Lecture Notes for ECS 289A: Theory of Molecular Computation

David Doty

Contents

Introduction

TODO: add section and homework on surface CRNs (see DNA25 about swap reactions on a surface for fun HW ideas)

What this course is about

roughly: **theory of computing** meets **nanotechnology**

For the latter, we largely consider *DNA* nanotechnology, due to

1. the limitations of my own expertise,
2. DNA has more natural information-bearing/processing abilities than other molecules occurring in nature, and
3. more experimentalists in DNA nanotechnology are excited/motivated by the idea of "molecular computation" than in other molecular engineering fields (e.g., graphene).

What does the theory of computing teach us? Some basic principles:

- If we accept just a single hypothesis about the physical world (physicists call these "laws of nature") called the *Church-Turing thesis*,[1] then we can confine our theoretical study to the formal mathematical model of the Turing machine, knowing that what we prove about that model also applies to real physical computing devices that we can build; in particular, proving a *limitation* on a TM implies (roughly) that any real physical device suffers the same limitation.
- Certain problems are inherently difficult or impossible for computers to solve. This is not a statement about how difficult it is for us humans to go about creating an algorithm. It is a statement about the problem itself, and the lack of *any* algorithm, or any efficient algorithm, for the problem.
- There are many different models of computation: TM, finite-state machine, polynomial-time/space TM, Boolean circuit, distributed network with limited communication but unlimited computational ability at each node. None of them is inherently more or less

[1] One way to state the thesis is: *For any device that can be built in our universe that can "reasonably" said to compute a function $f : \{0,1\}^* \to \{0,1\}^*$, f is also computable by a Turing machine.*

correct as a model of *all computing devices*; they are all useful in some sense, even though none of them is an exact description of, e.g., your laptop. When we prove something about one of these models, it does tell us something about your laptop, just *not* as directly as "the theorem applies directly to your laptop in all circumstances." For example, the theorem "*the balanced parentheses language is not regular*" could technically be interpreted as *"since your laptop does not have infinite memory, it cannot decide if parentheses are balanced".* But a more useful interpretation is, *"if you want to write a program (on your laptop) to decide balanced parentheses, somewhere in the program there had better be a stack/array/list/recursion/some kind of unbounded memory."*

In this course, we will apply these principles to *molecular* computing systems. As a general theme, our goal is to learn what are the limitations on our ability to engineer molecular systems that autonomously control their own behavior.

Two specific mathematical models will dominate our discussion in the course, because they are well-studied, and they are appropriate models of certain molecular systems, and (most especially) because I understand them very well:

- algorithmic tile self-assembly
- chemical reaction networks

My hope is that 10-20 years from now, this course would have different models, based on new theoretical and experimental results that haven't happened yet. So I don't want to give the impression that molecular computing is defined, now and forever, as the study of algorithmic tile self-assembly and chemical reaction networks. But I think a great way to learn how to model the world mathematically, in general, is to see how others have already succeeded in modeling these very specific corners of the world using these two models.

Other topics we will cover more sporadically:

- DNA strand displacement
- genetic regulatory networks
- thermodynamics of computing
- cellular automata

More generally, we are interested in any sense in which "computation" appears to be occurring or relevant in some physical system, and in which we want to understand the unique physical constraints of the system (and how those constraints might present different challenges than the constraints imposed by traditional electronic semiconductors).

Chapter 1

Algorithmic Tile Self-Assembly

- abstract Tile Assembly Model introduction: `https://vimeo.com/54214122`
- ISU TAS simulator: `http://self-assembly.net/wiki/index.php?title=ISU_TAS`
- STOC 2000 paper on self-assembling an $n \times n$ square: `http://www.dna.caltech.edu/Papers/squares_STOC.pdf`

1.1 Definitions

There's several ways to mathematically formalize the idea of tile assembly. The following is one particular one I used in a paper.

Fix an alphabet Σ. Σ^* is the set of finite strings over Σ. Given a "discrete object" O, $\langle O \rangle$ denotes a standard encoding of O as an element of Σ^*. $\mathbb{Z}$, $\mathbb{Z}^+$, and $\mathbb{N}$ denote the set of integers, positive integers, and nonnegative integers, respectively. For a set A, $\mathcal{P}(A)$ denotes the power set of A. Given $A \subseteq \mathbb{Z}^2$, the *full grid graph* of A is the undirected graph $G_A^{\mathrm{f}} = (V, E)$, where $V = A$, and for all $u, v \in V$, $\{u, v\} \in E \iff \|u - v\|_2 = 1$; i.e., iff u and v are adjacent on the integer Cartesian plane. A *shape* is a set $S \subseteq \mathbb{Z}^2$ such that G_S^{f} is connected. A shape Υ is a *tree* if G_Υ^{f} is acyclic.

A *tile type* is a tuple $t \in (\Sigma^* \times \mathbb{N})^4$; i.e., a unit square with four sides listed in some standardized order (e.g., south, west, north, east), each side having a *glue* $g \in \Sigma^* \times \mathbb{N}$ consisting of a finite string *label* and nonnegative integer *strength*. We assume a finite set T of tile types, but an infinite number of copies of each tile type, each copy referred to as a *tile*. An *assembly* is a nonempty connected arrangement of tiles on the integer lattice $\mathbb{Z}^2$, i.e., a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$ such that $G_{\mathrm{dom}\ \alpha}^{\mathrm{f}}$ is connected and dom $\alpha \neq \emptyset$. The *shape* $S_\alpha \subseteq \mathbb{Z}^2$ of α is dom α. Two adjacent tiles in an assembly *interact* if the glues on their abutting sides are equal (in both label and strength) and have positive strength. Each assembly α induces a *binding graph* $G_\alpha^{\mathrm{b}} = (V, E)$, where $V = S_\alpha$ and $\{u, v\} \in E \iff \alpha(u)$ and $\alpha(v)$ interact.[1] Given $\tau \in \mathbb{Z}^+$, α is *τ-stable* if every cut of G_α^{b} has weight at least τ, where the weight of

[1] For $G_{S_\alpha}^{\mathrm{f}} = (V_{S_\alpha}, E_{S_\alpha})$ and $G_\alpha^{\mathrm{b}} = (V_\alpha, E_\alpha)$, G_α^{b} is a spanning subgraph of $G_{S_\alpha}^{\mathrm{f}}$: $V_\alpha = V_{S_\alpha}$ and $E_\alpha \subseteq E_{S_\alpha}$.

an edge is the strength of the glue it represents. That is, α is τ-stable if at least energy τ is required to separate α into two parts. When τ is clear from context, we say α is *stable*. Given two assemblies $\alpha, \beta : \mathbb{Z}^2 \dashrightarrow T$, we say α is a *subassembly* of β, and we write $\alpha \sqsubseteq \beta$, if $S_\alpha \subseteq S_\beta$ and, for all points $p \in S_\alpha$, $\alpha(p) = \beta(p)$.

