Programmability of Chemical Reaction Networks

Matthew Cook[1], David Soloveichik[2], Erik Winfree[2], and Jehoshua Bruck[2]

[1] Institute of Neuroinformatics, UZH, ETH Zürich, Switzerland
cook@ini.phys.ethz.ch

[2] California Institute of Technology, Pasadena, California, USA
{dsolov,winfree,bruck}@caltech.edu

Summary. Motivated by the intriguing complexity of biochemical circuitry within individual cells we study Stochastic Chemical Reaction Networks (SCRNs), a formal model that considers a set of chemical reactions acting on a finite number of molecules in a well-stirred solution according to standard chemical kinetics equations. SCRNs have been widely used for describing naturally occurring (bio)chemical systems, and with the advent of synthetic biology they become a promising language for the design of artificial biochemical circuits. Our interest here is the computational power of SCRNs and how they relate to more conventional models of computation. We survey known connections and give new connections between SCRNs and Boolean Logic Circuits, Vector Addition Systems, Petri Nets, Gate Implementability, Primitive Recursive Functions, Register Machines, Fractran, and Turing Machines. A theme to these investigations is the thin line between decidable and undecidable questions about SCRN behavior.

1 Introduction

Stochastic chemical reaction networks (SCRNs) are among the most fundamental models used in chemistry, biochemistry, and most recently, computational biology. Traditionally, analysis has focused on *mass action* kinetics, where reactions are assumed to involve sufficiently many molecules that the state of the system can be accurately represented by continuous molecular concentrations with the dynamics given by deterministic differential equations. However, analyzing the kinetics of small-scale chemical processes involving a finite number of molecules, such as occurs within cells, requires stochastic dynamics that explicitly track the exact number of each molecular species [1, 2, 3]. For example, over 80% of the genes in the *E. coli* chromosome are expressed at fewer than a hundred copies per cell [4], averaging, for example, only 10 molecules of Lac repressor [5]. Further, observations and computer simulations have shown that stochastic effects resulting from these small numbers may be physiologically significant [6, 7, 8].

In this paper, we examine the computational power of Stochastic Chemical Reaction Networks. Stochastic Chemical Reaction Networks are closely related to computational models such as Petri nets [9], Vector Addition Systems (VASs) [10], Fractran [11, 12], and Register Machines (sometimes called Counter Machines) [13], and for many of these systems we can also consider stochastic or nondeterministic variants. Our initial route into this subject came through the analysis of a seemingly quite unrelated question: What digital logic circuits are constructible with a given set of gate types when it is not possible to copy values (as is true for example in quantum circuits)? It turns out that this gate implementability question, as we will discuss in Section 4.1, is very closely related to the question of what states can be reached by a Stochastic Chemical Reaction Network.

Given the importance of stochastic behavior in Chemical Reaction Networks, it it particularly interesting that whereas most questions of *possibility* concerning the behavior of these models are decidable [10], the corresponding questions of *probability* are undecidable [14, 15]. This result derives from showing that Stochastic Chemical Reaction Networks can simulate Register Machines [16] efficiently [17] within a known error bound that is independent of the unknown number of steps prior to halting [14]. This result – that when answers must be guaranteed to be correct, computational power is limited, but when an arbitrarily small error probability can be tolerated, the computational power is dramatically increased – can be immediately applied to the other models (Petri nets and VASs) when they are endowed with appropriate stochastic rates. This result is surprising, in light of the relatively ineffective role the addition of probability plays in the widely held belief that $BPP = P$.

Several further results extend and refine this distinction.

- When endowed with special *fast* reactions guaranteed to occur before any *slow* reaction, Stochastic Chemical Reaction Networks become Turing universal and thus can compute any computable function without error.
- Stochastic Chemical Reaction Networks with reaction rates governed by standard chemical kinetics can compute any computable function with probability of error less than ϵ for any $\epsilon > 0$, but for $\epsilon = 0$ universal computation is impossible [10, 17, 14].
- Stochastic Chemical Reaction Networks in which each reaction's probability of occuring depends only on what reactions are possible (but not on the concentrations) are not capable of universal computation with any fixed bounded probability of success.
- Taking the result of the longest possible sequence of reactions as the answer, Stochastic Chemical Reaction Networks are capable of computing exactly the class of primitive recursive functions without error.
- The time and space requirements for Stochastic Chemical Reaction Networks doing computation, compared to a Turing Machine, are a simple polynomial slowdown in time, but an exponential increase in space [17, 14].

This last result, regarding the complexity, is the best that can be expected, due to the unavoidable fact that information must effectively be stored in the bits comprising the number of molecules present of each species. For uniform computations, wherein the same finite set of chemical species and reactions are used to solve any instance of the problem, storing n bits requires the presence of $2^{\Omega(n)}$ molecules. In practice, keeping an exponentially large solution well-stirred may take a correspondingly large amount of time, but in any event, due to the space constraint, Stochastic Chemical Reaction Networks will effectively be limited to logspace computations.

The intention of this paper is to review, present, and discuss these results at an intuitive level, with an occasional foray into formal exactitude. Enjoy.

2 Formalization of Chemistry

2.1 Stochastic Chemical Reaction Networks

A Stochastic Chemical Reaction Network is defined as a finite set of d reactions acting on a finite number m of species. Each reaction α is defined as a vector of non-negative integers specifying the stoichiometry of the reactants, $\mathbf{r}_\alpha = (r_{\alpha,1}, \ldots, r_{\alpha,m})$, together with another vector of non-negative integers specifying the stoichiometry of the products, $\mathbf{p}_\alpha = (p_{\alpha,1}, \ldots, p_{\alpha,m})$. The stoichiometry is the non-negative number of copies of each species required for the reaction to take place, or produced when the reaction does take place. We will use capital letters to refer to various species and we will use standard chemical notation to describe reactions. So for example, the reaction $A + D \rightarrow A + 2E$ consumes 1 molecule of species A and 1 molecule of species D and produces 1 molecule of species A and 2 molecules of species E (see figure 1). In this reaction, A acts catalytically because it must be present for the reaction to occur, but its number is unchanged when the reaction does occur.[3]

The state of the network is defined as a vector of non-negative integers specifying the quantities present of each species, $\mathcal{A} = (q_1, \ldots, q_m)$. A reaction is possible in state $\mathcal{A}$ only if there are enough reactants present, that is, $\forall i, q_i \geq r_{\alpha,i}$. When reaction α occurs in state $\mathcal{A}$, the reactant molecules are used up and the products are produced. The new state is $\mathcal{B} = \mathcal{A} * \alpha = (q_1 - r_{\alpha,1} + p_{\alpha,1}, \ldots, q_m - r_{\alpha,m} + p_{\alpha,m})$. We write $\mathcal{A} \overset{C}{\rightarrow} \mathcal{B}$ if there is some reaction in the Stochastic Chemical Reaction Network C that can change $\mathcal{A}$ to $\mathcal{B}$; we write $\overset{C*}{\rightarrow}$ for the reflexive transitive closure of $\overset{C}{\rightarrow}$. We write $\Pr[\mathcal{A} \overset{C}{\rightarrow} \mathcal{B}]$ to indicate the probability that, given that the state is initially $\mathcal{A}$, the next reaction will transition to the state to $\mathcal{B}$. $\Pr[\mathcal{A} \overset{C*}{\rightarrow} \mathcal{B}]$ refers to the probability that at *some* time in the future, the system is in state $\mathcal{B}$.

[3] In chemistry, catalysis can involve a series of reactions or intermediate states. In this paper, however, we will generally use the word catalyst to mean a species which participates in, but is unchanged by, a *single* reaction.

Every reaction α has an associated rate constant $k_\alpha > 0$. The rate of every reaction α is proportional to the concentrations (number of molecules present) of each reactant, with the constant of proportionality being given by the rate constant k_α. Specifically, given a volume V, for any state $\mathcal{A} = (q_1, \ldots, q_m)$, the rate of reaction α in that state is

$$\rho_\alpha(\mathcal{A}) = k_\alpha V \prod_{i=1}^{m} \frac{(q_i)^{\underline{r_{\alpha,i}}}}{V^{r_{\alpha,i}}} \quad \text{where} \quad q^{\underline{r}} \overset{def}{=} \frac{q!}{(q-r)!} = q(q-1)\cdots(q-r+1). \tag{1}$$

Since the solution is assumed to be well-stirred, the time until a particular reaction α occurs in state $\mathcal{A}$ is an exponentially distributed random variable with the rate parameter $\rho_\alpha(\mathcal{A})$; i.e. the dynamics of a Stochastic Chemical Reaction Network is a continuous-time Markov process, defined as follows.

We write $\Pr[\mathcal{A} \overset{C}{\to} \mathcal{B}]$ to indicate the probability that, given that the state is initially $\mathcal{A}$, the next reaction will transition to the state to $\mathcal{B}$. These probabilities are given by

$$\Pr[\mathcal{A} \overset{C}{\to} \mathcal{B}] = \frac{\rho_{\mathcal{A}\to\mathcal{B}}}{\rho^{tot}_{\mathcal{A}}} \tag{2}$$

$$\text{where} \quad \rho_{\mathcal{A}\to\mathcal{B}} = \sum_{\alpha \text{ s.t. } \mathcal{A}*\alpha=\mathcal{B}} \rho_\alpha(\mathcal{A}) \quad \text{and} \quad \rho^{tot}_{\mathcal{A}} = \sum_{\mathcal{B}} \rho_{\mathcal{A}\to\mathcal{B}}$$

The average time for a step $\mathcal{A} \to \mathcal{B}$ to occur is $1/\rho^{tot}_{\mathcal{A}}$, and the average time for a sequence of steps is simply the sum of the average times for each step. We write $\Pr[\mathcal{A} \overset{C*}{\to} \mathcal{B}]$ to refer to the probability that at *some* time in the future, the system is in state $\mathcal{B}$.

This model is commonly used for biochemical modelling [1, 2, 3]. When using this model as a language for describing real chemical systems, the reasonableness of the underlying assumptions are affirmed (or denied) by the model's accuracy with respect to the real system. However, in the work presented here we will be using the model as a programming language—we will write down sets of formal chemical reactions that have no known basis in reality, and any network that is formally admitted by the model will be fair game. That is, while Stochastic Chemical Reaction Networks are usually used *descriptively*, we will be using them *prescriptively*: we imagine that if we can specify a network of interest to us, we can then hand it off to a talented synthetic chemist or synthetic biologist who will design molecules that carry out each of the reactions. Therefore, our concern is with what kinds of systems the formal model is capable of describing—because our philosophy is that if it can be described, it can be made. Of course, this might not be true. A similar issue arises in classical models of computation: It is often observed that Turing Machines cannot be built, because it is impossible to build an infinite tape or a machine that is infinitely reliable. Nonetheless, it is enlightening to study them. We believe the formal study of Stochastic Chemical Reaction Networks will be similarly enlightening. But before proceeding, it is worth considering just how unrealistic the model can become when we are given free reign to postulate arbitrary networks.

An immediate concern is that, while we will consider SCRNs that produce arbitrarily large numbers of molecules, it is impossible that so many molecules can fit within a pre-determined volume. Thus we recognize that the reaction volume V must change with the total number of molecules present, which in turn will slow down all reactions involving more than one molecule as reactants. Choosing V to scale proportionally with the total number of molecules present (of any form) results in a model appropriate for analysis of reaction times. Note, however, that for any Stochastic Chemical Reaction Network in which every reaction involves exactly the same number of reactants, the transition probabilities $\Pr[\mathcal{A} \overset{C}{\to} \mathcal{B}]$ are *independent* of the volume. For all the positive results discussed in this paper, we were able to design Stochastic Chemical Reaction Networks involving exactly two reactants in every reaction, and therefore volume needs to be considered only where computation time is treated. A remaining concern – which we cannot satisfactorily address – is that the assumption of a well-stirred reaction may become less tenable for large volumes. (However, this assumption seems intrinsically no less justified than the common assumption that wires in boolean circuits may be arbitrarily long without transmission errors, for example.)

A second immediate concern is that the reactions we consider are of a very general form, including reactions such as $A \to A + B$ that seem to violate the conservation of energy and mass. The model also disregards the intrinsic reversibility of elementary chemical steps. In other words, the model allows the reaction $A + B \to C$ without the corresponding reverse reaction $C \to A + B$. This is true, but it is necessary for modeling biochemical circuits within the cell, such as genetic regulatory networks that control the production of mRNA molecules (transcription) and of protein molecules (translation). Although no real reaction is strictly irreversible, many natural cellular reactions such as cleavage of DNA can be modeled as being effectively irreversible, or an implicit energy source (such as ATP) may be present in sufficiently high quantities to drive the reaction forward strongly. Thus, our models intrinsically assume that energy and mass are available in the form of chemical fuel (analogous to ATP, activated nucleotides, and amino acids) that is sufficient to drive reactions irreversibly and to allow the creation of new molecules. Together with the dependence of V on the total number of molecules, we envision the reaction solution as a two-dimensional puddle that grows and shrinks as it adsorbs fuel from and releases waste to the environment. This is very similar in spirit to computational models such as Turing Machines and Stack Machines that add resources (tape or stack space) as they are needed.

Another potentially unrealistic feature of the SCRN formalism is that it allows reactions of any order (any number of reactants), despite the generally accepted principle that all underlying physical chemical reactions are binary and that higher order reactions are approximations in situations with some very fast rate constants. For this reason, in our constructions we restrict ourselves to use reactions with at most two reactants. Further, it is generally accepted that Michaelis-Menten kinetics are followed for catalytic reactions. For example, the above reaction $A + B \rightarrow C + B$ should be decomposed into two reactions $A + B \rightarrow M$ and $M \rightarrow C + B$ where M is some intermediate species, but the abbreviated intermediate-free form is also allowed in the model. Another principle involving catalysts is that if a reaction can occur in the presence of a catalyst, then it can usually also occur (albeit usually much more slowly) without the catalyst. For example if $A + B \rightarrow C + B$ can occur then so can $A \rightarrow C$. Continuing in this vein, a wealth of further restrictions, each applicable in certain contexts, could arise from detailed considerations of the types of molecules being used.

