as a jumper2:

$$\text{jumper}(A \lozenge B) \triangleq \text{jumper1}(B \gg A) \vee \text{jumper2}(A \gg B) \tag{42}$$

# Network properties

Here we define some observed properties of networks of reactions. The definition of these properties depends on the properties of more than one reaction, including counterfactual cases (reactions that do not occur).

## Mutual replication

A simple network property that does not require counterfactual properties is mutual replication.

*Definition – mutual replication*: the mutual replication property holds where each string can replicate the other:

$$\text{mutual-repl}(A \lozenge B) \triangleq\ \text{repl}(A \lozenge B) \wedge \text{repl}(B \lozenge A) \tag{43}$$

The property mutual-repl$(A \lozenge B)$ can be read as '$A$ replicates $B$ and $B$ replicates $A$' .

[6] This explains the reason for having separate definitions for repl1 and repl2. It allows us to flip the order of the arguments of repl1 in the definition of repl, so we can identify which reactant string is the replicator, which may be either string1 or string2.

[7] We define this as a jumper reaction only if $B$ jumps away from $(A, B)$, so does not include the case where $B$ 'jumps' to $A$'s position.

Examples of the mutual-repl$(A \Diamond B)$ property:

$$A \succcurlyeq B \rightarrow [A, B, B] \quad \wedge B \succcurlyeq A \rightarrow [B, A, A] \tag{44}$$
$$A \succcurlyeq B \rightarrow [A, B, A] \quad \wedge B \succcurlyeq A \rightarrow [B, A, B] \tag{45}$$
$$A \succcurlyeq B \rightarrow [A, B, A, B] \quad \wedge B \blacktriangleright A \tag{46}$$

The classic case (eqn.44) is $A$ replicates $B$, and $B$ replicates $A$, as case repl2$(A \succcurlyeq B) \wedge$ repl2$(B \succcurlyeq A)$. There are other possibilities: each string might replicate itself, using the other as a catalyst, as case repl1$(A \succcurlyeq B) \wedge$ repl1$(B \succcurlyeq A)$ (eqn.45); $A$ might replicate $B$ and replicate itself, as case repl1$(A \succcurlyeq B) \wedge$ repl2$(A \succcurlyeq B)$ (eqn.46).

## Hypercycle property

The definition of hypercycles[8] requires a counterfactual property, in that strings should not be able to replicate themselves, only each other.

*Definition – hypercycle*: the hypercycle property holds of a pair of reactants if they are mutual replicators but are not self replicators:

$$\text{hypercycle}(A \Diamond B) \triangleq \text{mutual-repl}(A \Diamond B) \wedge \neg \text{repl}(A \Diamond A) \wedge \neg \text{repl}(B \Diamond B) \tag{47}$$

$A$ and $B$ depend on each other for replication, as neither can replicate itself. Hypercycles are observed as an emergent property in aspatial stringmol (Hickinbotham et al., 2016).

This definition can be extended to hypercycles of more than two reactants in the obvious way.

Examples of the hypercycle$(A \Diamond B)$ property:

$$A \succcurlyeq B \rightarrow [A, B, B] \wedge B \succcurlyeq A \rightarrow [B, A, A] \wedge A \blacktriangleright A \wedge B \blacktriangleright B \tag{48}$$

$$A \succcurlyeq B \rightarrow [A, B, B] \wedge B \succcurlyeq A \rightarrow [B, A, A] \wedge A \succcurlyeq A \rightarrow [A, A, A'] \wedge B \succcurlyeq B \rightarrow [B, B, B'] \tag{49}$$

The classic case (eqn.48) is $A$ replicates $B$, and $B$ replicates $A$, as case repl2$(A \succcurlyeq B) \wedge$ repl2$(B \succcurlyeq A)$, and neither $A$ nor $B$ self-replicate because they cannot self-bind. Alternatively, the lack of self-replication may have self-binding, but no self-production (eqn.49)

## Parasitic property

Parasitic reactions in Stringmol have 'freeloader' strings that are replicated, but do not themselves replicate other strings. The definition of the parasitic property requires both of these cases to hold.

*Definition – parasitic property*: The parasitic property holds of strings $P$ and $R$ if $R$ replicates $P$, but $P$ does not replicate $R$:

$$\text{parasitic}(P \Diamond R) \triangleq \text{repl}(R \Diamond P) \wedge \neg \text{repl}(P \Diamond R) \tag{50}$$

[8] Hypercycles in Stringmol are "catalytic hypercycles", as defined by Eigen and Schuster (2012, fig.7)

The property parasitic$(P \Diamond R)$ can be read as '$P$ is parasitic on $R$'. In the context of $R$, $P$ is a parasite, but $P$ need not be a parasite in the context of other strings.