A *tile assembly system* (TAS) is a triple $\mathcal{T} = (T, \sigma, \tau)$, where T is a finite set of tile types, $\sigma : \mathbb{Z}^2 \dashrightarrow T$ is the finite, τ-stable *seed assembly*, and $\tau \in \mathbb{Z}^+$ is the *temperature*. Given two assemblies $\alpha, \beta : \mathbb{Z}^2 \dashrightarrow T$, we write $\alpha \to_1^{\mathcal{T}} \beta$ if $\alpha \sqsubseteq \beta$, $|S_\beta \setminus S_\alpha| = 1$, and, letting $S_\beta \setminus S_\alpha = \{\mathbf{p}\}$ the cut $(\{\mathbf{p}\}, S_\alpha)$ has strength $\geq \tau$. In this case we say α *$\mathcal{T}$-produces β in one step.*[2] If $t = \beta(\mathbf{p})$, we write $\beta = \alpha + (\mathbf{p} \mapsto t)$. The *$\mathcal{T}$-frontier* of α is the set $\partial^{\mathcal{T}}\alpha = \bigcup_{\alpha \to_1^{\mathcal{T}} \beta} S_\beta \setminus S_\alpha$, the set of empty locations at which a tile could stably attach to α.

A sequence of $k \in \mathbb{Z}^+ \cup \{\infty\}$ assemblies $\alpha_0, \alpha_1, \ldots$ is a *$\mathcal{T}$-assembly sequence* if, for all $1 \leq i < k$, $\alpha_{i-1} \to_1^{\mathcal{T}} \alpha_i$. We write $\alpha \to^{\mathcal{T}} \beta$, and we say α *$\mathcal{T}$-produces* β (in 0 or more steps) if there is a $\mathcal{T}$-assembly sequence $\alpha_0, \alpha_1, \ldots$ of length $k = |S_\beta \setminus S_\alpha| + 1$ such that 1) $\alpha = \alpha_0$, 2) for all $0 \leq i < k$, $\alpha_i \sqsubseteq \beta$, and 3) $S_\beta = \bigcup_{0 \leq i < k} S_{\alpha_i}$. We say that β is the *result* of the assembly sequence, and we say the assembly sequence is *terminal* if β is terminal. If k is finite then it is routine to verify that $\beta = \alpha_{k-1}$.[3] We say α is *$\mathcal{T}$-producible* if $\sigma \to^{\mathcal{T}} \alpha$, and we write $\mathcal{A}[\mathcal{T}]$ to denote the set of $\mathcal{T}$-producible assemblies.

An assembly α is *$\mathcal{T}$-terminal* if α is τ-stable and $\partial^{\mathcal{T}}\alpha = \emptyset$. We write $\mathcal{A}_\square[\mathcal{T}] \subseteq \mathcal{A}[\mathcal{T}]$ to denote the set of $\mathcal{T}$-producible, $\mathcal{T}$-terminal assemblies. A TAS $\mathcal{T}$ is *directed (a.k.a., deterministic, confluent)* if the poset $(\mathcal{A}[\mathcal{T}], \to^{\mathcal{T}})$ is directed; i.e., if for each $\alpha, \beta \in \mathcal{A}[\mathcal{T}]$, there exists $\gamma \in \mathcal{A}[\mathcal{T}]$ such that $\alpha \to^{\mathcal{T}} \gamma$ and $\beta \to^{\mathcal{T}} \gamma$.[4]

Let $S \subseteq \mathbb{Z}^2$ be a shape. We say that a TAS $\mathcal{T}$ *strictly self-assembles* S if, for all $\alpha \in \mathcal{A}_\square[\mathcal{T}]$, $S_\alpha = S$; i.e., if every terminal assembly produced by $\mathcal{T}$ has shape S. If $\mathcal{T}$ strictly self-assembles some shape S, we say that $\mathcal{T}$ is *strict*. Note that the implication "$\mathcal{T}$ is directed $\implies$ $\mathcal{T}$ is strict" holds, but the converse does not hold.

Let $P \subseteq \mathbb{Z}^2$ be a set (not necessarily a connected shape). We say that $\mathcal{T}$ *weakly self-assembles* P if there is a subset $B \subseteq T$ (the "black tiles") such that, for all $\alpha \in \mathcal{A}_\square[\mathcal{T}]$, $P = \alpha^{-1}(B)$, i.e., the set of points with a black tile is P.[5]

When $\mathcal{T}$ is clear from context, we may omit $\mathcal{T}$ from the notation above and instead write $\to_1$, $\to$, $\partial\alpha$, *frontier*, *assembly sequence*, *produces*, *producible*, and *terminal*.

---

[2] Intuitively $\alpha \to_1^{\mathcal{T}} \beta$ means that α can grow into β by the addition of a single tile. It is easy to see that if α is τ-stable, then so is β: the cut $(\{\mathbf{p}\}, S_\alpha)$ is stable by definition, and any other cut of β is simply a cut of α (with strength $\geq \tau$ by hypothesis), possibly with extra edges due to the presence of the tile at $\mathbf{p}$.

[3] If we had defined the relation $\to^{\mathcal{T}}$ based on only finite assembly sequences, then $\to^{\mathcal{T}}$ would be simply the reflexive, transitive closure $(\to_1^{\mathcal{T}})^*$ of $\to_1^{\mathcal{T}}$. But this would mean that no infinite assembly could be produced from a finite assembly, even though there is a well-defined, unique "limit assembly" of every infinite assembly sequence.

[4] The following convenient characterizations of "directed" are routine to verify. $\mathcal{T}$ is directed if and only if $|\mathcal{A}_\square[\mathcal{T}]| = 1$. $\mathcal{T}$ is *not* directed if and only if there exist $\alpha, \beta \in \mathcal{A}[\mathcal{T}]$ and $\mathbf{p} \in S_\alpha \cap S_\beta$ such that $\alpha(\mathbf{p}) \neq \beta(\mathbf{p})$.

[5] In the first tile system shown the first day, the shape strictly self-assembled was the entire second quadrant, and, considering the orange tile types to be the set B, the set weakly self-assembled was the discrete Sierpinkski triangle.

1.2 Basic results on producibility

The following observation states that if a tile attachment is stable, then it is also stable in the presence of additional tiles.

Observation 1.2.1. *Let $\alpha \sqsubseteq \beta$ be stable assemblies and $\mathbf{p} \in \mathbb{Z}^2 \setminus S_\beta$ such that $\alpha + (\mathbf{p} \mapsto t)$ is stable. Then $\beta + (\mathbf{p} \mapsto t)$ is stable.*

Proof. Since β is stable, and glue strengths are nonnegative, in $\beta + (\mathbf{p} \mapsto t)$, the only cut that could possibly be unstable is the one between t and the rest of β. But since $\alpha \sqsubseteq \beta$ and $\alpha + (\mathbf{p} \mapsto t)$ is stable, this cut is also stable since glue strengths are nonnegative and β only has *extra* tiles on the other side of the cut, compared to α. □

The following is a useful tool, first proven in Paul Rothemund's Ph.D. thesis. Note that it is *not* saying $\alpha \to \beta$; indeed, this may not be the case. It is saying that if we can add some tiles to α to get γ, then if some of them are already there (those in $S_\beta \setminus S_\alpha$), then the remaining tile additions form a valid assembly sequence.