Instead of focusing on these or other restrictions, we focus on the cleanest and most elegant formalism for Stochastic Chemical Reaction Networks and treat it as a programming language. We happily leave the task of accurately implementing our networks to the synthetic chemists and synthetic biologists!

2.2 Other Models of Chemical Computing

It is worth noting that several other flavors of chemical system have been shown to be Turing universal. Bennett [18] sketched a set of hypothetical enzymes that will modify a information-bearing polymer (such as DNA) so as to exactly and efficiently simulate a Turing Machine. In fact, he even analyzed the amount of energy required per computational step and argued that if the reactions are made chemically reversible and biased only slightly in the favorable direction, an arbitrarily small amount of energy per computational step can be achieved. Since then, there have been many more formal works proving that biochemical reactions that act on polymers can perform Turing-universal computation (e.g. [19, 20, 21]). In all of these studies, unlike the work presented here, there are an infinite number of distinct chemical species (polymers with different lengths and different sequences) and thus, formally, an infinite number of distinct chemical reactions. These reactions, of course, can be represented finitely using an augmented notation (e.g. "cut the polymer in the middle of any ATTGCAAT subsequence"), but as such they are not finite Stochastic Chemical Reaction Networks.

A second common way of achieving Turing universality is through compartmentalization. By having a potentially unbounded number of spatially separate compartments, each compartment can implement a finite state machine and store a fixed amount of information. Communication between compartments can be achieved by diffision of specific species, or by explicit transfer reactions. This is, for example, exploited in the Chemical Abstract Machine [22] and in Membrane Systems [23]. Note that [24], contrary to its title, only establishes that Chemical Reaction Networks appear to be able to implement feed-forward circuits (along the lines of section 3), making them empirically at least P-hard.

3 Bounded Models: Boolean Logic Circuits

A natural relation to boolean circuits leads one to expect that Stochastic Chemical Reaction Networks may well have similar computational power. For example, given a circuit built from NAND gates, we can construct a corresponding Stochastic Chemical Reaction Network by replacing each gate

$$x_k = x_i \text{ NAND } x_j$$

with the four reactions

$$\begin{aligned}
A_i + A_j &\to A_i + A_j + B_k \\
A_i + B_j &\to A_i + B_j + B_k \\
B_i + A_j &\to B_i + A_j + B_k \\
B_i + B_j &\to B_i + B_j + A_k
\end{aligned}$$

The presence of a single A_i molecule represents that $x_i = 0$, the presence of a single B_i molecule represents that $x_i = 1$, and the presence of neither indicates that x_i has not yet been computed. If the circuit has only feed-forward dependencies, it is easy to see that if one starts with a single A or B molecule for each input variable, then with probability 1 the correct species will be eventually produced for each output variable. In this sense, a Stochastic Chemical Reaction Network can deterministically compute the same function as the boolean circuit, despite the uncontrollable order in which reactions occur. Note that in this particular network, the specific rate constants can affect the speed with which the computation occurs, but do not change the eventuality.

Circuits of the same general flavor as the one above can be modified to work with mass action chemical kinetics [25, 24], showing that individual boolean logic gates can be constructed, and that they can be connected together into a circuit. This provides for efficient computation but is a non-uniform model: the number of chemical species increases with the number of gates in the circuit and thus with the size of the problem being solved.

Contrary to the limited (finite state) computational power of boolean circuits, individual Stochastic Chemical Reaction Networks are not limited by finite state spaces: there may potentially be an unbounded number of molecules of any given species. As even minimal finite-state machinery coupled with unbounded memory tends to allow for Turing-universal computation, one might speculate that the same should hold true for Stochastic Chemical Reaction Networks. If so, then Stochastic Chemical Reaction Networks would be capable of uniform computation, and predicting their long-term behavior would be undecidable.

The following sections will show that this is indeed the case. Stochastic Chemical Reaction Networks are in fact much more powerful than one might think from the simple boolean circuit approach shown above.

4 Unordered Program Models: Petri Nets and VASs

The main complicating factor when trying to "program" a Stochastic Chemical Reaction Network is that reactions occur in an uncontrollable order, making it quite difficult to guarantee a unique outcome for a non-trivial computation. Stochastic Chemical Reaction Network computations that *are* arranged so as to guarantee a unique outcome will be called *confluent* computations.

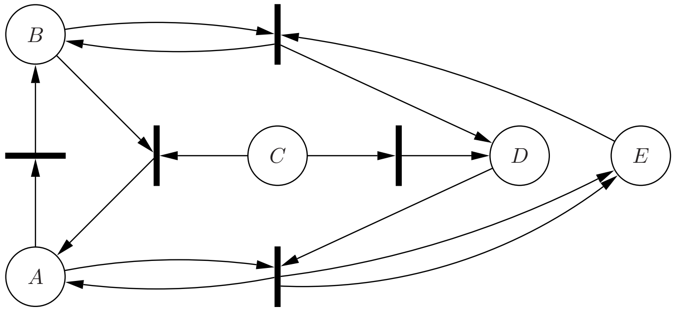
We can find clues regarding how to program such systems, including relevant theorems, by examining the related computational models mentioned above and shown in figure 1. The differences between the models are minor, amounting mostly just to different interpretations or viewpoints of the same underlying fundamental process. For example, consider Petri nets [26], as shown in figure 1(b). In this model a network consists of a directed bipartite graph, having connections between *places* (shown as circles) and *transitions* (shown as black bars). The state consists of a non-negative number of *tokens* at each place, and a new state is achieved by the *firing* of a transition. When a transition fires, it consumes one token from the incident place for each incoming edge, and produces one token into the incident place for each outgoing edge (there is no difference between the two sides of the black bar). Thus, a transition is *enabled* only if there are enough tokens in the input places. In any given state, there are typically many transitions that could fire. Which one fires first is intentionally left unspecified: the theory of Petri nets addresses exactly the question of how to analyze asynchronous events. If the system uses rate constants as in equation 1 for each transition (in which case the model is a type of stochastic Petri net), the model is formally identical to Stochastic Chemical Reaction Networks: each place corresponds to a molecular species (the number of tokens is the number of molecules) and each transition corresponds to a reaction [27].

A closely related model, Vector Addition Systems (VASs), was developed and studied by Karp and Miller [10] for analyzing asynchronous parallel processes. Here, questions concern walks through an m dimensional integer lattice, where each step must be one of d given vectors $\mathbf{V}_\alpha \in \mathbb{Z}^m$, and each point in the walk must have no negative coordinates. Whether it is possible to walk from a point x to a point y (the *reachability* question) is in fact decidable [28]. It is also decidable whether it is possible for a walk to enter a linearly-defined subregion [29] – a special case is whether the i^{th} component of the point ever becomes non-zero (the *producibility* question).

The correspondence between Vector Addition Systems, Stochastic Chemical Reaction Networks and Petri nets is direct. First consider chemical reactions in which no species occurs both on the left side (as a reactant) and on the right side (as a product) – i.e. reactions that have no *instantaneous catalysts*. When such a reaction α occurs, the state of the Stochastic Chemical Reaction Network, represented as a vector, changes by addition of the vector $\mathbf{p}_\alpha - \mathbf{r}_\alpha$. Thus the trajectory of states is a walk through $\mathbb{Z}^m$ wherein each step is any of d given vectors, subject to the inequalities requiring that the number a)

$$\begin{aligned} A &\to B \\ C &\to D \\ B+C &\to A \end{aligned} \qquad \begin{aligned} A+D &\to A+2E \\ B+E &\to B+D \end{aligned}$$

b)



c)

$$\begin{array}{c|ccccccc|c} & A & B & C & D & E & F & G & \\ \hline \langle & -1 & 1 & 0 & 0 & 0 & 0 & 0 & \rangle \\ \langle & 0 & 0 & -1 & 1 & 0 & 0 & 0 & \rangle \\ \langle & 1 & -1 & -1 & 0 & 0 & 0 & 0 & \rangle \\ \langle & -1 & 0 & 0 & -1 & 0 & 1 & 0 & \rangle \\ \langle & 1 & 0 & 0 & 0 & 2 & -1 & 0 & \rangle \\ \langle & 0 & -1 & 0 & 0 & -1 & 0 & 1 & \rangle \\ \langle & 0 & 1 & 0 & 1 & 0 & 0 & -1 & \rangle \end{array}$$

d)

$$\frac{3}{2} \qquad \frac{7}{5} \qquad \frac{2}{15} \qquad \frac{13}{14} \qquad \frac{242}{13} \qquad \frac{17}{33} \qquad \frac{21}{17}$$

Figure 1. Four representations of the same computation. Starting with 1 A and n C's, the maximum number of D's that can be produced is 2^n. (a) A Stochastic Chemical Reaction Network. (b) A Petri net. Each circle corresponds to a *place* (a molecular species), and each black bar corresponds to a *transition* (a reaction). (c) A Vector Addition System. Note that dimensions F and G must be added to the Vector Addition System to capture the two reactions that are catalyzed by A and B. (d) A Fractran program. The numerators correspond to the reaction products, and the denominators correspond to the reactants. The first seven prime numbers are used here in correspondence to the letters A through G in the other examples. As in the previous example, F (13) and G (17) must be introduced, here to avoid unreduced fractions for the catalyzed reactions.

of molecules of each species remain non-negative, thus restricting the walk to the non-negative orthant.

Karp and Miller's decidability results for VASs [10] directly imply the decidability of the question of whether a catalyst-free Stochastic Chemical Reaction Network can possibly produce a given target molecule (the *producability* question again). As a consequence, confluent computation by Stochastic Chemical Reaction Networks cannot be Turing universal, since questions such as whether the YES output molecule or the NO output molecule will be produced are decidable. The restriction to catalyst-free reactions is inessential here: each catalytic reaction can be replaced by two new reactions involving a new molecular species (an "intermediate state", see figure 1(c)), after which all reachability and producibility questions (not involving the new species) are identical for the catalyst-free and the catalyst-containing networks.

4.1 Gate Implementability

The initial path leading the authors to consider the computational power of Stochastic Chemical Reaction Networks came from a surprisingly unrelated topic. We were considering the general question of whether circuits constructed from available gate types are able to implement a desired target function. We call this the *gate implementability* question. The terms *gate* and *function* will be used interchangeably here.

It has been known since the time of Post [30] that, given a set of functions of boolean values, only a finite number of tests need to be done to know whether a particular target function can or cannot be implemented by them, if function values, once produced, can be used repeatedly (in other words, if *fan-out* is available). However, in situations where values cannot be used repeatedly (as is the case for example in quantum computation), the implementability question becomes much less clear. Indeed, if the analogous questions are asked for circuits built of *relations*, rather than *functions*, then the ability to reuse values makes this question become decidable, whereas it is undecidable if values, once produced, can only be used once [31].

It is natural to wonder, if fan-out is *not* available, might the gate implementability question become *un*decidable, as it did for relations?

First of all, we have to be clear about what we mean by "circuits without fan-out." From a feed-forward point of view, a fan-out node in a circuit is a device with one input and two outputs, and both outputs equal the input. So, we will be generous and expand the definition of "function" to allow multiple outputs. (If we do not do this, then all circuits must be trees, and it becomes difficult to implement anything at all, since in contrast with formulas, inputs cannot be used at more than one leaf of the tree.) We will define the outputs of a feed-forward circuit to be all of the output wires which have not been fed into some other gate, and the inputs are of course all the input wires which are not produced as the output of another gate.

This gives us an implementability question for feed-forward circuits that is comparable to the implementability question for relations. As with relations, the availability of fan-out makes the question easily decidable: Simply iteratively expand the set of implementable functions, starting with the inputs and the given functions. However, without fan-out available, the situation is not quite so easy.

4.2 Gate Implementability is Equivalent to Reachability in Stochastic Chemical Reaction Networks

In this section, we will show that any gate implementability question can in fact be reduced to a reachability question for a chemical reaction network, and vice versa. Intuitively, the idea is that the wires in the circuit correspond to molecules, the gates in the circuit correspond to reactions, the designer of the circuit corresponds to the source of randomness in the Stochastic Chemical Reaction Network, and the ability to implement a given function corresponds to the reachability question for the Stochastic Chemical Reaction Network.

The idea for the forward direction is that we consider all possible inputs to the circuit simultaneously. Since we know what we are trying to implement, we know how many inputs there are, and what the possible values for each input are, and thus we know exactly how many distinct possible states the entire circuit can be in. For example, if there are five boolean inputs, then there are $2^5 = 32$ possible states for the circuit (one for each possible combination of values on the input wires), and every wire in the circuit can have its behavior described by a vector of length 32, giving the value of that wire in each of the 32 possible states the circuit might be in. In this example, the five inputs to the circuit would be described by the following vectors:

$\langle 0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1\rangle$
$\langle 0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1\rangle$
$\langle 0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1\rangle$
$\langle 0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1\rangle$
$\langle 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1\rangle$

The vector describing an output of a gate is easily calculated from the vectors for the inputs.

The corresponding chemical reaction network will be designed to have one species for each possible vector. (In the example above, there would be 2^{32} species.) Then, each gate available in the implementability question is converted into a long list of chemical reactions: For each possible combination of input vectors to the gate, we provide a chemical reaction which takes those species as reactants and produces the appropriate species (those corresponding to the correct outputs of the gate for these inputs) as products.

The starting state for the chemical reaction network is one molecule of each of the species used as inputs (in the example above, recalling that each vector is a species, the starting state would be the five listed vectors), and the target state for the reachability question is simply the corresponding set of output vector species for the target gate in the implementability question. It is clear from the design that the target state is reachable in the chemical reaction network if and only if the target gate is implementable in the implementability question.

Now we will show the other direction, that any reachability question for a chemical reaction network can be reduced to an implementability question for gates without fan-out.