Examples of parasitic$(P \Diamond R)$ reaction pairs:

$$R \succcurlyeq P \rightarrow [R, P, P] \ \wedge \ P \blacktriangleright R \tag{51}$$
$$R \succcurlyeq P \rightarrow [R, P, P] \ \wedge \ P \succcurlyeq R \rightarrow [P, R, A] \tag{52}$$
$$R \succcurlyeq P \rightarrow [R, P, P] \ \wedge \ P \succcurlyeq R \rightarrow [P, R', R] \tag{53}$$
$$P \succcurlyeq R \rightarrow [P, R, P] \ \wedge \ R \blacktriangleright P \tag{54}$$

A parasitic reaction depends on the behaviour of each string as a replicator. The classic case is $R$ replicates $P$, but $P$ does not bind as the first partner to $R$, and so does not replicate it (eqn.51). This case is commonly seen in aspatial stringmol (Hickinbotham et al., 2016), and in the early stages of the spatial Stringmol runs (Hickinbotham et al., 2021).

There are other possibilities: $P$ might bind as string1 but produce something other than $R$, a new-product$(P \succcurlyeq R)$ reaction (eqn.52); $P$ might indeed copy $R$ but the original is destroyed, a jumper2$(P \succcurlyeq R)$ reaction (eqn.53). The definition admits yet more exotic possibilities: $R$ might replicate $P$ where $P$ is string1 (a case of repl1$(P, R)$), with $R$ not being able to bind as string1 (eqn.54).

We classify the $P$ as a parasite only in the context of $R$. $P$ may not be a parasite with respect to a different string $R'$: $R'$ might not replicate $P$ (so the first conjunct of eqn.50 does not hold), or $P$ might replicate $R'$ (so the second conjunct does not hold). Indeed, if we have both $R'$ does not replicate $P$ ($\neg$ repl$(R' \Diamond P)$) *and* $P$ replicates $R'$ (repl$(P \Diamond R')$), then $R'$ is the parasite in that context.

For purposes of implementation, where properties of the string1 and string2 may be calculated at different times, it is useful to split the parasitic property into two parts, one that holds when the parasite $P$ is the second string and one where it is the first string.

*Definition – para2*: the para2$(P \Diamond R)$ property holds when string1 $R$ replicates string2 $P$, but $P$ does not replicate $R$:

$$\text{para2}(P \Diamond R) \triangleq \text{repl2}(R \succcurlyeq P) \wedge \neg \text{repl}(P \Diamond R) \tag{55}$$

*Definition – para1*: the para1$(P \Diamond R)$ property holds when string2 $R$ replicates string1 $P$, but $P$ does not replicate $R$:

$$\text{para1}(P \Diamond R) \triangleq \text{repl1}(P \succcurlyeq R) \wedge \neg \text{repl}(P \Diamond R) \tag{56}$$

We can show some equivalences:

$$\text{parasitic}(P \Diamond R) \equiv \text{para1}(P \Diamond R) \vee \text{para2}(P \Diamond R) \tag{57}$$
$$\text{para2}(P \Diamond R) \equiv \text{repl2}(R \succcurlyeq P) \wedge \neg(\text{repl1}(R \succcurlyeq P) \vee \text{repl2}(P \succcurlyeq R)) \tag{58}$$
$$\text{para1}(P \Diamond R) \equiv \text{repl1}(P \succcurlyeq R) \wedge \neg(\text{repl1}(R \succcurlyeq P) \vee \text{repl2}(P \succcurlyeq R)) \tag{59}$$

Breaking down the definition of parasitic in this way provides easier implementation of analysis in Stringmol, but

the definitions are too involved to be appropriate as the 'intuitive' definition of the parasitic property. Indeed, the formalisation exercise reported here was performed precisely in order to understand and analyse parasitism in spatial Stringmol. Finding a formulation that is both simple and implementable has not been possible for parasitism, because of some of the intricacies in how evolved strings in Stringmol can behave, and also in how the analysis is implemented. Formalisation allows us to prove equivalences between different formulations, and so provides the best of both worlds: (relative) clarity of definition (eqn.50), and ease of implementation (eqns.58,59).

Any given replication reaction is either mutual replication or parasitic, but not both. In order to know which, the reverse reaction must also be considered.

$$\begin{aligned}\mathsf{repl}(A \lozenge B) \equiv\ & \mathsf{mutual\text{-}repl}(A \lozenge B) \\ & \text{XOR } \mathsf{parasitic}(B \lozenge A)\end{aligned} \tag{60}$$

Note the change in order of arguments in this. We have: '$A$ replicates $B$ iff either $A$ and $B$ mutually replicate, or $B$ is parasitic on $A$', so $A$ is the replicator in both terms.

## Discussion

In simulations and experiments with 'closed' form and where the interrogation of the system is straightforward, the task of identifying and following the abundance of features is trivial. In studies of evolution where the goal is to generate an 'open' system, it becomes less clear what the defining features are. As we get closer to the goal of open-ended evolution the emergence of such novel features is something to be expected, and planned for as part of the experimental design.

The definition of properties given above was developed to characterise the analysis in Hickinbotham et al. (2021) in a way that is understandable, correct, and implementable. Evolution in the system generates changes in the replication reaction that can be captured and tracked only via a formalisation that accommodates the observed phenomena, which helps us to identify the edge cases that let us identify when the system had moved outside its original model. This is particularly useful in the definition of the parasitic property, as several alternative complex behaviours are collected in a single high level definition (eqn.50). In Stringmol, the parasitic property has two possible execution pathways, para1 and para2, from the way Stringmol decides which string executes initially. Each different system may well have a definition of parasitic in terms of its own idiosyncratic pathways; stating formally how this is defined will make it easier to compare emergence between systems.