Lemma 1.2.2 (Rothemund's Lemma)**.** *Let $\alpha \sqsubseteq \beta \sqsubseteq \gamma$ be assemblies such that $\alpha \to \gamma$. Then $\beta \to \gamma$.*

Proof. Let $(\alpha = \alpha_0, \alpha_1, \ldots)$ be an assembly sequence showing that $\alpha \to \gamma$. Let $\mathbf{p}_0, \mathbf{p}_1, \ldots \in \mathbb{Z}^2$ be the sequence defined by $\{\mathbf{p}_i\} = S_{\alpha_{i+1}} \setminus S_{\alpha_i}$, i.e., $\mathbf{p}_i$ is the position of the i'th added tile, and let $t_i = \gamma(\mathbf{p}_i)$ be the tile type added. Let $i_0 < i_1 < \ldots$ be the subsequence such that $S_\gamma \setminus S_\beta = \{\mathbf{p}_{i_0}, \mathbf{p}_{i_1}, \ldots\}$, i.e., the subsequence representing tile attachments to γ outside of β. Define the assembly sequence $(\beta = \beta_0, \beta_1, \ldots)$ by $\beta_{j+1} = \beta_j + (\mathbf{p}_{i_j} \mapsto t_{i_j})$, i.e., adding tiles to $S_\gamma \setminus S_\beta$ in the order they were added to α (but skipping tiles already in S_β). Then for each j, $\alpha_{i_j} \sqsubseteq \beta_j$, so Observation 1.2.1 implies that $\beta_j + (\mathbf{p}_{i_j} \mapsto t_{i_j})$ is stable. Thus the assembly sequence is valid and shows that $\beta \to \gamma$. □

We say an assembly sequence $(\alpha_0, \alpha_1, \ldots)$ is *fair* if either it is finite and its result is terminal, or if it is infinite and, for all $i \in \mathbb{N}$ and all $\mathbf{p} \in \partial\alpha_i$, there exists $j \in \mathbb{N}$ such that $\mathbf{p} \in S_{\alpha_j}$, i.e., if every point that is on the frontier at any point in the assembly sequence, at some later point gets a tile. (The former condition about finite assembly sequences is a technical condition to make the following result hold.)

One way to algorithmically create a fair assembly sequence is with a first-in, first-out queue; put the frontier of the seed in the queue, choose a frontier location $\mathbf{p}$ from the front of the queue, pick a tile to add to $\mathbf{p}$, and check the ≤ 3 locations adjacent to $\mathbf{p}$ to see if they are now on the frontier. If so, add them to the back of the queue. At any time, the queue is finite, so eventually all frontier locations get a tile.

Lemma 1.2.3. *Let $(\alpha_0, \alpha_1, \ldots)$ be a fair assembly sequence. Then its result is terminal.*

Proof. This holds trivially if the sequence is finite, so assume it is infinite. Let γ be the result of $(\alpha_0, \alpha_1, \ldots)$. Suppose for the sake of contradiction that γ is not terminal, and let $\mathbf{p} \in \partial\gamma$.

Then there are tiles in adjacent locations to $\mathbf{p}$ whose glue strengths sum to the temperature τ and a tile t with matching glues. Call this set of positions P. Since $S_\gamma = \bigcup_{i=0}^{\infty} S_{\alpha_i}$, there exists i such that $P \subseteq S_{\alpha_i}$, i.e., these tiles are present after some finite number of tile attachments. Thus $\mathbf{p} \in \partial\alpha_i$. Since the assembly sequence is fair, there exists $j \in \mathbb{N}$ such that $\mathbf{p} \in S_{\alpha_j} \subseteq S_\gamma$, which contradicts the assumption that $\mathbf{p} \in \partial\gamma$ since $\partial\gamma \cap S_\gamma = \emptyset$. □

Since a fair assembly sequence can be applied to any assembly, we have the following.

Corollary 1.2.4. *For all $\alpha \in \mathcal{A}[\mathcal{T}]$, there is $\gamma \in \mathcal{A}_\square[\mathcal{T}]$ such that $\alpha \to \gamma$.*

1.3 Information theoretic lower bounds on tile complexity

This section has two proofs of what is often called a "standard information-theoretic lower bound" on tile complexity.

1.3.1 Simple counting argument

Theorem 1.3.1. *There are infinitely many $n \in \mathbb{N}$ such that $\mathrm{C}^{\mathrm{tc}}(S_n) \geq \frac{1}{4}\frac{\log n}{\log\log n}$.*

Proof. Let $m \in \mathbb{N}$. We will show such an n exists in the set $\{m+1, m+2, \ldots, 2m\}$.

Let $k = \frac{1}{4}\frac{\log m}{\log\log m}$. How many TAS's are there with exactly k tile types? There are at most $4k$ distinct glues. For each tile type, there are thus $(4k)^4$ ways to choose the glues for that tile type.

Thus there are $(4k)^{4k}$ ways to choose all k tile types. Since there are k ways to choose the seed tile, there are $k(4k)^{4k}$ total TAS's with exactly k tile types. Thus there are at most $\sum_{i=1}^{k} i(4i)^{4i} < \sum_{i=1}^{k} k(4k)^{4k} = k^2(4k)^{4k}$ total TAS's with *at most* k tile types.

To see that $k^2(4k)^{4k} < m$, it suffices to show that $\log\left(k^2(4k)^{4k}\right) < \log m$:

$$\begin{aligned}
\log\left(k^2 \cdot (4k)^{4k}\right) &= \log\left(k^2 \cdot \left(\frac{\log m}{\log\log m}\right)^{\frac{\log m}{\log\log m}}\right) \\
&= 2\log k + \frac{\log m}{\log\log m}\log\left(\frac{\log m}{\log\log m}\right) \\
&= 2\log k + \frac{\log m}{\log\log m}(\log\log m - \log\log\log m) \\
&< 2\log k + \frac{\log m}{\log\log m}(\log\log m - 2) \\
&= 2\log k + \log m - 2\frac{\log m}{\log\log m} \\
&= 2\log\left(\frac{1}{4}\frac{\log m}{\log\log m}\right) + \log m - 2\frac{\log m}{\log\log m} \\
&= 2\log\frac{1}{4} + 2\log\frac{\log m}{\log\log m} + \log m - 2\frac{\log m}{\log\log m} \\
&< \log m - \frac{\log m}{\log\log m} \\
&< \log m.
\end{aligned}$$

Since $k^2(4k)^{4k} < m$, there are strictly less than m different TAS's with $\leq k$ tile types. By the pigeonhole principle, there must be some value of $n \in \{m+1, \ldots, 2m\}$ such that no TAS with $\leq k$ tile types strictly self-assembles S_n. Since $m < n$, we have $k = \frac{1}{4}\frac{\log m}{\log\log m} < \frac{1}{4}\frac{\log n}{\log\log n}$. □

In fact, we can even get the conclusion that for "most" n, $\mathrm{C}^{\mathrm{tc}}(S_n) \geq \frac{1}{4}\frac{\log n}{\log\log n}$. From the second-to-last lines above we have that the number K of TAS's with at most k tile types obeys

$$\log K < \log m - \frac{\log m}{\log\log m} < \log(m)\left(1 - \frac{1}{\log\log m}\right).$$

Exponentiating both sides,

$$K < 2^{\log(m)\left(1-\frac{1}{\log\log m}\right)} = m^{1-\frac{1}{\log\log m}} < m^{1/2} < 0.01m,$$

In other words, there are m squares S_n defined by n in the range $\{m+1, \ldots, 2m\}$, but there are less than $0.01m$ TAS's with at most $\frac{1}{4}\frac{\log m}{\log\log m} < \frac{1}{4}\frac{\log n}{\log\log n}$ tile types. Therefore for large enough m, more than 99% of all squares S_n with $n \in \{m+1, \ldots, 2m\}$ require a TAS with more than $\frac{1}{4}\frac{\log n}{\log\log n}$ tile types.