The idea for this direction is to design some gates that can only be usefully combined by following exactly the reactions of the given network. The alphabet of values used by the gates will consist of one symbol for each of the chemical species, plus an extra symbol "ϵ," which we will think of as an error symbol. There will be one gate per reaction, plus one extra gate. Each reaction will be converted into a gate with as many inputs as reactants and as many outputs as products. For example, the reaction $A + 2B \to C + D$ would become a gate with 3 inputs and 2 outputs, and the computation performed by the gate is almost trivial: It outputs ϵ on every output, *unless* its inputs are $\langle A, B, B\rangle$, in which case it outputs $\langle C, D\rangle$. Other reactions are similarly converted. We also provide an extra gate with two inputs and two outputs, which is a two-wire identity gate, except that if *either* input is ϵ, then *both* outputs are ϵ. Otherwise the first output matches the first input, and the second output matches the second input. The purpose of this gate is to allow the error symbol ϵ to spread from one wire to another, as we will see shortly.

The initial state and target state for the reachability question then become the inputs and outputs of the target gate, and again every other possible input should lead to all outputs being ϵ.

Any satisfactory solution to this implementability question clearly corresponds to a partially ordered sequence of reactions that demonstrates a positive answer to the reachability question. Conversely, any sequence of reactions reaching the target state of the reachability question can be directly converted into a circuit of gates that is *almost* guaranteed to implement the target gate. The only potential problem is that if the input given to the circuit differs just slightly from the intended input, then some of the gates will still be getting exactly the inputs that were intended, and for some circuits, it may not be the case that *all* outputs are ϵ, but rather just some subset of them. It is for this reason that we supplied the extra "error propagating" gate. If necessary, this gate can be used many times at the end of a circuit ($2n$-3 times for a circuit with n outputs) to ensure that if any outputs are ϵ, then all outputs must be ϵ. Clearly the availability of this gate will not otherwise affect the ability to simulate the sequence of reactions. Thus, the answer to the gate implementability question will match exactly the answer to the chemical reachability question.

5 Almost Universal: Primitive Recursive Computation

It has long been known that certain questions about whether a Petri net "might do X" are decidable, where typical values of X are, in the language of Stochastic Chemical Reaction Networks, "keep having reactions forever" or "grow without bound" or "reach a certain state" or "produce at least some given quantities of given species" [10, 28, 9]. These results carry over directly to Stochastic Chemical Reaction Networks so long as the question does not ask about the *probability* of X happening, but only about the *possibility* of it happening (i.e. only about whether the probability of X is zero vs. non-zero).

As mentioned in section 4, confluent computation by Stochastic Chemical Reaction Networks can only implement decidable decision problems. Thus, for questions about the output of a Stochastic Chemical Reaction Network (given by some final quantity of the output species) to have any hope of being undecidable, the output *must* be probabilistic in nature. We will examine questions of probability in section 6; here we restrict ourselves to questions of possibility.

Although the questions of possibility listed above are known to be decidable, their complexity is sometimes not so clear. The complexity of the problem for X="grow without bound" is known to be doubly exponential [32], but the complexity of the problem for X="reach a certain state" has been an open problem for decades [9].

Even though double exponential complexity sounds quite complex, the complexity of these types of problems can in fact be far greater. Some suspect that the reachability problem (i.e., X="reach a certain state") may have complexity comparable to primitive recursive functions, which are so powerful that few natural non-primitive recursive functions are known.

In section 5.3 we present examples of problems whose complexity does exactly match the power of primitive recursive functions. Specifically, if X = "have a molecule of S_1 present when attaining the maximum possible amount of S_2," or X = "have a molecule of S_1 present after taking the longest possible (over all sequences) sequence of reactions." These questions are equivalent in power to primitive recursively defined predicates, where the number of primitive recursive functions used to recursively build up the predicate is on the order of the number of molecular species in the Stochastic Chemical Reaction Network, and the input to the predicate corresponds to the initial state of the Stochastic Chemical Reaction Network.

To show that such question are no more powerful than primitive recursive functions, in section 5.2 we show that for any Stochastic Chemical Reaction Network, it is possible to define a primitive recursive function which can return the amount of S_1 that is produced by whichever sequence of reactions leads to the largest possible amount of S_2. Our proof, while far from straightforward, is much simpler than previous similar proofs (which used results on bounds for solutions to bounded versions of Hilbert's tenth problem), since it gives an explicitly primitive recursive formula bounding the size of the tree of all possible runs of the Stochastic Chemical Reaction Network. The bulk of the proof lies in defining this bounding function and proving that it indeed bounds the depth of the tree. This bound enables the definition of a primitive recursive function which analyzes the entire tree, explicitly finding the run with the largest amount of S_2 and returning the corresponding amount of S_1.

5.1 Primitive Recursive Functions

Primitive Recursive Functions were first investigated in the 1920's, starting with Skolem [33], who pointed out that many standard functions on non-negative integers can be defined using just function composition and recursion, starting with just the successor function. This surprising fact is illustrated in figure 2, which shows how functions can be built up in this way, including for example a function that will tell you whether a number is prime or not.

		Recursive Definition	
Function	Name	if $n = 0$	if $n = m + 1$
$S(n) = n + 1$	successor		
$A(n, a) = n + a$	addition	a	$S(A(m, a))$
$M(n, a) = n \times a$	multiplication	0	$A(a, M(m, a))$
$E(n, a) = a^n$	exponentiation	1	$M(a, E(m, a))$
$V(n) = \text{sign}(n)$	positivity	0	1
$P(n) = n - 1$	predecessor	0	m
$D(a, n) = a - n$	subtraction	a	$P(D(a, m))$
$R(n, a) = n \bmod a$	remainder	0	$\left\{\begin{array}{l} M(S(R(m, a)), \\ \quad V(D(P(a), \\ \qquad R(m, a)))) \end{array}\right.$
$C(n, a) = \prod_{i=2}^{n+1} i \bmod a$	mod product	1	$\left\{\begin{array}{l} M(C(m, a), \\ \quad R(a, S(S(m)))) \end{array}\right.$
$Z(n) = 1$ if n is prime	primality	0	$\left\{\begin{array}{l} V(M(m, \\ \quad C(P(m), S(m)))) \end{array}\right.$

Figure 2. Examples of Primitive Recursive Functions. Starting with only the successor function, other functions can be built up one by one using a simple form of recursion. Where the function being defined is used in its own recursive definition, the rule is that it must have exactly the same arguments but with n replaced by m.

The wide range of functions that could be defined in this way led logicians to wonder whether all mathematical functions could be defined in this way, or at least all those functions for which there exists a deterministic algorithm for calculating the value. Recall that this was long before people had ever written algorithms for electronic computers, before Gödel's famous incompleteness theorem [34] and before Turing Machines [35], in short, before people had figured out any satisfactory way of standardizing or formalizing the process of mathematical calculation. Perhaps this was the way?

It turned out that this was not the way. In 1928, Ackermann [36] showed that there is a limit to how fast a Primitive Recursive Function can grow (depending on how many other functions are used to help define it), and there turn out to exist simple deterministic algorithms for calculating functions that grow even faster than this limit, as shown in figure 3. Thus, the world of Primitive Recursive Functions is not large enough to encompass all mathematical calculations.

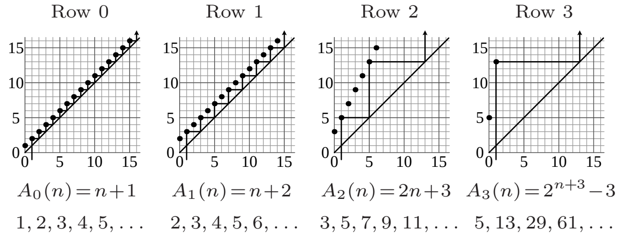


Figure 3. An illustration of the Ackermann function. The Ackermann function $A_i(n)$ is a function of two variables, i and n. The i^{th} row of the Ackermann function, A_i, can be constructed visually from the previous row A_{i-1} as shown: A zig-zag line starts going up at $x = 1$, and bounces back and forth between the function values (shown as dots) and the line $x = y$. The function values hit by the zig-zag line become the entries for the next row. The formal definition is $A_0 = S,\ A_{i+1}(0) = A_i(1),\ A_{i+1}(m+1) = A_i(A_{i+1}(m))$. Although each row is a Primitive Recursive Function, the diagonal $f(n) = A_n(n)$ grows faster than any Primitive Recursive Function in the same sense that 2^n grows faster than any polynomial.

Not long after Ackermann's paper made it clear that Primitive Recursive Functions were merely a strict subset of the functions that can be calculated, Herbrand in 1931 [37] and Gödel in 1934 [38] defined General Recursive Functions, which in 1936 were argued by both Church [39] and Turing [35] to correspond exactly to the set of all functions that can possibly be calculated in any algorithmic way. This argument was accepted by most people, and is now well known as the Church-Turing Thesis.

A major distinction between the General Recursive Functions and the Primitive Recursive Functions is that the latter (and also Ackermann's function) are defined for all inputs—that is to say, computation eventually halts and produces an output, no matter what the input is—whereas the former include additional functions, some of which halt only for some inputs. Figuring out which General Recursive Functions halt for which input is known as the Halting Problem, and it is formally undecidable: there is no General Recursive Function that will always correctly determine whether a given algorithm halts.

While most people turned their attention at this point to General Recursive Functions, Rózsa Péter [40] continued to develop the theory of Primitive Recursive Functions, treating them not as a historical mistake, but as an opportunity for study. Her work makes it clear that the following definition is an equivalent way to define Primitive Recursive Functions:

Definition: Primitive Recursive Functions are exactly those functions which can be computed by a Turing Machine in time bounded by *some* row of the Ackermann function.

This definition makes it evident that just about every algorithm ever used for practical calculation is in fact Primitive Recursive, since most rows of the Ackermann function grow far faster than the time required for any practical calculation.

Although Péter's work showed that many seemingly different definitions all lead to this same set of functions, the definitions were rather abstractly mathematical in nature, none of them corresponding to what we would think of today as a fundamental computational model like a Turing Machine. So it is interesting that Primitive Recursive Functions arise here in relation to Stochastic Chemical Reaction Networks, a fundamentally reality-based model.

5.2 A Primitive Recursive Bound on the Depth of the Tree of Reachable States

Theorem 1. *Given two states $\mathcal{A}$ and $\mathcal{B}$, in order to determine whether starting from $\mathcal{A}$ a Stochastic Chemical Reaction Network can reach a state with at least as many molecules as $\mathcal{B}$ is decidable and requires a search tree of size bounded by a primitive recursive function of the number of molecules of each species and the stoichiometric coefficients of the reactants.*

Here we present the details of our proof that the tree of possible execution paths of a Stochastic Chemical Reaction Network has depth bounded by a primitive recursive function whose "degree" is on the order of the number of species in the Stochastic Chemical Reaction Network.

For those familiar with the subject, the algorithm is nearly identical to Karp and Miller's [10] but the rest of the proof is much more direct than comparable previous proofs which relate other questions about the tree to primitive recursive functions. See also [15].

The Algorithm

In this section we will present an algorithm for finding which species can be produced and which cannot. That is, it will find out whether any reachable states have non-zero levels of any species of interest. In fact, it will do slightly more: For any given set of molecule quantities (such as $(10A, 3B, \ldots)$), the algorithm can find out whether or not it is possible to reach any state that has at least these levels of these species.

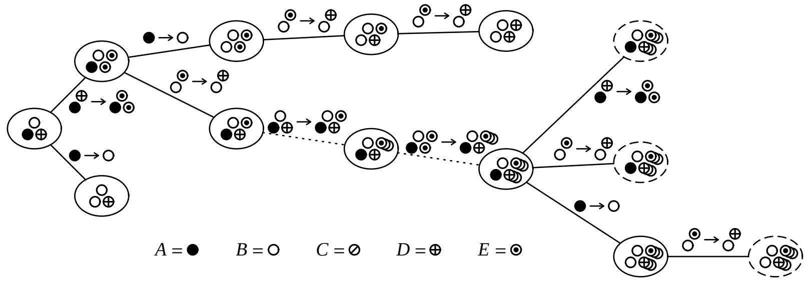


Figure 4. The search tree for the system of figure 1, starting on the left with state (A, B, D). Solid lines represent single reactions, while dotted lines represent any number of further repetitions of a completed cycle that purely increases a molecular quantity, leading to the attainability of arbitrarily large quantities of that species, shown for example as ⊙. The dashed circles are repeats of previous states and thus do not require further exploration even if further reactions are possible.

In this example, the search tree is finite. Must this always be the case? If so, then there are no undecidable questions among questions which can be answered by scanning the full search tree. This section shows that the search tree is finite, and indeed boundable by a primitive recursive function.

The algorithm is simply to search through the full tree of all possible reaction sequences, using a couple of simple tricks to try to avoid getting stuck in infinite loops.

If state $\mathcal{B}$ has at least as many molecules of each species as state $\mathcal{A}$ does, then we will say that $\mathcal{B} \geq \mathcal{A}$. On the other hand, if $\mathcal{B}$ has more of some species and less of others than $\mathcal{A}$ has, we say that $\mathcal{B}$ and $\mathcal{A}$ are incomparable: $\mathcal{A} \not\geq \mathcal{B}$ and $\mathcal{B} \not\geq \mathcal{A}$.

A fundamental observation is that if the system is in state $\mathcal{A}$ at some point, and then later it is in state $\mathcal{B}$, and $\mathcal{B} \geq \mathcal{A}$, then the sequence of reactions that led from $\mathcal{A}$ to $\mathcal{B}$ may be repeated arbitrarily many times before continuing. This would appear to be a serious obstacle to exhaustively searching the space of reachable states, but in fact it will be the key to bounding the search. When this happens, we can consider two cases: $\mathcal{B} = \mathcal{A}$ or $\mathcal{B} > \mathcal{A}$.

If $\mathcal{B} = \mathcal{A}$, then this sequence of reactions leading from $\mathcal{A}$ to $\mathcal{B}$ had no effect, and may be omitted entirely. In particular, it is clear that the shortest sequence of reactions leading from the initial state of the system to any particular final state will not visit any state more than once. Thus, no possibilities will be missed if the search tree is simply pruned at any point where a previous state is repeated.