The properties defined above are all based on examining the outcomes of reactions in Stringmol experiments. In Stringmol, this is more feasible than trying to determine properties by looking for patterns in the opcode strings, given the evolved complexity of opcode execution order, which loops, can skip over substrings, and can flip back and forth between strings multiple times per reaction. Thus it is not possible just to identify 'the replication code' in a particular string: the defined properties are genuinely properties of reactions or networks, not of individual strings. Having developed the formalism given here, it was then relatively straightforward to automate the analysis of the experimental results to identify the emergence and dynamics of these properties, as described in Hickinbotham et al. (2021).

The properties defined here are *in vitro* analysis properties, of perfect execution in an environment where mutation has been switched off. Further work is needed to define *in vivo* properties of evolving systems, such as parasite and replicator lineages, and other yet-to-emerge, or yet-to-be-recognised, properties.

In future, we plan to implement a 'symbolic' run of the Stringmol system, using these definitions as proxies for the actual program executions, attaching and evolving rates to each reaction. Although such a system cannot be open-ended in the same fine-grained way provided by detailed string mutation, it would allow larger-scale simulations to be run more quickly, leading to an understanding of the Stringmol system that may be used to develop new open-ended simulation frameworks. It may also be possible to combine the detailed Stringmol system with these symbolic runs, to get a large-scale, open-ended simulation, such as described in Nellis and Stepney (2010).

In a truly open system, this process of analysis via defining new properties will be on-going. As the system continually moves outside its model, entirely new kinds of properties, applied to different features at different scales, will be needed. It is not possible to define all such properties *a priori*; the very presence of open-endedness precludes it. Here we have shown, by example, a method for capturing properties that are simultaneously mathematically simple, capture complexity, and readily implementable.

## Acknowledgments

The logfiles from the runs were parsed with these definitions using the R software package github.com/uoy-research/Rstringmol v0.3.1. The dataset analysed during the development of this study is available at doi: 10.15124/305dfdb6-9483-4c5b-8a01-c030570b9c31

The Stringmol system was originally developed under the Plazzmid project, EPSRC grant EP/F031033/1, and was further developed under the EU FP7 project EvoEvo, grant number 610427. We thank Paulien Hogeweg for several stimulating discussions on what precisely is a 'parasite'.

## References

Bansho, Y., Ichihashi, N., Kazuta, Y., Matsuura, T., Suzuki, H., and Yomo, T. (2012). Importance of parasite RNA species repression for prolonged translation-coupled RNA self-replication. *Chemistry & Biology*, 19(4):478–487.

Banzhaf, W., Baumgaertner, B., Beslon, G., Doursat, R., Foster, J. A., McMullin, B., de Melo, V. V., Miconi, T., Spector, L., Stepney, S., and White, R. (2016). Defining and simulating open-ended novelty: Requirements, guidelines, and challenges. *Theory in Biosciences*, 135(3):131–161.

Dittrich, P., Ziegler, J., and Banzhaf, W. (2001). Artificial Chemistries—A review. *Artificial Life*, 7(3):225–275.

Eigen, M. and Schuster, P. (2012). *The Hypercycle: A Principle of Natural Self-Organization*. Springer.

Hickinbotham, S., Clark, E., Nellis, A., Stepney, S., Clarke, T., and Young, P. (2016). Maximizing the adjacent possible in automata chemistries. *Artificial Life*, 22(1):49–75.

Hickinbotham, S., Clark, E., Stepney, S., Clarke, T., Nellis, A., Pay, M., and Young, P. (2012). Specification of the stringmol chemical programming language version 0.2. Technical Report YCS-2010-458, University of York. https://www.cs.york.ac.uk/library/reports/2010/YCS/458/YCS-2010-458.pdf.

Hickinbotham, S. J., Stepney, S., and Hogeweg, P. (2021). Nothing in evolution makes sense except in the light of parasites. *bioR$\chi$iv*. https://www.biorxiv.org/content/10.1101/2021.02.25.432891v1.

Koonin, E. V., Wolf, Y. I., and Katsnelson, M. I. (2017). Inevitability of the emergence and persistence of genetic parasites caused by evolutionary instability of parasite-free states. *Biology Direct*, 12(1):31.

Nellis, A. and Stepney, S. (2010). Automatically moving between levels in artificial chemistries. In *ALife XII, Odense, Denmark, August 2010*, pages 269–276. MIT Press.

Stepney, S. and Hickinbotham, S. (2020). Innovation, variation, and emergence in an automata chemistry. *Artificial Life 2020, Montreal, Canada (virtual), July 2020*, 32:753–760.

Takeuchi, N. and Hogeweg, P. (2012). Evolutionary dynamics of RNA-like replicator systems: A bioinformatic approach to the origin of life. *Phys. Life Rev.*, 9(3):219–263.