1.3.2 Kolmogorov complexity proof

The theory of Kolmogorov complexity is not *required* to prove tile complexity lower bounds; it is merely a convenient way to structure the argument if you are familiar with algorithmic information theory. Section 1.3.1 has a simple proof based directly on direct counting. However, it is common to see Kolmogorov complexity proofs, which outsource some of the counting and tedious algebra to the more general theory, and allow for shorter proofs that are easier to understand, provided you already understand Kolmogorov complexity.

Let U be a universal Turing machine, and for all $x \in \{0,1\}^*$, define the *Kolmogorov complexity* of x to be

$$\mathrm{C}(x) = \min_{p \in \{0,1\}^*} \{ \ |p| \ | \ U(p) = x \ \},$$

the length of the shortest program that prints x and halts.

If x is "easy to describe" (e.g., $x = 0^n$ for large n), then $\mathrm{C}(x) \ll |x|$. All strings x obey $\mathrm{C}(x) \leq |x| + O(1)$, but "most" strings x have the property that $\mathrm{C}(x) \geq |x|$.[6] We call such strings *algorithmically random*.

The proof that most strings are algorithmically random is similar to the counting proof in Section 1.3.1: simply put, there aren't enough short programs to go around. One very easy observation is that for each n, *at least one* string $x \in \{0,1\}^n$ obeys $C(x) \geq |x|$. There are 2^n elements of $\{0,1\}^n$ (strings x we want to output) but only $2^n - 1$ elements of $\{0,1\}^{<n}$ (programs p that are shorter than length n), so by the pigeonhole principle, at least one string $x \in \{0,1\}^n$ is not output by any program of shorter length. By being a bit more careful, we can actually show most strings have this property, but leave this as an exercise to the reader.

If $n \in \mathbb{N}$, let $\mathrm{C}(n) = \mathrm{C}(\mathrm{bin}(n))$, where $\mathrm{bin}(n) \in \{0,1\}^*$ is the binary expansion of n. Note that $|\mathrm{bin}(n)| = \lfloor \log n \rfloor + 1$, so $\mathrm{C}(n) \leq \log n + O(1)$ for all n and $\mathrm{C}(n) \geq \log n$ for "most" n.

What does any of this matter? The aTAM can be *simulated on a computer*. (ISU TAS is one such simulator.) Thus, if a tile assembly system $\mathcal{T}$ produces terminal assemblies that all have some property, and that property can be expressed as a binary string x,[7] then $\mathcal{T}$, together with the aTAM simulator to produce a terminal assembly α, together with some code to read x from α, is a program to print x. This idea is the basis of the proof of the following theorem, due to Rothemund and Winfree.

For a tile assembly system $\mathcal{T} = (T, \sigma, \tau)$, we define the *size* of $\mathcal{T}$ to be $|\mathcal{T}| = |T|$. Given a shape $S \subseteq \mathbb{Z}^2$ and $\tau \in \mathbb{N}$, define the *temperature-τ tile complexity*

$$\mathrm{C}^{\mathrm{tc}}_{\tau}(S) = \min \{ \ |\mathcal{T}| \ | \ \mathcal{T} = (T, \sigma, \tau) \text{ is singly-seeded and strictly self-assembles } S \ \}.$$

We write $\mathrm{C}^{\mathrm{tc}}(S)$ to denote $\mathrm{C}^{\mathrm{tc}}_2(S)$. For all $n \in \mathbb{N}$, define $S_n = \{0, 1, \ldots, n-1\}^2$ to be the $n \times n$ square. We proved earlier that for all $n \in \mathbb{N}$, $\mathrm{C}^{\mathrm{tc}}(S_n) = O(\log n)$. We now show it's not possible to do "much" better for most values of n.

[6] One way to formalize "most" is that, for all but finitely many $n \in \mathbb{N}$, at least 99% of all strings $x \in \{0,1\}^n$ obey $\mathrm{C}(x) \geq |x|$.

[7] For example, if $\mathcal{T}$ produces an $n \times n$ assembly α, then $\mathrm{bin}(n)$ can be "read off" from α.

Theorem 1.3.2. *For most* $n \in \mathbb{N}$, $\mathrm{C}^{\mathrm{tc}}(S_n) = \Omega\left(\frac{\log n}{\log\log n}\right)$.

Proof. Most n satisfy $\mathrm{C}(n) \geq \log n$, so we focus on such algorithmically random n. Let $p_n \in \{0,1\}^*$ be the description of the Turing machine that does the following. It simulates the aTAM on $\mathcal{T}_n$ until a terminal assembly α is produced, then prints bin(width(S_α)). By assumption, S_α is an $n \times n$ square, so p_n is a program to print the string bin(n).

$|p_n|$ is dominated by the description of $\mathcal{T}_n$; the rest has length $O(1)$ with respect to n. Thus, $\log n \leq \mathrm{C}(n) \leq |p_n| = O(1) + \#$(bits needed to describe $\mathcal{T}_n$).

How many bits are needed to describe $\mathcal{T}_n$? We must describe T and σ. There are at most $4|T|$ total glues, so each glue can be described by $2 + \log 4|T| = 4 + \log|T|$ bits: 2 to describe the strength in $\{0,1,2\}$ and $\log 4|T|$ to describe the glue label. Thus each $t \in T$ can be described by $16 + 4\log|T|$ bits, so T can be described with $16|T| + 4|T|\log|T|$ bits.

Since $|\mathrm{dom}\,\sigma| = 1$, it requires $\log|T|$ bits to say which tile type is the seed.

Thus $\log n \leq |p| \leq O(1) + 16|T| + 4|T|\log|T| + \log|T| = O(|T|\log|T|)$, so $|T| \geq \Omega\left(\frac{\log n}{\log\log n}\right)$. □

Note that nothing special about squares was used in the proof of Theorem 1.3.2. The theorem applies to any shape such that a string x can be easily computed from the terminal assembly(ies) of the tile assembly system $\mathcal{T}$: whenever x is algorithmically random, then $|\mathcal{T}| \geq \frac{|x|}{\log|x|}$.

To summarize the big picture: The general strategy for using Kolmogorov complexity to show that "most" elements of some set have a certain tile complexity is this. Show that any tile system with k tile types can be described in $f(k)$ bits (in this case, $f(k) = \Theta(k\log k)$, but for other models it might be something else; for example allowing a "non-diagonal" glue function in which unequal glues can bind with positive strength requires more bits to encode than the standard model). The tile system, with only $O(1)$ additional bits (such as a simulator) can be turned into a program of $f(k) + O(1)$ bits that outputs some string x (in our case, x was the binary expansion of the width of the terminal assembly). Since most strings x require a program of size at least $|x|$, we conclude that $f(k) \geq |x|$, so $k \geq f^{-1}(|x|)$.

1.4 Meeting the $\frac{\log n}{\log\log n}$ lower bound for assembling an $n \times n$ square

We have seen that $\mathrm{C}^{\mathrm{tc}}(S_n) = O(\log n)$ for all n, and $\mathrm{C}^{\mathrm{tc}}(S_n) = \Omega(\frac{\log n}{\log\log n})$ for most n. Can we revisit the upper bound and show how to use only $O(\frac{\log n}{\log\log n})$ tile types to strictly self-assemble an $n \times n$ square?