On the other hand, if $\mathcal{B} > \mathcal{A}$, that is, if $\mathcal{B}$ has strictly more of some species than the earlier state $\mathcal{A}$ had, then by repeating this sequence of reactions, an arbitrarily large amount of those species may be produced. We will call such species *freely generatable* after the sequence of reactions from $\mathcal{A}$ to $\mathcal{B}$ has occurred. If at any later point in the calculation, some potential reaction is not possible because one of the freely generatable species has run out, we can simply retroactively assume that more repeats of the sequence from $\mathcal{A}$ to $\mathcal{B}$ were performed back at the time when that species became freely generatable, and this will allow the potential reaction to proceed after all. For this reason, when a species becomes freely generatable, it may effectively be removed from the problem statement, reducing the problem to a simpler problem. So although the search tree cannot be pruned when $\mathcal{B}$ is reached, the subtree beyond that point corresponds to searching the space of a simpler problem, in which a further repetition of the reaction sequence leading from $\mathcal{A}$ to $\mathcal{B}$ would indeed lead to pruning, since states $\mathcal{A}$ and $\mathcal{B}$ are equal in the reduced problem. The algorithm therefore specifies the quantity of a freely generatable species as ∞, a value which is considered larger than any other value, and which is unchanged by the addition or removal of molecules.

It remains to show that the search tree is always finite, and thus this algorithm will always terminate.

The Data Structure

Now we will define a data structure whose purpose will be to help us define the bound in the next section.

At each point in the search tree, there is a (usually infinite) set $\mathbb{S}$ of all states $\mathcal{S}$ satisfying $\mathcal{S} \ngeq \mathcal{A}$ for every $\mathcal{A}$ which is an ancestor of that point in the search tree. We will call this set of states $\mathbb{S}$ the *remaining states* for that point in the search tree, because these are the states which, if reached on the next step, will not lead to pruning or simplification. Our proof will examine this set of states and use the structure of this set to provide a bound on how much deeper the search tree can be.

For any given point in the search tree, we represent the set of remaining states by lists $\mathbb{L}_i$, with each entry in list $\mathbb{L}_i$ representing an i-dimensional region of remaining states, specified by $n-i$ integers (specifying quantities of $n-i$ of the n species). The union of all regions from all lists exactly yields the set of remaining states for the given point in the search tree.

When a reaction takes the system to a new state (taking the search to a new point in the search tree), the lists are modified by eliminating each list entry which represents a region containing any state greater than or equal to the new state. Each eliminated entry is replaced by new entries in the list of next lower index. The new entries are found by considering all regions of dimension one less than the old region, lying within the old region, with a previously unspecified coordinate now specified as some particular integer k, with $0 \leq k < m$, where m is the number of molecules present, in the new state, of the species corresponding to the dimension now being specified. In general, this might lead to some redundancy, if some of the new regions lie inside other existing regions, but we will not need to worry about this.

$$\begin{array}{ccc}
\underline{\qquad\mathbb{L}_4\qquad} & & \underline{\qquad\mathbb{L}_3\qquad} \\
 & & \vdots \\
 & & (2,0,\cdot,5,\cdot,3,\cdot) \\
\vdots & & (2,1,\cdot,5,\cdot,3,\cdot) \\
 & & (2,2,\cdot,5,\cdot,3,\cdot) \\
 & & (2,3,\cdot,5,\cdot,3,\cdot) \\
(2,\cdot,\cdot,5,\cdot,3,\cdot) & \rightsquigarrow & (2,\cdot,0,5,\cdot,3,\cdot) \\
 & & (2,\cdot,\cdot,5,0,3,\cdot) \\
 & & (2,\cdot,\cdot,5,1,3,\cdot) \\
\vdots & & (2,\cdot,\cdot,5,2,3,\cdot) \\
 & & \vdots
\end{array}$$

Figure 5. Left: An example of a possible entry in list $\mathbb{L}_4$, for a system with 7 species. Right: All the entries that will be added to list $\mathbb{L}_3$ to replace the entry on the left, if the system arrives at state $(2,4,1,3,3,3,0)$. The union of the new 3-dimensional regions is precisely that portion of the old 4-dimensional region which is $\not\geq$ the new state.

The lists for the initial state of the system are created similarly, with the "old" region taken to be the full n-dimensional space, just a single entry in list $\mathbb{L}_n$. Thus, a system started in state $(q_1, q_2, ..., q_n)$, where q_i is the quantity of the i^{th} species, will start with $\sum_i q_i$ entries in list $\mathbb{L}_{n-1}$. Similarly, whenever an entry in list $\mathbb{L}_i$ is replaced by new entries in list $\mathbb{L}_{i-1}$ due to a new state $(q_1, q_2, ..., q_n)$, the number of new entries will be $\sum_{i\in P} q_i$, where P is the set of species whose quantity is unspecified in the old entry.

If the i^{th} species becomes freely generated, all list entries in all lists will have their i^{th} entry changed to be specified as ∞, which may move some of them to the list of next lower index: Since ∞ is treated by the lists as a specified quantity, any list entry which previously did not specify the quantity of the i^{th} species will now have one fewer unspecified quantities, and will thus move to the list of next lower index.

It remains to show that these lists eventually get exhausted as the algorithm progresses deeper into the tree. For readers familiar with the game of Chomp [41], this process is quite similar to Chomp on infinite boards.

The Bound

To each point in the search tree, with its state and its lists, we can assign a positive integer as described below. We will see that regardless of which reaction is performed at the next step, the positive integer assigned to the ensuing point in the search tree will always be less than the positive integer assigned to the current point. Since the positive integer strictly decreases with depth, it is in fact a bound on the depth.

The integer for a given state $\mathcal{A}$ and lists $\mathbb{L}_i$ is defined for a system with n species in the following non-trivial way:

$$\mathfrak{B}(\mathcal{A}, \mathbb{L}) = f_{n-1}^{|\mathbb{L}_{n-1}|}(f_{n-2}^{|\mathbb{L}_{n-2}|}(...(f_1^{|\mathbb{L}_1|}(f_0^{|\mathbb{L}_0|+m\cdot r+q_{\max}}(0)))...))$$

where r is the number of non-freely generatable species, $q_{\max}$ is the largest number of molecules present of any of those r species, and m, a constant, is one more than the maximum coefficient appearing on the right hand side of any reaction.

The functions f_i are defined as follows:

$$f_i(x) = f_{i-1}^{i\cdot x+m}(x)$$
$$f_0(x) = x+1$$

These definitions are not meant to capture intuitive notions of any meaningful functions, but rather are designed to (a) be explicitly primitive recursive, and (b) be of a form that enables the necessary proof steps below to work.

In these definitions, the exponents on the functions denote multiple applications of the function, so for example $f_8^3(x) = f_8(f_8(f_8(x)))$. Each f_i, as well as $\mathfrak{B}$, is a Primitive Recursive Function, since it is easy to define repeated application of a function: Given a function $g(x)$, we can define $h(n,x) = g^n(x)$ using the recursive definition $h(0,x) = x,\ \ h(m+1,x) = g(h(m,x))$.

It is straightforward to show that the functions $f_i(x)$ are strictly increasing in x, and that $f_{i+1}(x) > f_i(x)$. Thus, if the exponents appearing within the definition of $\mathfrak{B}$ are in any way reduced or shifted to the right, $\mathfrak{B}$ will decrease.

This can be used to show that regardless of whether a reaction leads to a remaining state or leads to a new freely generatable species, $\mathfrak{B}$ will always decrease.

If a reaction results in one or more freely generatable species, then some parts of the exponents may shift to the right, and r will decrease. In the exponent of f_0, the decrease of r will more than make up for any increase in $q_{\max}$ (by the definition of m), so $\mathfrak{B}$ will decrease as promised.

If a reaction leads to a remaining state, then one or more list entries will be replaced by other entries. Each i-dimensional entry to be removed will be replaced by $\sum_{j\in P} q_j$ entries that are $(i-1)$-dimensional. This number of new entries is no more than $i \cdot q_{\max}$, since P, the set of species of unspecified quantity, is of size i. So the exponent of f_i is reduced by 1 while the exponent of f_{i-1} increases by at most $i\cdot q_{\max}$. In the formula for $\mathfrak{B}$, then, an f_i gets replaced with $f_{i-1}^{i\cdot q_{\max}}$, and then this exponent is possibly reduced. But the original f_i was equivalent (by definition) to $f_{i-1}^{i\cdot x+m}$, where x is the full argument (which must be at least $q_{\max}$, since $q_{\max}$ appears in the exponent of f_0), so even just the $f_{i-1}^{i\cdot x}$ portion was bigger than the replacement, and the f_{i-1}^m portion more than compensates for any increase in the exponent of f_0 due to any change in $q_{\max}$. The total effect is therefore again a decrease in $\mathfrak{B}$.

Thus we have finished showing that $\mathfrak{B}$, a primitive recursive function of the initial state, bounds the depth of the search tree. Thus both the depth of the tree, and its total size (being at most exponential in the depth), are not only finite but bounded by a primitive recursive function of the initial state. In the next section we will see examples which cannot be bounded by anything smaller than this.

5.3 The Max-Path Problem

We have shown that any Chemical Reaction System can be analyzed by Primitive Recursive Functions, but the reverse question is also interesting: Can any Primitive Recursive Function be calculated by a Chemical Reaction System? This question raises conceptual issues not present in the forward question, since Chemical Reaction Systems are inherently nondeterministic, it being unspecified at each step which reaction should occur next. Thus one must choose how to define which of the possible sequences of reactions should be considered as leading to the correct (or incorrect) calculation of the function. If one chooses, say, the longest possible sequence of reactions (the deepest leaf in the search tree), or the sequence that leads to the most molecules being produced (either of all species put together, or of some particular species), then it is indeed possible to calculate any Primitive Recursive Function, where the input and output are given as numbers of molecules of certain species. These choices provide an exact equivalence in power between Chemical Reaction Systems and Primitive Recursive Functions. Admittedly, this is not a practically useful notion of calculation by a SCRN—if I have the chemicals in my lab, how do I perform an experiment that indicates the output of the computation?—but it does help clarify the boundary between decidable and undecidable questions about SCRNs.

Theorem 2. *For any primitive recursive function f, a Stochastic Chemical Reaction Network can be designed with special species S_{in}, S_{out}, and S_{max} computing f as follows. Starting with n molecules of S_{in} (and some fixed number of molecules of other species) the reachable state with the maximal amount of S_{max} will have exactly $f(n)$ molecules of S_{out}.*

We prove this theorem with a construction. We begin by presenting, for any chosen fixed integer i, a SCRN that Max-Path-computes the i^{th} row of the Ackermann function. This simple example of Max-Path "computing" by SCRNs is enlightening in and of itself, but more importantly, it plays a crucial role in our general construction, where it is used to bound the number of steps taken by a Register Machine that computes the Primitive Recursive Function in question.

SCRNs for Rows of the Ackermann Function

Figure 1(a) gives a SCRN that computes 2^n. This example can be generalized to compute any chosen row of the Ackermann function. Since the Ackermann function grows faster than any primitive recursive function, the full Ackermann function cannot be Max-Path computed by any single SCRN; using a different SCRN for each row of the function is the best we could hope to do.

$$\begin{array}{ccc}
 & Y_0 \rightarrow X + W & \\
 & Y_1 \rightarrow X + Y_0 & \\
X + Y_2 \rightarrow Z_1 + Y_2 & Y_2 \rightarrow X + Y_1 & W + Z_1 \rightarrow Y_1 \\
X + Y_3 \rightarrow Z_2 + Y_3 & Y_3 \rightarrow X + Y_2 & W + Z_2 \rightarrow Y_2 \\
\vdots & \vdots & \vdots \\
X + Y_i \rightarrow Z_{i-1} + Y_i & Y_i \rightarrow X + Y_{i-1} & W + Z_{i-1} \rightarrow Y_{i-1}
\end{array}$$

Figure 6. A Chemical Reaction System for nondeterministically computing entries for the first i rows of the Ackermann function using $2i + 2$ species. However, as shown in this paper, no Chemical Reaction System with a finite number of species is able to compute *all* rows of the Ackermann function. To compute an entry in the i^{th} row, $A_i(n)$, start this Chemical Reaction System in the state (Y_i, nX). Then, the maximum number of molecules of X that can be produced is exactly $A_i(n)$, achievable by always choosing the first possible reaction from the list.

We prove that the construction works by proving the two sides: First, we prove that starting in state (Y_i, nX) we *can* produce $A_i(n)$ X's. Second, we prove that no more than $A_i(n)$ X's can be produced.

We prove the first part by induction on row index i. Our inductive assumption will be that from (Y_{i-1}, nX) we can get to $(W, A_{i-1}(n)X)$. (The base case is easy to confirm.) Now starting with (Y_i, nX), we first convert all X's to Z_{i-1}'s by reactions in the first column. Then through a reaction in the second column we reach (Y_{i-1}, X, nZ_{i-1}), and the inductive assumption allows us to reach $(W, A_{i-1}(1)X, nZ_{i-1})$. Now we repeatedly use the first possible reaction in the third column, producing $(Y_{i-1},\ \text{same}\ X,\ \text{one fewer}\ Z_{i-1})$, followed by the inductive assumption, producing $(W, A_{i-1}(\text{previous}\ X),\ \text{same}\ Z_{i-1})$, until we can no longer use that reaction in the third column. At this point, we have produced

$$(W, \underbrace{A_{i-1}(\cdots(A_{i-1}(}_{n+1\ \text{times}}1))X) = (W, A_i(n)X).$$

This shows that it is indeed possible to produce $A_i(n)$ X's.

Now we argue that no more than $A_i(n)$ X's can be produced from (Y_i, nX). The proof consists of showing that the expression

$$T_i(T_{i-1}(\cdots(T_2(A_1^{\#Y_1}(A_0^{\#Y_0}(\#X)))))) \text{ where } T_i(m) = A_{i-1}^{\#Z_{i-1}}(A_i^{\#Y_i}(m))$$

does not increase no matter which reaction is performed, assuming there is a exactly one of the Y's or W present (an invariant enforced by our system).

Since the initial value of this expression is $A_i(n)$ when starting in (Y_i, nX), we would then know that no more than $A_i(n)$ X's can be produced.

The following two lemmas are useful.