Recall how the construction using $O(\log n)$ tile types worked. $\log n$ unique tile types encode n in binary in a $1 \times \log n$ row of tiles, and $O(1)$ additional tile types "read" the value n, written in binary in $\log n$ glues on this row, and assemble in an $n \times n$ square surrounding the row. Obviously to write n in binary requires $\log n$ total *tiles*, but we have already seen how algorithmic tile assembly can reuse the same tile *types* in clever ways. Can we somehow encode n using fewer tile types, that algorithmically "reads" n from $\frac{\log n}{\log\log n}$ tiles

that (somehow) encode n, and assembles around them into a structure that encodes n in binary in $\log n$ glues?

To solve this problem, it helps to think about *why* the $O(\log n)$ construction is not optimal; where is the waste? Each tile type in the first row is unique; they are all elements of a set of cardinality $m = \log n$. In principle, in an information theoretic sense, if I have m unique elements in a set, each time I communicate one of them, I am sending $\log m$ bits of information. However, we used them wastefully, encoding only a single bit of n for each tile type. But with m different tile types, we could theoretically encode $\log m$ bits per tile type, not just one bit.

So, this is the key to the idea: encode more than one bit of n per tile type. Another way to think of this is that the glues of each of these tiles will represent a digit of n in a larger base than 2. For example, encoding n in base 8 means each octal digit represents 3 bits of n. But, to be useful to the tiles that actually grow the $n \times n$ square, this encoding of n must first be translated from this higher base to base 2. This will itself require some new tile types.

Here is how we then choose which base in which to encode n. By increasing the base b in which we encode n, we require 1) fewer tile types to write down n in base b, but 2) more tile types to convert it from base b to base 2. It turns out, if we analyze the construction we are about to describe, that the best base to choose, which balances these considerations (1) and (2) against each other, is base $b \approx \frac{\log n}{\log \log n}$. It is convenient to choose b to be a power of two, so we let $b = 2^k$ such that $2^k \leq \frac{\log n}{\log \log n} < 2^{k+1}$. Each symbol in base b then encodes k bits of n, and thus requires $\frac{\log n}{k} = O(\frac{\log n}{\log \frac{\log n}{\log \log n}}) = O(\frac{\log n}{\log \log n - \log \log \log n}) = O(\frac{\log n}{\log \log n})$ total tile types to hard-code a row representing n in base b.

Figure 1.1 shows how this conversion process works, and the caption explains it. As we saw, the blue "seed row" requires $O(\frac{\log n}{\log \log n})$ tile types.

Each group of tile types has specialized versions for most and least significant digits, so we'll just count the tile types for the internal version, keeping in mind that we really need up to 3 times that number. There are at most $2b = O(\frac{\log n}{\log \log n})$ "copy to north" tiles and 4 "copy to east" tiles.

For the "base convert" tiles, we consider each position in the orange column of tiles that converts from base b to base 2. The total number of digits is b, so the lowest tile has b different types. Each subsequent tile type has half the total number of possibilities (since one bit is discarded each time), so the total number of tile types, summing over all vertical positions (assuming k total positions), is $\sum_{i=1}^{k} 2^i < 2^{k+1} = O(b) = O(\frac{\log n}{\log \log n})$.

1.5 Theory of tile complexity for more general shapes

show TM simulation

Intuitively, most of the complexity of an $n \times n$ square is captured by its width; geometry does not play much of a role. What about shapes with more complicated geometry? One

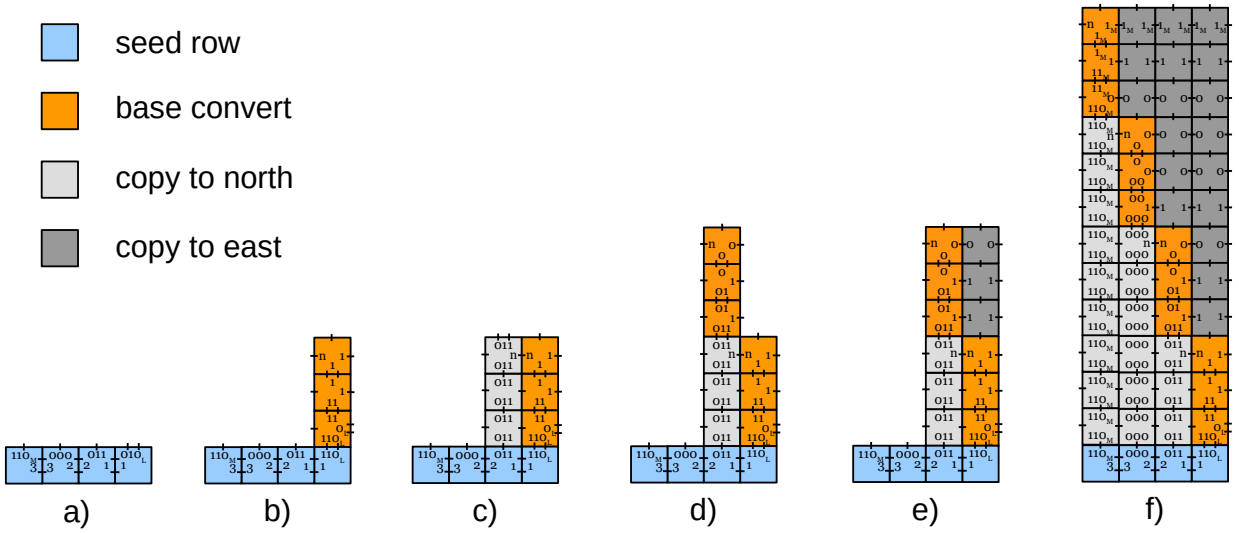

Figure 1.1: Tiles to convert from base $b = 2^k$ to base 2. In this example, $k = 3$, so $b = 8$, and $n = 110000011010_2$ in binary. (If $n = 110000011010_2$, then $\frac{\log n}{\log \log n} \approx 3.3$, which means we should have $k = 2$ and $b = 4$, but for instructional purposes we choose $k = 3$ and $b = 8$ instead.) a) The seed row hardcodes n in base b (using k-bit strings to represent each base-b digit), with least and most significant digits specially marked. b) The base conversion tiles translate the least significant base b digit to base 2. The individual bits are advertised on east-facing glues. c) The next base-b digit needs to have its bits represented above the bits from first base-b digit. Thus, before doing the conversion, the digit must first be copied to the north. d) The same set of base conversion tiles (other than least significant bit) are used to convert the second digit to binary. e) Since the bits from the second are shifted west from the bits from the first digit, they must be copied to the east to line up properly. f) Remaining digits are copied and converted in a similar manner.

would expect that even if such a shape has a compact algorithmic description, self-assembly that simulates this algorithm may not be possible to execute *within* the shape. We have already seen that one particular shape, a $1 \times n$ line, requires far more tile types than the information theoretic lower bound of $\frac{\log n}{\log \log n}$. Intuitively, the reason is that although nothing stops a *program* of length $\log n$ from outputting a description of a $1 \times n$ line, and although the aTAM can simulate any program, it cannot simulate a program *within the confines of the line*. In other words, the *geometric constraints* of the shape prevent its efficient self-assembly.

But, Soloveichik and Winfree showed that the algorithmic complexity of a shape *is* intimately related to its tile complexity, so long as we are willing to accept all the various *scalings* of a shape as equivalent.