Lemma 1 $A_i(A_j(m)) \geq A_j(A_i(m))$ *for* $i > j$

Proof. If $i > j$, then $A_j(A_i(m)) \leq A_{i-1}(A_i(m)) = A_i(m+1) \leq A_i(A_j(m))$.□

Lemma 2 $A_i(m) \geq A_{i-1}^2(m)$

Proof. First we expand $A_i(n) = A_{i-1}(A_{i-1}(\cdots(1)))$ where the composition occurs $n+1$ times. Except in edge cases, the lemma is then equivalent to showing that $A_{i-1}(\cdots(1)) \geq n$ where the composition occurs $n-1$ times. This inequality holds because applying the Ackermann function increases the argument by at least one. □

Now we will use these lemmas to show that each of the three types of reactions (in the three columns) does not increase our expression.

Consider the reaction $X + Y_i \rightarrow Z_{i-1} + Y_i$. The reaction takes subexpression $T_i(T_{i-1}(\cdots(\#X))) = A_{i-1}^{\#Z_{i-1}}(A_i(T_{i-1}(\cdots(\#X))))$ to subexpression $A_{i-1}^{\#Z_{i-1}+1}(A_i(T_{i-1}(\cdots(\#X-1))))$. The start subexpression is equal to

$$A_{i-1}^{\#Z_{i-1}}(A_i(T_{i-1}(A_0(\cdots(\#X-1))))) \geq A_{i-1}^{\#Z_{i-1}}(A_i(A_0(T_{i-1}(\cdots(\#X-1)))))$$

using the first lemma. Since $A_{i-1}(A_i(m-1)) = A_i(m)$, this expression equals the end subexpression.

Now consider the reaction $Y_i \rightarrow X + Y_{i-1}$. It takes the subexpression $A_i(A_{i-2}^{\#Z_{i-2}}(T_{i-2}(\cdots(\#X))))$ to the subexpression

$$A_{i-2}^{\#Z_{i-2}}A_{i-1}(T_{i-2}(\cdots(\#X+1)))) \leq A_{i-1}^2(A_{i-2}^{\#Z_{i-2}}(T_{i-2}(\cdots(\#X))))$$

by applications of the first lemma. This is not more than the original subexpression by the second lemma.

Lastly consider the reaction $W + Z_i \rightarrow Y_i$. This reaction takes subexpression $A_i^{\#Z_i}(A_{i-1}^{\#Z_{i-1}}(T_{i-1}(\cdots(\#X))))$ to $A_i^{\#Z_i-1}(A_{i-1}^{\#Z_{i-1}}(A_i(T_{i-1}(\cdots(\#X))))$, which is not greater than the original by applying the first lemma.

SCRNs for Primitive Recursive Functions

Now, we show that given any primitive recursive function f, a Stochastic Chemical Reaction Network can be designed so that the state with the maximal amount of S_2 will have exactly $f(n)$ molecules of S_1, where n is given as input by being the number of molecules of an input species S_3 when the system is started. We sketch the proof here.

Any primitive recursive function can be computed by a Register Machine[4]. in time bounded by some row of the Ackermann function (see section 5.1). The required row can be determined by a structural examination of the primitive recursive function. Our Stochastic Chemical Reaction Network is designed to first compute an upper bound B on the running time needed to compute f by computing the appropriate row of the Ackermann function as in the previous section.

The Stochastic Chemical Reaction Network then simulates a Broken Register Machine (that is, a Register Machine whose decrement instructions may fail nondeterministically even when the register is not empty) for B steps, which we know is more than enough time for the Register Machine program to finish. After each of the B steps (with the `halt` instruction changed to a `nop` (no operation) instruction so that B steps can indeed occur), the Stochastic Chemical Reaction Network passes control to a "subroutine" which doubles the amount of S_2 (actually, all it can do is allow the amount of S_2 to at most double, but that is good enough). In addition, every successful decrement of a register produces an extra molecule of S_2. Thus, S_2 winds up being a large integer whose binary digits are a record of the times at which decrement instructions successfully decremented a register. This means that any run with the largest possible amount of S_2 must have always succeeded at decrementing whenever possible. In other words, it emulated the Register Machine in the correct, non-broken way. Thus we can be sure that in this run, S_1 has been computed correctly. Since the bulk of the time is consumed by doubling S_2, the correct run is also the longest possible sequence of reactions for the Stochastic Chemical Reaction Network, and the same remains true if we append a "clean up" routine to the end of the computation, that clears away the large quantity of S_2.

Thus primitive recursive functions are in perfect correspondence with questions of the form "How many molecules of S_1 will there be if a Stochastic Chemical Reaction Network produces the maximal amount of S_2?" or "How many molecules of S_1 will there be if the Stochastic Chemical Reaction Network takes the longest possible sequence of reactions?" So although questions of possibility in Stochastic Chemical Reaction Networks are decidable, we have shown here that in some ways they have the full power of primitive recursive functions.

[4] See section 6 for a description of Register Machines and Broken Register Machines, and how SCRNs can be designed to simulate Broken Register Machines.

6 Ordered Program Models: Register Machines and Fractran

Because of the above and other decidability results, Petri nets, Stochastic Chemical Reaction Networks, and VASs are typically conceptually grouped with non-uniform models such as boolean circuits, as was mentioned in section 3. However, when provided with rate constants and evaluated in a probabilistic context, these models are in fact capable of uniform computation as well.

Bennett [18] proposed a method for simulating a TM that uses a DNA-like information carrying polymer as the equivalent of a TM tape, with an attached chemical group representing the head position and head state.[5] Reactions then occur on this polymer that mimic the operation of the TM. The SCRN corresponding to this system has a different species for each polymer sequence, length, and the "head" chemical group and location. A single molecule then represents a single TM (tape and attached head), and reactions transform this molecule from one species to another. Thus infinitely many species and infinitely many reactions are needed to represent Bennett's biomolecular TM simulation as a SCRN (although augmented combinatorial formalisms, which go beyond SCRNs, can represent Bennett's chemical TMs and other Turing-universal polymer-based chemical machines; see for example [21].)

Taking a different approach of storing and processing information, we show that SCRNs with a *finite* set of species and chemical reactions are Turing universal in probability – they can execute any computer program for any length of time, and produce the correct output with high probability. Thus to increase the complexity of the computation performed by SCRNs it is not necessary to add new reactions or species (as is the case when simulating circuits or using arbitrarily complex polymers). Our method, building on [16] as described in [14], involves showing that Register Machines (RMs) can be simulated by SCRNs for any length of time with little probability of error. Since it is known that any computer program can be compiled to a RM [42, 13], we can conclude that any computer program can be effectively compiled to a SCRN. Also since there exist specific RMs known to be Turing-universal (i.e. capable of simulating *any* computer program), we can conclude that there is a *Turing-universal* SCRN that can simulate any computer program with high probability.

[5] Recall that a TM consists of an infinite tape, and a head which can be in some finite number of internal states pointing to a specified position on the tape and capable of reading and writing from and to the tape. Reading a bit of the tape allows the head to transition to different internal states and move left or right depending on the read bit; whether and which symbol is written depends of the state of the head.

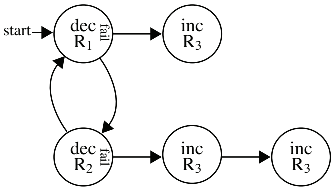


Figure 7. A register machine comparing the value of register R_1 to R_2. If $R_1 \leq R_2$ then it outputs 1 in register R_3. If $R_1 > R_2$ then it outputs 2 in register R_3. The start state is indicated with "start" and the halting states are those without outgoing arrows.

Register Machines are a simplified, idealized abstraction of how computers work, with a CPU manipulating memory. Minsky showed in the 60's that Register Machines are capable of universal computation. A Register Machine is a machine that has a fixed number of *registers*, each of which can hold an arbitrary non-negative integer. In addition to the registers, it has a fixed *program* which consists of a set of instructions. Every instruction is either an increment instruction, a decrement instruction, or a halt instruction. The increment and decrement instructions specify which register is to be incremented or decremented, and they also specify which instruction should be executed next, after the increment or decrement. Decrement instructions, however, might not succeed with their intended decrement—if the register is 0, it cannot be decremented. In this case, the decrement instruction is said to *fail*, and each decrement instruction specifies an alternate next instruction to go to in the case that the decrement fails. The current *state* of a Register Machine is given by the values of the registers, along with which instruction is the next one to execute. A simple example of an RM comparing two integers is shown in Figure 7. Register Machines are nice because of their simplicity, which makes it easy for other systems to simulate them.

One variant of Register Machines which in our experience is sometimes useful is what we call Broken Register Machines. These are the same as Register Machines except that decrement instructions are allowed to fail (nondeterministically) even if the register is non-zero. (If the register is zero, the instruction is of course forced to fail as before.) It is possible to show that Broken Register Machines turn out to be equivalent to Petri nets and VASs (and thus to Stochastic Chemical Reaction Networks as well), although the equivalence is not quite as direct as for the other systems. The nature of the equivalence between Broken Register Machines and Stochastic Chemical Reaction Networks, combined with the fact that Broken Register Machines only need to decide between two options at a time, enables one to show that in fact only two priority levels are necessary for a Stochastic Chemical Reaction Network to be universal.

Another model that turns out to be related is a lesser known model called Fractran [11], shown by Conway to be Turing universal. A Fractran program consists of an ordered list of rational numbers (see figure 1(d)). Execution is deterministic: starting with a positive integer n as input, we find the first fraction on the list that produces an integer when multiplied by n, and this product becomes the new number n'. This process is iterated forever unless it halts due to no fraction resulting in an integer. Conway showed that any Register Machine program can be converted directly into a Fractran program: representing every integer in fully factored form, $n = p_1^{a_1} \cdots p_m^{a_m}$, where p_i is the i^{th} prime, the exponents $a_1 \dots a_k$ store the contents of the k registers, while other distinct primes p_h are each present iff the Register Machine is in state h. The denominator of each Fractran fraction conditions execution on being in state h and – if the operation is to decrement the register – on having a non-empty register. The numerator provides for increments and sets the new state. Since Register Machines are Turing-universal (although since they only allow increment and decrement operations, thus storing all state in unary, they entail exponential slowdowns compared to more reasonable computational models), it follows that Fractran is also universal.

Examination of Conway's construction illustrates the relation to VASs, Petri nets, and Stochastic Chemical Reaction Networks. Considering the integer n as the vector of exponents in its prime factorization, multiplication by a fraction corresponds to subtracting the exponents in the denominator and adding the exponents in the numerator, subject to the condition that no negative exponents are generated. This corresponds exactly to a Vector Addition System. Equivalently, each fraction can be interpreted as a chemical reaction: each species is represented by a unique prime number, and the denominator specifies the reactants and their stoichiometry, while the numerator specifies the products. (Catalytic reactions would correspond to non-reduced fractions, and can be avoided as shown in figure 1.) The determinism – and hence universal computational power – inherent in Fractran execution corresponds to there being a strict priority in which the various possible transitions are applied.

6.1 Computation in Stochastic Chemical Reaction Networks

If it were possible to prioritize the reactions in a Stochastic Chemical Reaction Network, then by analogy to the ordered fractions in Fractran, this would establish the Turing-universality of Stochastic Chemical Reaction Networks. (This result is also well known in the field of Petri nets, and our analysis of Register Machines shows that in fact only two distinct priority levels are necessary.)

By giving higher-priority reactions vastly faster rate constants k_α, we can approximate a priority list: almost surely, of all reactions for which all reactants are present in sufficient number, a reaction with a much faster rate will occur first. However, "almost surely" turns out not to be good enough for a couple of reasons. First, there is a non-zero probability of the slow reaction happening at each step, and thus probability of successful output falls exponentially with the number of steps. Second, the number of molecules of a given species can potentially exceed any bound, so the ordering of actual rates $\rho_\alpha(\mathcal{A})$ may eventually be different from the specified ordering of rate constants k_α. Especially in light of the decidability results mentioned above, it is not surprising that this naive approach to achieving Turing universality with Stochastic Chemical Reaction Networks fails.

If there were some way to increase rate constants over time, this could solve these problems, but of course, rate constants cannot change. Another way to promote one reaction over another would be to give the preferred reaction some extra time to occur before the competing reaction has a chance to occur. This approach turns out to be workable, and it is not too hard to set up some reactions that produce a signal after some delay, where the delay depends on a particular concentration. We refer to such a set of reactions as a *clock*. An important technical point is that since the entire computation will consist of an unknown number of steps, the probability of error at any given step must be decreasing so that the sum of all the probabilities can remain small regardless of how long the computation winds up taking. To address this issue, the clock can at each step increase the concentration that controls its delay, so that the delays are progressively longer, and thus the probabilities of error are progressively smaller. Fortunately, it turns out that a simple polynomial slowdown in overall computation time is all that is required for making the total probability of error (over the entire course of the arbitrarily long computation) be small.

In the following, we give a construction for simulating Register Machines with Stochastic Chemical Reaction Networks with only a polynomial slowdown, and we prove that successful output will occur with fixed probability $1 - \epsilon$ independent of the input and computation time. An initial number of "precision molecules" can be added to reach any desired level of ϵ. Thus, tolerating a fixed, but arbitrarily low, probability that computation will result in an error, Stochastic Chemical Reaction Networks become Turing universal. In consequence, the probabilistic variants of the reachability and producibility questions are undecidable.

The simulation is relatively simple to understand, but its performance is limited by the fact that it is simulating a Register Machine, which is exponentially slower than a Turing Machine (in the space used by the Turing Machine), due to its unary representation of information. Can Stochastic Chemical Reaction Networks do better than this? It turns out that they can. In section 7, we discuss a more sophisticated algorithm that allows Stochastic Chemical Reaction Networks to directly polynomially simulate a Turing Machine.

Probability in SCRNs is Undecidable

Theorem 3. *For all $0 \leq \epsilon < 1/2$, the following problem is undecidable: given a Stochastic Chemical Reaction Network C, a species S, and a starting state $\mathcal{A}$, determine, to within ϵ, the probability that C starting from $\mathcal{A}$ will produce at least one molecule of S.*

To prove this theorem, we will show how Stochastic Chemical Reaction Networks are capable of simulating Register Machines. First, we define the correspondence between instantaneous descriptions of Register Machines and states of Stochastic Chemical Reaction Networks that our construction attains. Then, we show that determining whether a Register Machine ever reaches a particular instantaneous description is equivalent to ascertaining whether our Stochastic Chemical Reaction Network enters a set of states with sufficiently high probability.

Definition 1 *An* instantaneous description *ID of a Register Machine M with t registers is a vector $(a, c_1, \ldots, c_t)$ where a is a state of M and $c_i \in \mathbb{N}$ represents the value of register i.*

Definition 2 *The reachability relation $ID \overset{M*}{\rightarrow} ID'$ is defined naturally. Namely, it is satisfied if M eventually reaches ID' starting from ID.*

Definition 3 *For two states $\mathcal{A}$ and $\mathcal{B}$ of a Stochastic Chemical Reaction Network C we write $\mathcal{A} \overset{C}{\rightarrow} \mathcal{B}$ if there is a reaction that takes the system from $\mathcal{A}$ to $\mathcal{B}$. Let $\overset{C*}{\rightarrow}$ be the reflexive transitive closure of $\overset{C}{\rightarrow}$.*

Instantaneous descriptions of a Register Machine map to sets of states of our Stochastic Chemical Reaction Network as follows:

Definition 4 *For an instantaneous description $ID = (a, c_1, \ldots, c_t)$ of a Register Machine M let $\mathcal{M}(ID, n)$ be the state of a Stochastic Chemical Reaction Network that contains exactly:*

- *n molecules of species A,*
- *c_i molecules of R_i $\forall 1 \leq i \leq t$,*
- *1 molecule of S_a,*
- *and 1 molecule of T, B, B' and B'' each.*

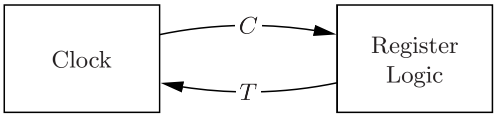
Definition 5 *Our Stochastic Chemical Reaction Networks will be said to ϵ-*follow *a Register Machine M if there is some n_0 such that for all instantaneous descriptions ID and ID' of M we have:*

$$(a)\; ID \overset{M*}{\rightarrow} ID' \iff \Pr[\mathcal{M}(ID, n_0) \overset{C*}{\rightarrow} \mathcal{M}(ID', n) \text{ for some } n] > 1 - \epsilon$$
$$(b)\; ID \overset{M*}{\not\rightarrow} ID' \iff \Pr[\mathcal{M}(ID, n_0) \overset{C*}{\rightarrow} \mathcal{M}(ID', n) \text{ for some } n] < \epsilon$$

Theorem 4. *For any Register Machine M, and any $\epsilon > 0$, there is a Stochastic Chemical Reaction Network C that ϵ-follows M.*

In fact, slight modifications of our construction can show that all questions about whether a Stochastic Chemical Reaction Network "might do X" mentioned section 5 become uncomputable in the probabilistic setting ("does X with probability $> \epsilon$").

(a) The Clock and Register Logic Modules



(b) Clock

$$
\begin{array}{l}
T + B \rightarrow T' + B \\
T' + A \rightarrow T + A \\
T' + B' \rightarrow T'' + B' \\
T'' + A \rightarrow T' + A \\
T'' + B'' \rightarrow C + B'' + A
\end{array}
$$

(c) Register Logic

In state a increment register i and go to state b $\Rightarrow$ $S_a + C \;\rightarrow\; S_b + R_i + T$

In state a decrement register i and go to state b, or if the register is empty go to state c $\Rightarrow$

$$
\begin{array}{lcl}
S_a + R_i & \rightarrow & S'_a \\
S'_a + C & \rightarrow & S_b + T \\
S_a + C & \rightarrow & S_c + T
\end{array}
$$

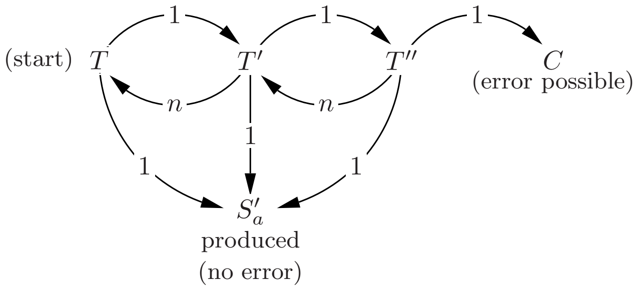
Figure 8. Simulating a register machine. (a) The communication between the clock and the register logic modules is through single molecules of species C and T. (b) The clock module is responsible for producing a C molecule once every so often. The clock module is designed so that the length of time between receiving a T and producing a C slowly increases throughout the computation, thus slowing down the register logic module to help it avoid error. Specifically, the more A's there are, the longer the delay. The clock starts out with n_0 A's and one each of B, B', and B'' and T. Every clock cycle not only produces a C, but increases the number of A's by one. Thus at the beginning of the k^{th} cycle, there are $n = k + n_0$ molecules of A. The clock's operation is further analyzed in figure 9. (c) The register logic module simulates the register machine state transitions. The register logic module starts out with quantities of molecules of R_i indicating the starting value of register i, and a single molecule of species S_a where a is the start state of the register machine. Note that at all times the entire system contains at most a single molecule of any species other than the A and R_i species. All rate constants are 1. (The construction will work with any rate constants.)

Proof (Proof of theorem 4). We construct a Stochastic Chemical Reaction Network to simulate the Register Machine, consisting of two components: a clock module and a register logic module (figure 8). The communication between the modules is established through two species, T and C, of which at most a single molecule is present. Whenever the clock releases the C, the register logic module can complete a step of the register machine (with the exception of the actual decrement of a decrement instruction), converting the C into a T in the process. The clock module then takes the T and, after a delay, releases another C to repeat the process. The delay imposed by the clock module makes it exceedingly likely that any decrement waiting to happen will occur before the next C is released. This effectively enforces the reaction order that is necessary for correct computation.

The register logic module has a single molecule of species S_a for every state a of the register machine. The number of molecules of species R_i stores the value of the register i. If the current register machine state a is an increment state, once the clock module releases the C then the reaction $S_a + C \rightarrow S_b + R_i + T$ increments the ith register and transitions to the next state b. If the current state is a decrement state and the register i being read is empty, then the reaction $S_a + R_i \rightarrow S'_a$ is not possible, and once the clock module releases the C, the reaction $S_a + C \rightarrow S_c + T$ takes place and transitions to the state c indicating that the register is empty. If the register i is not empty (i.e. there is at least one molecule of R_i in solution), then the intent is that the reaction $S_a + R_i \rightarrow S'_a$ should decrement the register and capture S_a before the clock module next releases a C. (Otherwise, the reaction $S_a + C \rightarrow S_c + T$ could occur first, erroneously sending the register logic module into the state c, which is only supposed to happen if the register is empty.)

Thus, the only possible error that can occur in the register logic module is if $S_a + C \rightarrow S_c + T$ occurs before $S_a + R_i \rightarrow S'_a$ in a decrement step, when register i is not empty. By delaying the release of the C, the clock module ensures that the probability of this happening is low. The delay increases from step to step sufficiently to guarantee that the union bound taken over all steps of the probability of error does not exceed ϵ.

Let us analyze the probability of error quantitatively. Suppose the current step is a decrement step and that the decremented register has value 1. This is the worst case scenario since if the register holds value greater than 1, the rate of the reaction $S_a + R_i \rightarrow S'_a$ is correspondingly faster, and if the step is an increment step or the register is zero, then no error can occur. Figure 9 illustrates the state diagram of the relevant process. All of the reactions in our Stochastic Chemical Reaction Network have the same rate constant of 1. Thus, all reactions with exactly one molecule of each reactant species in solution have the same reaction rate of 1. There are two reactions for which this single molecule condition is not true: $T' + A \rightarrow T + A$ and $T'' + A \rightarrow T' + A$, since there are many A's in solution. If there are n A's in solution, each of these two reactions has rate n. Now, we'll bound the probability that the clock produces the C before the $S_a + R_i \rightarrow S'_a$ reaction occurs, which is a bound on the probability of error. The top 4 states in the diagram (figure 9) represent the 4 possible states of the clock: we either have a T, T', T'', or a C. A new cycle starts when the register logic module produces the T and this is the start state of the diagram. No matter what state the clock is in, the reaction $S_a + R_i \rightarrow S'_a$ can occur at rate 1 in the register logic module. Once this happens, no error is possible. On the diagram this is indicated by the bottom state (no error) which is a sink. On the other hand, if a C is produced first then an error is possible. This is indicated by the sink state C (error possible).



$$\frac{d}{dt}\begin{bmatrix} s \\ s' \\ s'' \\ p \\ q \end{bmatrix} = \begin{bmatrix} -2 & n & 0 & 0 & 0 \\ 1 & -2-n & n & 0 & 0 \\ 0 & 1 & -2-n & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} s \\ s' \\ s'' \\ p \\ q \end{bmatrix}$$

Figure 9. The state diagram for a single decrement operation when there are n A's and the register to be decremented holds the value 1, and the corresponding system of differential equations governing the instantaneous probabilities of being in a given state. The numbers on the arrows are the transition rates. The instantaneous probability of being in state T is s, in state T' is s', and in state T'' is s''. The instantaneous probability of being in the error-possible state is p and the probability of being in the no-error state is q.

We compute the absorption probability of the error-possible state by solving the corresponding flow problem. Solving the system of differential equations in figure 9 for $\frac{dp}{dt}$ under the condition that $\frac{ds}{dt} = -1$, $\frac{ds'}{dt} = \frac{ds''}{dt} = 0$, we find that the absorption probability of the error-possible state is $p = \frac{1}{(n+2)^2+4}$. Thus the probability of error for a step with n A's is bounded by $p = \frac{1}{(n+2)^2+4}$. In order to be sure that the probability that no error occurs during *any* point in the computation is larger than $1 - \epsilon$, recall that n increases by one at each step, so we need

$$\sum_{n=n_0}^{\infty} \frac{1}{(n+2)^2+4} < \epsilon.$$

The terms in the above inequality are inversely quadratic in n, so if $n_0 = 1$ then the sum is finite (in fact, it is roughly 0.3354). This means that for any ϵ, we can choose an appropriate n_0, the initial number of A's, to make the above inequality true.

How fast is the Register Machine simulation? Since each consecutive step is potentially delayed longer and longer, we need to be careful that the simulation is not slowed down too much. Indeed, it can be shown that the expected time to perform t steps of the Register Machine simulation requires at most $O(t^4)$ SCRN time [14].

6.2 Eliminating Dependency on the Number of Molecules Disables Universal Computation

If the rates of the possible reactions do not depend on the number of molecules then it can be shown that the system is incapable of universal computation. In particular, it will be predictable in the sense that the probability that at least one molecule of a given species is eventually produced can be computed to arbitrary precision. This precludes the previous output method of using an indicator species whose production with high or low probability indicates the outcome of the computation. Further, any other method of output that can be converted to this form of output cannot be universal either. This includes, for example, Stochastic Chemical Reaction Networks that enter a specific state with high or low probability to indicate the output. Specifically, the model we are considering here is the following: Suppose we are given a Stochastic Chemical Reaction Network with given constant rates for all the reactions, and an initial set of molecules. Then at each step, based solely on the reaction rates, a reaction is chosen. This reaction then occurs if the reactants for it are present. Such steps continue indefinitely.

The difference between this model and the standard stochastic one is that in the standard model, the reaction rate is obtained by combining a rate constant with the current concentrations as described in section 2.1 (eqn. 1), while here for all reactions α and states $\mathcal{A}$, $\rho_\alpha(\mathcal{A}) = k_\alpha$ if all the reactants of α are present in $\mathcal{A}$ and 0 otherwise.

Theorem 5. *Let $\mathbb{S}$ be the infinite set of all states with at least one molecule of the indicator species present. Suppose for all reactions α and states $\mathcal{A}$, $\rho_\alpha(\mathcal{A}) = k_\alpha$ if all the reactants of α are present in $\mathcal{A}$ and 0 otherwise. Then there is an algorithm that given $0 < \epsilon$ and any starting state $\mathcal{A}$, computes $\Pr[\mathcal{A} \overset{C*}{\to} \mathcal{B}$ for some $\mathcal{B} \in \mathbb{S}]$ within ϵ.*

Let $\tilde{\mathbb{S}}$ be the (probably infinite) set of states from which no state in $\mathbb{S}$ is reachable, and let $\mathbb{R}$ be the set of states outside $\mathbb{S}$ from which it is possible to reach $\mathbb{S}$. (Note that given any state, the question of whether it is possible to reach some state in $\mathbb{S}$ is computable, as shown in section 5.2.) Note also that there is a bound b such that for any state $\mathcal{A} \in \mathbb{R}$, the length of the shortest sequence of reactions leading from $\mathcal{A}$ into $\mathbb{S}$ is at most b. This means that there is some constant p_0 such that for any state $r \in \mathbb{R}$, the probability of entering $\mathbb{S}$ within b steps is at least p_0. Thus, the probability of remaining in $\mathbb{R}$ must decay at least exponentially.

This implies that the probability that the system will eventually enter $\mathbb{S}$ or $\tilde{\mathbb{S}}$ is 1, and so simply by computing the probabilities of the state tree for $\mathbb{R}$ far enough, one can compute the probability of entering $\mathbb{S}$ to arbitrary precision.

7 Efficiency of Computation by Stochastic Chemical Reaction Networks

Section 6 showed that universal computation (in probability) can be performed by SCRNs, but our construction inherits the ridiculous inefficiency of Register Machines, which in general require exponential time to simulate Turing machine computations. Is it possible to use the power of chemistry to perform computations more quickly and efficiently?

Trivial ways to speed up a chemical computer involve changing environmental conditions such as increasing the temperature or the effective concentration (molecular count per unit volume). In order to discuss the "intrinsic speed" of the computer we are proposing, we fix the temperature, as well as the maximum concentration (recall that the volume scales dynamically with the molecular count in our model, see section 2.1). Then the performance of the chemical computer will be gauged asymptotically as the size of the "tape" as well as the number of simulation steps increases. With improved chemical programming, it turns out that compared to the abstract Turing Machine, its chemical implementation incurs only a polynomial slowdown. The volume required, however, inevitably grows exponentially with the size of the tape of the Turing machine being simulated. This is impossible to avoid since fixing the number of species there is simply no way to store information in a form other than unary.