Given a shape $S \subseteq \mathbb{Z}^2$ and $c \in \mathbb{Z}^+$, define the *c-scaling* of S as $S^c = \{ (x, y) \mid (\lfloor \frac{x}{c} \rfloor, \lfloor \frac{y}{c} \rfloor) \in S \}$. Given a finite shape $S \subset \mathbb{Z}^2$, define $\mathrm{C}(S)$ to be the *Kolmogorov complexity* of S, the length in bits of the shortest program printing the points in the shape. We say two shapes S_1, S_2 are *scale-equivalent*, and we write $S_1 \simeq S_2$, if there exist $c_1, c_2 \in \mathbb{N}$ and a shape $S \subseteq \mathbb{Z}^2$ such that $S_1 = S^{c_1}$ and $S_2 = S^{c_2}$, i.e., if they are both scalings (perhaps trivially by scale factor 1) of some shape.

Define the *scale-free Kolmogorov complexity* of S to be $\mathrm{C}_{\mathrm{sf}}(S) = \min_{S' \simeq S} \mathrm{C}(S)$. Define the *scale-free (temperature 2) tile complexity* of S to be $\mathrm{C}^{\mathrm{tc}}_{\mathrm{sf}}(S) = \min_{S' \simeq S} \mathrm{C}^{\mathrm{tc}}(S)$.

Theorem 1.5.1. *For every finite shape S, $\mathrm{C}^{\mathrm{tc}}_{\mathrm{sf}}(S) = \Theta\left(\frac{\mathrm{C}_{\mathrm{sf}}(S)}{\log \mathrm{C}_{\mathrm{sf}}(S)}\right)$.*

Proof. We prove the upper and lower bounds separately.

$\mathrm{C}^{\mathrm{tc}}_{\mathrm{sf}}(S) = \Omega\left(\frac{\mathrm{C}_{\mathrm{sf}}(S)}{\log \mathrm{C}_{\mathrm{sf}}(S)}\right)$: Intuitively, this is true for the same reason as the $\Omega(\frac{\log n}{\log \log n})$ tile complexity lower bound for strictly self-assembling an $n \times n$ square: a smaller tile set would lead to a program of length less than $\log n$ for printing n, which is a contradiction for algorithmically random n.

Let $\mathcal{T}$ be a singly-seeded temperature 2 TAS that strictly self-assembles S^c for some $c \in \mathbb{Z}^+$. Then by simulating $\mathcal{T}$ until it produces a terminal assembly α, then printing the points in S_α, we obtain a program p for S_α. We showed in the proof of Theorem 1.3.2 that $|p| = O(|\mathcal{T}| \log |\mathcal{T}|)$, i.e., $|\mathcal{T}| = \Omega\left(\frac{|p|}{\log |p|}\right)$. Since $\mathcal{T}$ was an arbitrary TAS strictly self-assembling S_α, this shows that $\mathrm{C}^{\mathrm{tc}}_{\mathrm{sf}}(S) = \Omega\left(\frac{|p|}{\log |p|}\right) = \Omega\left(\frac{\mathrm{C}_{\mathrm{sf}}(S)}{\log \mathrm{C}_{\mathrm{sf}}(S)}\right)$.

$\mathrm{C}^{\mathrm{tc}}_{\mathrm{sf}}(S) = O\left(\frac{\mathrm{C}_{\mathrm{sf}}(S)}{\log \mathrm{C}_{\mathrm{sf}}(S)}\right)$: In Section 1.4, we showed the following (on the way to making a statement about squares): For each $p \in \{0,1\}^*$, there is a TAS $\mathcal{T}$ with $|\mathcal{T}| = \Omega\left(\frac{|p|}{\log |p|}\right)$ that self-assembles a $|p| \times \frac{|p|}{\log |p|}$ rectangle that encodes p in the glues of the tiles on its north side (rotated). This will be our starting point; what should we choose for p? It will be such that $|p| = \mathrm{C}^{\mathrm{tc}}_{\mathrm{sf}}(S)$, i.e., a smallest program printing some scaling S^c of S.

(See paper: `http://www.dna.caltech.edu/Papers/SAshapes_SICOMP2007.pdf`)

□

There are three key ingredients to making the second part of the proof ($\mathrm{C}^{\mathrm{tc}}_{\mathrm{sf}}(S) = O\left(\frac{\mathrm{C}_{\mathrm{sf}}(S)}{\log \mathrm{C}_{\mathrm{sf}}(S)}\right)$).

1. $O\left(\frac{k}{\log k}\right)$ tile types can "encode" any string $p \in \{0,1\}^k$, in the sense that they grow from a single seed tile into a $\frac{k}{\log k} \times k$ rectangle with each of the k glues on the north side being p.

   This was the key step in showing how to assemble any $n \times n$ square from $O\left(\frac{\log n}{\log \log n}\right)$ tile types.

2. Given any Turing machine M, there is a single set of tile types T such that, starting from any row (such as the north row of the assembly in the previous step) that encodes an input p to M, simulates $M(p)$ in the sense that the t'th row of the assembly that

grows represents the configuration of $M(p)$ after t steps. In this proof, we take M to be a universal Turing machine U and therefore interpret p as a program (in particular, a shortest program printing the shape S). Thus, $O(1)$ tile types are needed for this step, since we will always just be simulated U in this step.

In the proof there is some additional computation that must be done: finding a spanning tree of S, computing the children of the "current point" in S,[8] but the ability to do these is just another consequence of the ability to simulate U, which can do any computation that's possible to do.

3. There are $O(1)$ tile types that, given a row of glues encoding the following information:

   - the points in S
   - the "current point" in S

   computes the 0-3 children points of the "current" point in S, grows in the direction of the children points, carrying the description of S in each direction, and writes in the adjacent square block (representing the child point) the description of S and the child point, with appropriate glues to let the same computation repeat in that block (i.e., computing *its* children and growing in those directions), which will repeat in every block in S.

The latter two ingredients represent a constant set of tile types T that produces S^c from a $1 \times \frac{\mathrm{C_{sf}}(S)}{\log \mathrm{C_{sf}}(S)}$ row encoding the program for S. In this case, we started from a single seed and used $O\left(\frac{\mathrm{C_{sf}}(S)}{\log \mathrm{C_{sf}}(S)}\right)$ tile types to grow a row encoding a shortest program for S, but we could also imagine simply declaring that the seed assembly is this row.

Thus, calling the tile set for the last two ingredients T, we have that T is a single tile set that is "universally programmable" (by seeding with an appropriate program) for building any finite shape S, if scaling factors are ignored.

[8] The "current" point is the one represented by the $c \times c$ block of tiles currently being assembled, which starts as the point $(0, 0)$, but these same tiles will repeat this action for other points.

Thursday, Jan. 25:

- strict vs. weak vs. computable
  show there is a shape that
  - can't be strictly self-assembled
  - is computable and can't be strictly self-assembled
  - is computable and can't be weakly self-assembled
- directed strict self-assembly versus strict self-assembly

Tues., Jan. 30:

- temperature 1: "proof" that it has to grow periodic patterns
- intrinsic universality
- other models of self-assembly
  - kTAM, proofreading
  - concentration programming
  - temperature programming
  - hierarchical self-assembly
  - staged self-assembly

Chapter 2

Chemical Reaction Networks

Informally, a CRN is a set of reactions like $A \to B + C$ or $X + Y \to X + 2Z$.