7.1 Stochastic Chemical Reaction Networks Can Efficiently Simulate Turing Machines

Theorem 6. *For any* $0 < \epsilon < 1/2$ *and any Turing Machine* M *we can make a Stochastic Chemical Reaction Network that, starting with* n *molecules of species* S_{in} *(and some number of molecules of other species, dependent on* ϵ *but not* n*), will with high probability (*$> 1 - \epsilon$*) result in fast (expected time* *polynomial in the running time of $M(n)$) and accurate (eventually produces S_{halt} iff $M(n)$ halts) computation.*

Of course, by having different output species, the same output method can be used to indicate a 0/1 output or in fact an integer output encoded in unary.

The overall idea to achieve this fast Turing Machine simulation is to adopt the Register Machine simulation, but allow more sophisticated operations on the registers [17, 14]. If in addition to incrementing and decrementing a register, we could double or halve the value of any register and check the remainder in a constant number of clock cycles of the Register Machine simulation, then we could simulate a Turing Machine in linear time. To do this, we can represent the accessed portion of the Turing Machine head tape as two integers whose binary representation encodes the tape to the left and to the right of the head respectively, such that their least significant bits represent the contents of the tape cells immediately to the left and right of the head. Since reading and removing the least significant bit corresponds to halving and checking for remainder, and writing a new least significant bit corresponds to doubling and possibly adding one, a single Turing Machine step can be performed in a small constant number of these enhanced Register Machine cycles. With registers represented in unary molecular counts, halving would correspond to a reaction scheme that takes two molecules of one species and produces one molecule of another species, while doubling would correspond to a reaction scheme that takes one molecule of one species and produces two molecules of another species. Conversion of all molecules must take place quickly and exactly—if a single molecule is not converted, the least significant bit(s) will be in error. Unfortunately, we will see that halving a register quickly is rather difficult, but fortunately we will be able to avoid the halving operation.

In the following section, we provide a construction similar to (but not identical to) that in of ref. [14] and give an informal explanation of how it works.

The Exploding Computer

To perform computation quickly using molecular counts, we have a number of challenges. The primary difficulty is that if every molecule matters for a decision, then the presence or absence of a single molecule (for example, the parity of a register) must be communicated to all other molecules in the system that are affected by the decision. But in our model of well-mixed chemistry, communication happens only by chance collisions between molecules—and rare species will therefore interact rarely.

The main technique that allows large numbers of molecules to be processed quickly is for the state changes to occur via explosive chain reactions. These "explosions" do not necessarily increase the number of molecules; they might simply change the molecules from one form to another. Each explosion starts as an exponentially growing chain reaction, until the amount of reactive material starts to get used up, at which point it finishes with exponential decay of the reactive material. Thus an exponential amount of reactive material can be processed in a given amount of time, as shown in figure 10. By changing the number of product molecules in the reaction, the explosion scheme can be easily transformed into a means to quickly and exactly double the number molecules present.

The naive implementation of having a halving reaction akin to $2M \rightarrow M'$ is slow for the same reasons as shown in figure 11.

If we are to avoid having to halve the value of a register, we must have an architecture for computation that only requires doubling when reading and writing bits to and from memory. To do this, we use the digits of the memory integer as a queue of binary digits. We can read and remove the most significant digits (as we will show), we can shift the digits over (by doubling, or multiplication by a constant), and we can write new low order digits by simply producing a few extra molecules. Thus freshly written digits get exponentially amplified step by step until they are the largest contribution to the overall magnitude, at which point the system is able to detect their value.

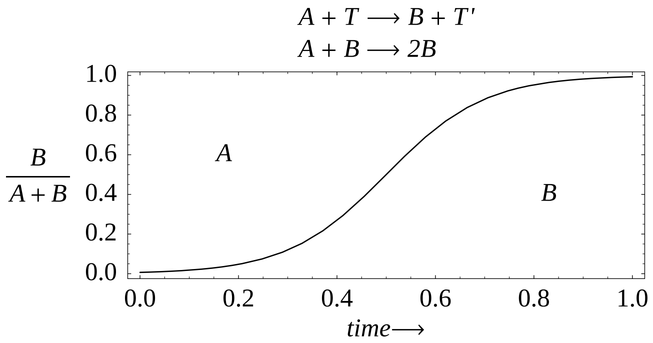


Figure 10. The time course of a reactant-limited chain reaction explosion, shown as a conversion from a species A to a species B, initiated by a trigger T. If at the beginning, a fraction p of all molecules in the system are A molecules, then the number of converted molecules grows like e^{kpt}, where t is time and k is the rate constant of the reaction catalyzed by B. For the first half of the chain reaction, at least $p/2$ of the molecules are A, and so the expected time for the first half to complete is under $(2/kp) \log |A|/2$. For the second half of the chain reaction, over $p/2$ of the molecules are B, so each molecule of A gets transformed at a rate above $kp/2$, so the quantity of A decreases faster than $e^{kpt/2}$, and the expected time for the second half to complete is under $(2/kp) \log |A|/2$. Thus the total time needed for the explosion to finish is on the order of $\log |A|/p$.

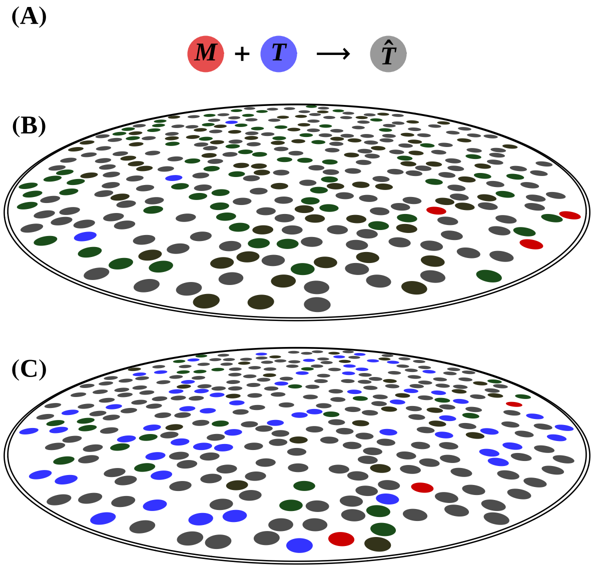


Figure 11. **(A)** To read the most significant digit of M, we compare M (red) to a known threshold quantity T (blue). This is done by a simple reaction, $M + T \to \hat{T}$. The goal is that after all the M and T molecules have reacted to form $\hat{T}$, only the species in excess, either M or T, will remain. However, the time it takes for this reaction to complete will depend on the amounts of M and T. **(B)** If M and T are present in nearly equal quantities, then towards the end of the reaction only a few molecules of M and a few molecules of T will remain, and we will need to wait a very long time to be confident that these last few molecules have found and reacted with each other, especially when the volume is large. **(C)** If either M or T is significantly in excess of the other, with the excess being at least some constant fraction of the entire volume, then towards the end of the reaction, as one of the species approaches extinction, its rate of decay will remain exponential, and the reaction will fully finish quickly, regardless of volume.

Before proceeding, we should make sure that these operations are sufficient for efficient simulation of Turing Machines. To see this, here's how to convert a Turing Machine into a program that uses only queues. First consider a Turing Machine that uses a fixed amount of space on a binary tape. This finite tape is encoded in the queue using three bits per cell, one bit for the cell's value, an another bit to indicate the cell that the Turing Machine head is reading, and the third bit to indicate the first and last cells. Note that after one time step, the tape will be changed only in the three cells around where the head is reading: the center cell might have a new value, and either of the adjacent cells might need to be marked to indicate that this is where the Turing machine is now reading. To implement a single time step of the Turing Machine, the new queue program will make a pass through the whole queue, keeping the most recent three cells memorized at every step. Each step consists of spitting out the correct new value for the oldest of the three cells and then reading in one more cell. The queue program knows when it has completed a pass, thanks to the third bit in each cell. Thus, to simulate Turing Machines that use arbitrary amounts of tape, the queue program can simply output some blank cells at the beginning and end of each pass. Overall, the queue program is slower than the original Turing Machine, but only by a linear factor—if the original machine took $O(t^{16})$ steps, the queue program will take $O(t^{17})$ steps. (Slightly more efficient implementations are possible [43, 14].)

With this queue architecture, the challenge of detecting a single molecule is avoided; all we need is a scheme that allows the system to be able to read the high order memory digit quickly and accurately. This can be achieved by storing integers in the memory using a Cantor-set structure. To be able to read the most significant digit of the memory integer, we need to compare the memory integer to a threshold value, and as shown in figure 11 it is important that the memory integer not be too close to the threshold value. That is, the magnitude of the memory integer, regardless of the contents of the memory, should be separated from the threshold value it is being compared to by a gap that is at least some fixed fraction of the threshold value itself, so that the comparison will always complete in logarithmic time. The way we will satisfy this requirement is by using numbers which, in base 3, use only the digits 1 and 2. These numbers have a Cantor-set structure. Thus the highest possible $k+1$ digit number starting with 1, namely $2 \cdot 3^k - 1 = 1222...2_3$, and the lowest possible $k+1$ digit number starting with 2, namely $2.5 \cdot 3^k - 0.5 = 2111...1_3$, are separated by an interval that is proportional in size to the numbers themselves, making the leading digit easily detectable.

The system can write a low order digit into the memory by simply having just a single molecule present of the species responsible for writing this digit.

We have discussed the representation of the queue (i.e. encoded Turing Machine tape) as the molecular counts of a register species—but how do we represent which step of the program is currently being executed? Since the program contains a finite number of states, it is possible to assign a distinct molecular species for each program state. In fact, we combine both representations into one: if the queue program is in state 10 with the integer M in the queue, then we will have M copies of the molecular species $\mathbf{M}_{10}$. The molecular count encodes the queue, and the molecular species encodes the program step being executed. Thus, to push a '1' onto the bottom of the queue and transition to state 20, we perform an "explosion" that converts the M copies of $\mathbf{M}_{10}$ into $2M$ copies of $\mathbf{M}_{20}$ and then produce one more $\mathbf{M}_{20}$. Effectively, the chemical system is designed as a sequence of conditional explosive reactions, each of which changes the "color" of the solution.

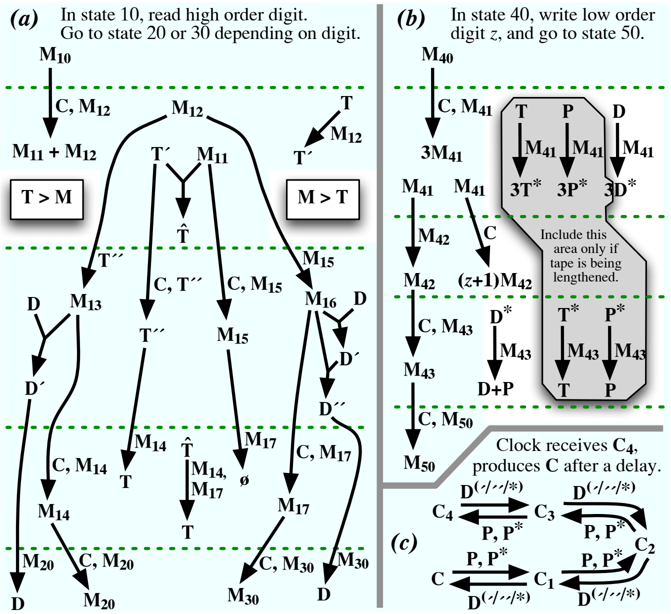


Figure 12. Reactions for a chemical system emulating a Turing Machine that has been converted into a queue program. The horizontal dashed lines represent clock cycles. This example uses the convention that the program states of the queue program are multiples of 10, while substates required for the chemical implementation modify the rightmost digit. Species listed to the side of an arrow are catalysts for the reaction: At least one of them must be present for the reaction to occur. In a slight abuse of this notation, when the clock signal **C** is used as a "catalyst", it is actually being converted to $\mathbf{C}_4$. (So, it is not really being used catalytically—but this notation makes the diagram cleaner.) $\mathbf{M}_i$ molecules store the memory integer and encode the program state i. **T** molecules (of various sorts) store the comparison threshold. **D** molecules store a single '1' in the most significant digit. There is more **D** than **T**. **P** molecules store the current precision of the system. **C** is the clock signal. There is exactly one **C** molecule at any time. ***(a)*** Reading a bit of memory. ***(b)*** Writing a bit of memory. ***(c)*** Operation of the clock. Any **D** species ($\mathbf{D}$, $\mathbf{D}'$, $\mathbf{D}''$, or $\mathbf{D}^*$) can serve as a catalyst for the conversion of the **C** in each step.

As in the Register Machine simulation of the previous section, the system can have a very low chance of ever making a mistake, even though there is some chance of error at every step, as a result of having the speed of the system regulated by a clock that slows down over time. Since each step is more likely to succeed when it is given more time to complete its work, the slowing clock makes successive steps more and more likely to succeed. Intuitively, if it makes it through the first several steps without error, later steps become progressively easier to do correctly, so it's quite likely to just go forever without making any mistakes. Mathematically, the probability of error decreases at each step fast enough that the sum of all the probabilities of error converges to a small value. This value can be made arbitrarily small, regardless of the computation to be done by the Turing machine, simply by starting the system with a greater number of precision molecules (**P** in figure 12).

The molecular species and reactions for the simulation are shown in figure 12. Four clock cycles are required for each step so that the various sub-operations do not interfere with each other. At each step the clock molecule **C** triggers an explosive chain reaction, and the output of that chain reaction is used to catalyze all the other reactions occurring at that step (with the exception of comparisons and subtractions, which have no catalysts).

When reading a bit of memory, the reactions compare the memory M with the threshold T, as discussed in figure 11. After the first clock cycle, which performs the comparison, only half of the remaining reactions will occur, according to the result of the comparison. If $T > M$, then the leading digit of M was '1', and only the reactions on the left side will occur. If $M > T$, then the leading digit of M was '2', and only the reactions on the right side will occur. During the second clock cycle, D is subtracted from M either once or twice, so as to zero out the leading digit (which was just read). During the third cycle the threshold T is restored, and the fourth cycle cleans up D and M.