Traditionally this model is used as a descriptive language to model natural systems of chemical reactions. However, we want to use it as a *programming* language to describe artificial chemical systems that we want to engineer. The idea is that although in a well-mixed system we cannot control the order in which molecules collide with each other, we can control how they react when they do collide.[1]

With this in mind, we ask, "*If it is possible to program chemical species by specifying reactions of our choosing, but once mixed we cannot control the order in which they will react, what can we compute with such a programming language?*" In the discrete model, a *configuration* is a multiset of nonnegative integer counts of each chemical species, e.g., $\{3A, B, 5Z\}$. We say a CRN computes a function $f : \mathbb{N} \to \mathbb{N}$ if for all $n \in \mathbb{N}$, starting from configuration $\{nX\}$, it is "guaranteed" to eventually reach a configuration $\{f(n)Y, \ldots\}$.

Let's devise CRNs to compute the following functions:

- $2n$

$$X \to 2Y$$

- $3n$

$$X \to 3Y$$

- $n/2$

$$2X \to Y$$

- $n/3$

$$3X \to Y$$

[1] We will see later in the course when we get to DNA strand displacement, one particular example of how one might go about creating artificial chemicals that experience reactions of our choosing.

- $3n$ using only $\leq$ 2-product reactions:

$$\begin{aligned} X &\rightarrow Y_1 + Y_2 \\ Y_1 &\rightarrow Y \\ Y_2 &\rightarrow 2Y \end{aligned}$$

- $n/3$ using only bimolecular reactions, start in configuration $\{1L_0, nX\}$:

$$\begin{aligned} L_0 + X &\rightarrow L_1 \\ L_1 + X &\rightarrow L_2 \\ L_2 + X &\rightarrow L_0 + Y \end{aligned}$$

- $n/3$ using only bimolecular reactions *and* starting in configuration $\{nX\}$? You try it!
- $n_1 + n_2$

$$\begin{aligned} X_1 &\rightarrow Y \\ X_2 &\rightarrow Y \end{aligned}$$

- $n_1 - n_2$ (assume $n_1 \geq n_2$)

$$\begin{aligned} X_1 &\rightarrow Y \\ X_2 + Y &\rightarrow \emptyset \end{aligned}$$

- $3n_1 - n_2/2$

$$\begin{aligned} X_1 &\rightarrow 3Y \\ 2X_2 + Y &\rightarrow \emptyset \end{aligned}$$

- $\min(n_1, n_2)$

$$X_1 + X_2 \rightarrow Y$$

- $\max(n_1, n_2)$

$$\begin{aligned} X_1 &\rightarrow Y + Z_1 \\ X_2 &\rightarrow Y + Z_2 \\ Z_1 + Z_2 &\rightarrow K \\ K + Y &\rightarrow \emptyset \end{aligned}$$

2.1 Definitions

If Λ is a finite set, write $\mathbb{N}^\Lambda$ to denote $\{f : \Lambda \to \mathbb{N}\}$.[2] Given $X \in \Lambda$ and $\mathbf{c} \in \mathbb{N}^\Lambda$, $\mathbf{c}(X)$ is the *count of X in* $\mathbf{c}$; if $\mathbf{c}$ is clear from context, write $\#X$ to denote $\mathbf{c}(X)$.[3] Write $\mathbf{c} \leq \mathbf{c}'$ to denote that $\mathbf{c}(X) \leq \mathbf{c}'(X)$ for all $X \in \Lambda$, and $\mathbf{c} < \mathbf{c}'$ if $\mathbf{c} \leq \mathbf{c}'$ and $\mathbf{c} \neq \mathbf{c}'$. If $\Delta \subset \Lambda$, we view a vector $\mathbf{c} \in \mathbb{N}^\Delta$ equivalently as a vector $\mathbf{c} \in \mathbb{N}^\Lambda$ by assuming $\mathbf{c}(X) = 0$ for all $X \in \Lambda \setminus \Delta$.

Given a finite set of chemical species Λ, a *reaction* over Λ is a triple $\alpha = \langle \mathbf{r}, \mathbf{p}, k \rangle \in \mathbb{N}^\Lambda \times \mathbb{N}^\Lambda \times \mathbb{R}^+$, specifying the stoichiometry of the reactants and products, respectively, and the *rate constant* k. If not specified, assume that $k = 1$, so that the reaction $\alpha = \langle \mathbf{r}, \mathbf{p}, 1 \rangle$ is also represented by the pair $\langle \mathbf{r}, \mathbf{p} \rangle$. For instance, given $\Lambda = \{A, B, C\}$, the reaction $A + 2B \to A + 3C$ is the pair $\langle (1, 2, 0), (1, 0, 3) \rangle$. A *(finite) chemical reaction network (CRN)* is a pair $\mathcal{C} = (\Lambda, R)$, where Λ is a finite set of chemical *species*, and R is a finite set of reactions over Λ. A *configuration* of a CRN $\mathcal{C} = (\Lambda, R)$ is a vector $\mathbf{c} \in \mathbb{N}^\Lambda$.[4]

Given a configuration $\mathbf{c}$ and reaction $\alpha = \langle \mathbf{r}, \mathbf{p} \rangle$, we say that α is *applicable* to $\mathbf{c}$ if $\mathbf{r} \leq \mathbf{c}$ (i.e., $\mathbf{c}$ contains enough of each of the reactants for the reaction to occur). If α is applicable to $\mathbf{c}$, then write $\alpha(\mathbf{c})$ to denote the configuration $\mathbf{c} + \mathbf{p} - \mathbf{r}$ (i.e., the configuration that results from applying reaction α to $\mathbf{c}$). If $\mathbf{c}' = \alpha(\mathbf{c})$ for some reaction $\alpha \in R$, we write $\mathbf{c} \Longrightarrow^1_{\mathcal{C}} \mathbf{c}'$, or merely $\mathbf{c} \Longrightarrow^1 \mathbf{c}'$ when $\mathcal{C}$ is clear from context. An *execution* (a.k.a., *execution sequence*) $\mathcal{E}$ is a finite or infinite sequence of one or more configurations $\mathcal{E} = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \ldots)$ such that, for all $i \in \{1, \ldots, |\mathcal{E}| - 1\}$, $\mathbf{c}_{i-1} \Longrightarrow^1_{\mathcal{C}} \mathbf{c}_i$.

Question: For each execution sequence, is there a unique sequence of *reactions* that could produce it?

Given a finite execution sequence $\mathcal{E} = (\mathbf{c}, \mathbf{c}_1, \ldots, \mathbf{c}')$, write $\mathbf{c} \Longrightarrow_{\mathcal{C}} \mathbf{c}'$, or merely $\mathbf{c} \Longrightarrow \mathbf{c}'$ when the CRN $\mathcal{C}$ is clear from context. In this case, we say that $\mathbf{c}'$ is *reachable* from $\mathbf{c}$. In other words, $\Longrightarrow_{\mathcal{C}}$ is the reflexive, transitive closure $(\Longrightarrow^1_{\mathcal{C}})^*$ of $\Longrightarrow^1_{\mathcal{C}}$.

---

[2] Equivalently, we view an element $\mathbf{c} \in \mathbb{N}^\Lambda$ as a vector of $|\Lambda|$ nonnegative integers, with each coordinate "labeled" by an element of Λ.

[3] There is a natural correspondence between vectors $\mathbf{c} \in \mathbb{N}^\Lambda$ and *multisets* of elements of Λ, where $\mathbf{c}(X)$ is the multiplicity of element $X \in \Lambda$.