Every read operation must be followed by a write operation, so that the tape does not shrink. Extra write operations are allowed, so the tape can grow, but states corresponding to such extra operations must include the reactions in the gray region. The first clock cycle multiplies M by 3, and if the tape is growing, then T, P, and D also get multiplied by 3. The second clock cycle writes the new digit of M. The third cycle cleans up D, T, and P, and it also adds D to P. This way, the precision P is always a multiple of D, and the multiple grows by one with each write operation, so the precision of the simulation increases at every step. The fourth cycle cleans up M.

The clock molecule is used to trigger the advance from one stage of the reaction to the next. Therefore, when **C** is used as a "catalyst", it is actually transformed into a $\mathbf{C}_4$, so that it cannot trigger the advance to the following stage until some time has passed. Effectively, the clock slowly lets the $\mathbf{C}_4$ become a **C** again. To become a **C**, it has to work its way down, but since P is greater than D by a growing factor, the process of the $\mathbf{C}_4$ becoming a **C** becomes slower and slower as time goes on. This slowing of the clock is what makes the whole system become more and more reliable over time.

A detailed construction based on the same principles is presented in [14], with an analysis of its error probability and running time. A more efficient construction can be implemented based on the work of Angluin et al in the distributed computing literature [17] (see [14, 44]). Simulating t_{TM} steps of a Turing Machine using s_{TM} space can be done in $\text{polylog}(m) \cdot t_{TM}$ time where $m = O(2^{s_{TM}})$ is the total maximum molecular count encountered.

7.2 Turing Machines Can Efficiently Simulate Robust Stochastic Chemical Reaction Networks

We have seen that enforcing reaction order, even probabilistically, is enough to achieve Turing-universality. However, our simulation of Turing Machines (and Register Machines) by Stochastic Chemical Reaction Networks, uses reaction propensities rather weakly: while it was essential that one reaction propensity is higher than another, and increases over time, the exact value of reaction propensities are not used in the computation. Intuitively we can say that a Stochastic Chemical Reaction Network behaves "robustly" if its behavior does not depend crucially upon getting the reaction propensities exactly right. Formal definitions can be found in [44], as well as the proof that the Turing Machine embedding based on [17] is robust.

Such robust chemical systems form an interesting class, whose computational power can be almost exactly captured, bounding above and below the maximum amount of computation such systems can perform in a unit of time, compared to a Turing Machine. Although on the order of m reactions can occur per unit time, where m is the total number of molecules present, the actual amount of computation is at most polylog(m) Turing Machine steps.

While fast Turing Machine embeddings in robust Stochastic Chemical Reaction Networks show a lower bound on their computational power, how can we show that they are not capable of performing more computation per unit time? The main idea of the argument is that robust chemical systems are easy to simulate by a Turing Machine. Intuitively, since robust chemical systems are robust to perturbations in reaction rates, they permit some sloppiness when trying to predict their behavior. Then, since it is widely believed that there is no universal way of speeding up Turing Machines, it should not be possible to speed up arbitrary Turing Machines by embedding them in an chemical system and simulating the system. With some caveats related to real-number arithmetic, for robust systems, the problem of estimating the probability of being in a given state at a given time t can be solved in polylog$(m) \cdot t$ computation time on a Turing Machine, where m is the maximum molecular count encountered. This implies that, loosely stated, for robust Stochastic Chemical Reaction Networks, it is neither possible to embed more than polylog(m) computation time per chemical unit time, nor is it possible to simulate the Stochastic Chemical Reaction Network using less than polylog(m) computation time per chemical unit time [44].

It should be emphasized that the correspondence between Turing Machines and Stochastic Chemical Reaction Networks is surprisingly tight. One can simulate the other with surprisingly little loss of efficiency (especially for programs using little memory where polylog(m) for $m = O(2^{s_{TM}})$ is small compared to t_{TM}).

8 Concluding Remarks

The power of different systems to do computation can vary greatly. It has previously been assumed that systems such as genetic regulatory networks and chemical reaction networks are much weaker than the gold standard computational systems such as Turing Machines. On the other hand, we have become accustomed to proofs that even some of the simplest systems are capable of universal computation [45, 46], meaning that they are in some senses equivalent in power to Turing Machines, and thus predicting their eventual behavior is impossible even in theory. Chemical reaction networks have been shown to be universal when combined with polymer memory [18] or membrane-separated compartmental memory [22], but researchers have previously assumed that on their own, a finite number of species in a well-mixed medium can only perform bounded computations [24, 22].

In contrast with this historical intuition, here we have shown that in fact such "plain" chemical reaction networks can indeed perform unbounded computation, using the concentration (number of molecules) of each species as the storage medium. We went on to pinpoint both the power and the weakness of chemical reaction network computation by showing that it is just as fast as Turing Machine computation, but that it requires exponentially more space.

This universality of chemical reaction networks turns out to derive from their probabilistic nature. If the possible reactions in a chemical system could be prioritized, so that the next reaction at each step is always the one with highest priority, then universal behavior is easily attainable (along the lines of [12]), but of course chemistry does not behave in this way. However, since the reaction rates in a chemical system are influenced by the concentrations, they are somewhat under the control of the system itself, and as we have shown, this weak form of prioritization turns out to be enough to let the system perform universal computation with high probability of success.

If we require that the chemical system be guaranteed to give the right answer without fail, then the system is effectively deprived of the opportunity to use its reaction rates, since they only influence what is likely to happen, not what is guaranteed to happen. Indeed, in this situation, the system is incapable of universal computation. Thus, the stochastic reaction rate foundation turns out to be the source of the computational power of chemical reaction networks.

Open questions, along the lines of the results we have given, include:
(1) Are continuous Stochastic Chemical Reaction Networks (using mass action kinetics) Turing universal?[6]
(2) Can one have a universal Stochastic Chemical Reaction Network which has constant probabilities (that don't depend on concentrations) for all reactions except one, with the remaining reaction having a decaying probability that depends on time (but not on concentrations)?

[6] Eric Stansifer (personal communication) seems to have answered this question in the affirmative.

(3) Can Stochastic Chemical Reaction Networks that have reversible reactions be universal?
(4) What is the power if one wishes to guarantee that all paths in a Stochastic Chemical Reaction Network lead to same result (confluent computation)? Are we limited to boolean logic circuits, or can we do some sort of uniform computation?
(5) Can a more efficient Turing Machine simulation be embedded in a non-robust Stochastic Chemical Reaction Network than a robust one?

Acknowledgments

The research was supported in part by the "Alpha Project" at the Center for Genomic Experimentation and Computation, an NIH Center of Excellence (grant number P50 HG02370), as well as NSF Grants No. 0523761 and 0728703 to EW and NIMH Training Grant MH19138-15.

References

1. Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81:2340–2361, 1977.
2. Adam P. Arkin, John Ross, and Harley H. McAdams. Stochastic kinetic analysis of a developmental pathway bifurcation in phage-l Escherichia coli. *Genetics*, 149:1633–1648, 1998.
3. Michael Gibson and Jehoshua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *Journal of Physical Chemistry A*, 104:1876–1889, 2000.
4. Purnananda Guptasarma. Does replication-induced transcription regulate synthesis of the myriad low copy number proteins of Escherichia coli? *Bioessays*, 17:987–997, 1995.
5. Benjamin Levin. *Genes VII.* Oxford University Press, 1999.
6. Harley H. McAdams and Adam P. Arkin. Stochastic mechanisms in gene expression. *Proceedings of the National Academy of Sciences*, 94:814–819, 1997.
7. Michael B. Elowitz, Arnold J. Levine, Eric D. Siggia, and Peter S. Swain. Stochastic gene expression in a single cell. *Science*, 297:1183–1185, 2002.
8. Gurol M. Suel, Jordi Garcia-Ojalvo, Louisa M. Liberman, and Michael B. Elowitz. An excitable gene regulatory circuit induces transient cellular differentiation. *Nature*, 440:545–550, 2006.
9. Javier Esparza and Mogens Nielsen. Decidability issues for Petri Nets – a survey. *Journal of Information Processes and Cybernetics*, 3:143–160, 1994.
10. Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(4):147–195, 1969.
11. John H. Conway. Unpredictable iterations. In *Proceedings of the 1972 Number Theory Conference*, pages 49–52. University of Colorado, Boulder, 1972.
12. John H. Conway. *Fractran: A Simple Universal Programming Language for Arithmetic*, chapter 2, pages 4–26. Springer-Verlag, New York, 1987.

13. Marvin Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, New Jersey, 1967.
14. David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 2008. Published online (DOI 10.1007/s11047-008-9067-y).
15. Gianluigi Zavattaro and Luca Cardelli. Termination problems in chemical kinetics. In Franck van Breugel and Marsha Chechik, editors, *CONCUR*, volume 5201 of *Lecture Notes in Computer Science*, pages 477–491. Springer, 2008.
16. Anthony M. L. Liekens and Chrisantha T. Fernando. Turing complete catalytic particle computers. In *Proceedings of Unconventional Computing Conference, York*, 2006.
17. Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. Technical Report YALEU/DCS/TR-1358, Yale University Department of Computer Science, 2006. Extended abstract to appear, DISC 2006.
18. Charles H. Bennett. The thermodynamics of computation – a review. *International Journal of Theoretical Physics*, 21(12):905–939, 1982.
19. Gheorghe Paun. On the power of the splicing operation. *Intern. J. Computer Math.*, 59:27–35, 1995.
20. Stuart A. Kurtz, Stephen R. Mahaney, James S. Royer, and Janos Simon. Biological computing. In L. A. Hemaspaandra and A. L. Selman, editors, *Complexity Theory Retrospective II*, pages 179–195. Springer, 1997.
21. Luca Cardelli and Gianluigi Zavattaro. On the computational power of biochemistry. In Katsuhisa Horimoto, Georg Regensburger, Markus Rosenkranz, and Hiroshi Yoshida, editors, *AB*, volume 5147 of *Lecture Notes in Computer Science*, pages 65–80. Springer, 2008.
22. Gerard Berry and Gerard Boudol. The chemical abstract machine. In *Proceedings of the 17th ACM SIGPLAN-SIGACT annual symposium on principles of programming languages*, pages 81–94, 1990.
23. Gheorghe Paun and Grzegorz Rozenberg. A guide to membrane computing. *Theoretical Computer Science*, 287:73–100, 2002.
24. Marcelo O. Magnasco. Chemical kinetics is Turing universal. *Physical Review Letters*, 78:1190–1193, 1997.
25. Allen Hjelmfelt, Edward D. Weinberger, and John Ross. Chemical implementation of neural networks and Turing machines. *Proceedings of the National Academy of Sciences*, 88:10983–10987, 1991.
26. C. A. Petri. Kommunikation mit automaten. Technical Report Schriften des IMM No. 2., Institut für Instrumentelle Mathematik, Bonn, 1962.
27. Peter J. E. Goss and Jean Peccoud. Quantitative modeling of stochastic systems in molecular biology by using stochastic Petri nets. *Proceedings of the National Academy of Sciences USA*, 95:6750–6755, 1998.
28. E. W. Mayr. Persistence of vector replacement systems is decidable. *Acta Informatica*, 15:309–318, 1981.
29. George S. Sacerdote and Richard L. Tenney. The decidability of the reachability problem for vector addition systems (preliminary version). *9th Annual Symposium on Theory of Computing, Boulder*, pages 61–76, 1977.
30. Emil L. Post. *On the Two-Valued Iterative Systems of Mathematical Logic*. Princeton University Press, New Jersey, 1941.
31. Matthew Cook. *Networks of Relations*. PhD thesis, California Institute of Technology, May 2005.

32. Louis E. Rosier and Hsu-Chun Yen. A multiparameter analysis of the boundedness problem for vector addition systems. *Journal of Computer and System Sciences*, 32:105–135, 1986.
33. Thoralf Skolem. Begründung der elementaren arithmetik durch die rekurrierende denkweise ohne anwendung scheinbarer veränderlichen mit unendlichem ausdehnungsbereich. *Videnskapsselskapets skrifter. 1. Matematisk-naturvidenskabelig klasse*, 6, 1923.
34. Kurt Gödel. über formal unentscheidbare sätze der principia mathematica und verwandter systeme, i. *Monatschefte für Mathematik und Physik*, 38:173–198, 1931.
35. Alan Turing. On computable numbers, with and application to the Entscheidungsproblem. *Proc. London Math. Soc.*, 42(2):230–265, 1936–37.
36. Wilhelm Ackermann. Zum hilbertschen aufbau der reellen zahlen. *Mathematische Annalen*, 99:118–133, 1928.
37. Jacques Herbrand. Sur la non-contradiction de l'arithmétique. *Journal für die reine und angewandte Mathematik*, 166:1–8, 1931.
38. Kurt Gödel. On undecidable propositions of formal mathematical systems. In Martin Davis, editor, *The Undecidable*, pages 39–74. Springer, 1934. Lecture notes taken by Kleene and Rosser at Princeton.
39. Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.
40. Rózsa Péter. *Rekursive Funktionen*. Akadémiai Kiadó, Budapest, 1951.
41. D. Gale. A curious nim-type game. *Amer. Math. Monthly*, 81:876–879, 1974.
42. Marvin L. Minsky. Recursive unsolvability of Post's Problem of 'tag' and other topics in theory of Turing machines. *Annals of Math*, 74:437–455, 1961.
43. Turlough Neary and Damien Woods. A small fast universal Turing machine. Technical Report NUIM-CS-2005-TR-12, Dept. of Computer Science, NUI Maynooth, 2005.
44. David Solovechik. Robust stochastic chemical reaction networks and bounded tau-leaping. *arXiv.org*, arXiv:0803.1030v1, 2008.
45. Matthew Cook. Universality in elementary cellular automata. *Complex Systems*, 15:1–40, 2004.
46. Matthew Cook, Paul W. K. Rothemund, and Erik Winfree. Self-assembled circuit patterns. In *DNA Computing 9*, volume 2943, pages 91–107. Springer-Verlag, 2004.