[4] This describes the discrete model in which the "amount" of each chemical species X is a nonnegative integer $\#X$ called its *count*. The model more commonly used in chemistry is the real-valued model in which the "amount" of a species X is a nonnegative real number $[X]$ called its *concentration*. The real-valued model receives much more attention from chemists and mathematicians, but the discrete model is much closer to the sort of models computer scientists are accustomed to, so more work on the computational ability of discrete CRNs has been done. In fact, there is a model of computation called Petri nets that is identical to CRNs (although funnily enough, Carl Petri first devised them as a way to think about chemical reactions, but later applied them to computation in his Ph.D. thesis.). Here's a couple examples of work on computation in the real-valued model: `http://web.cs.ucdavis.edu/~doty/papers/ricccrn.pdf`, `http://dl.acm.org/citation.cfm?id=1480445.1480447`. The typical names for each of these models is "stochastic" for the discrete model and "determinstic" or "mass-action" for the real-valued model, but these terms are misleading for the way we will use the models, so we simply call them "integer"- or "real"-valued, or sometimes "discrete" or "continuous".

2.2 Stable computation of functions

We now formalize our notion of computation of functions by CRNs that "work no matter the order in which reactions happen". The inputs to the function are the initial counts of input species $X_1, \ldots, X_k$, and the outputs are the counts of a single output species Y. The system stabilizes to an output when the counts of the output species can no longer change.

A *chemical reaction computer (CRC)* is a tuple $\mathcal{C} = (\Lambda, R, \Sigma, Y, \mathbf{s})$, where (Λ, R) is a CRN, $\Sigma \subset \Lambda$ is the *set of input species*, $Y \in \Lambda \setminus \Sigma$ is the *output species*, and $\mathbf{s} \in \mathbb{N}^{\Lambda \setminus \Sigma}$ is the *initial context*. If $\mathbf{s} = \mathbf{0}$, we say $\mathcal{C}$ is *leaderless*.[5] A *valid initial configuration* $\mathbf{i} \in \mathbb{N}^{\Lambda}$ obeys, for all $S \in \Lambda \setminus \Sigma$, $\mathbf{i}(S) = \mathbf{s}(S)$, and $\mathbf{i} \neq \mathbf{0}$. Write $\mathbf{i} \upharpoonright \Sigma$ to denote the restriction of $\mathbf{i}$ to Σ.[6] A configuration $\mathbf{o} \in \mathbb{N}^{\Lambda}$ is *output stable* if, for every $\mathbf{c}$ such that $\mathbf{o} \Longrightarrow \mathbf{c}$, $\mathbf{o}(Y) = \mathbf{c}(Y)$. We say that $\mathcal{C}$ *stably computes* a function $f : \mathbb{N}^k \to \mathbb{N}$ if for any valid initial configuration $\mathbf{i} \in \mathbb{N}^{\Sigma}$ and any $\mathbf{c} \in \mathbb{N}^{\Lambda}$, $\mathbf{i} \Longrightarrow \mathbf{c}$ implies $\mathbf{c} \Longrightarrow \mathbf{o}$ such that $\mathbf{o}$ is output stable with $f(\mathbf{i} \upharpoonright \Sigma) = \mathbf{o}(Y)$.[7]

Question: Why did I say this means the CRC is "guaranteed" to give the correct answer?[8]

Normally we won't go into this much detail, but to see how the definition of stable computation works with one of our examples from earlier, the following is a full proof of correctness of the CRC computing maximum that we saw earlier.

Theorem 2.2.1. *The leaderless CRC defined by*

$$X_1 \to Y + Z_1 \tag{1}$$
$$X_2 \to Y + Z_2 \tag{2}$$
$$Z_1 + Z_2 \to K \tag{3}$$
$$Y + K \to \emptyset \tag{4}$$

stably computes the function $f(n_1, n_2) = \max(n_1, n_2)$.

Proof. Let $\mathbf{i} = \{n_1 X_1, n_2 X_2\}$ be the initial configuration. We must show that for every $\mathbf{c} \in \mathbb{N}^{\Lambda}$ such that $\mathbf{i} \Longrightarrow \mathbf{c}$, there is an output stable $\mathbf{o} \in \mathbb{N}^{\Lambda}$ such that $\mathbf{c} \Longrightarrow \mathbf{o}$ and $\mathbf{o}(Y) = \max(n_1, n_2)$. Given such a $\mathbf{c}$, let f_1, f_2, f_3, f_4 denote the number of times each reaction above,

[5] This terminology comes from distributed computing. One can assume without loss of generality that if $\mathbf{s} \neq \mathbf{0}$, then $\mathbf{s}(L) = 1$ for some $L \in \Lambda \setminus \Sigma$ and $\mathbf{s}(X) = 0$ for all $X \in \Lambda \setminus (\Sigma \cup \{L\})$, i.e., $\mathbf{s}$ contains on a single copy of some non-input species. This is because, if some other initial context $\mathbf{s}$ is desired, then this can be easily imitated by starting with only the input and $1L$, and adding a reaction $L \to \mathbf{s}$. Thus, whether or not $\mathbf{s} = \mathbf{0}$ corresponds to asking whether we want to allow a single L or not, and this single L is referred to as a "leader". Typically the advantage of starting with a single L is that L "coordinates" the computation and thus acts like a leader that the rest of the species follow.

[6] By convention $\Sigma = \{X_1, X_2, \ldots, X_k\}$, thus we consider $\mathbf{i} \upharpoonright \Sigma$ equivalently as a vector $\mathbf{i} \in \mathbb{N}^k$.

[7] Note that this condition implies that no incorrect output stable configuration is reachable from $\mathbf{i}$.

[8] The definition merely states that it is always *possible* to give the correct answer. The short answer is that, once we define the kinetic model that tells us what will happen (or rather, what is likely to happen), then we will see that, under certain conditions (the set of reachable configurations is finite), then the above definition of always being able to each a correct stable configuration is actually the *same* as saying that such a configuration is actually reached with probability 1 in the kinetic model.

respectively, occurs to get from $\mathbf{i}$ to $\mathbf{c}$.[9] Then for $i \in \{1,2\}$, $\mathbf{c}(X_i) = n_i - f_i$, $\mathbf{c}(Z_i) = f_i - f_3$, $\mathbf{c}(K) = f_3 - f_4$, and $\mathbf{c}(Y) = f_1 + f_2 - f_4$. So from $\mathbf{c}$ we can execute each reaction $i \in \{1,2\}$ an additional $n_i - f_i$ times, followed by reaction 3 an additional $\min_{i\in\{1,2\}}(\mathbf{c}(Z_i) + n_i - f_i) = \min_{i\in\{1,2\}}(f_i - f_3 + n_i - f_i) = \min(n_1, n_2) - f_3$ times, followed by reaction 4 an additional

$$\begin{aligned} & \min(\mathbf{c}(Y) + n_1 - f_1 + n_2 - f_2, \mathbf{c}(K) + \min(n_1, n_2) - f_3) \\ = \; & \min(f_1 + f_2 - f_4 + n_1 - f_1 + n_2 - f_2, f_3 - f_4 + \min(n_1, n_2) - f_3) \\ = \; & \min(n_1 + n_2, \min(n_1, n_2)) - f_4 \\ = \; & \min(n_1, n_2) - f_4 \end{aligned}$$

times, calling the reached configuration $\mathbf{o}$. Thus from $\mathbf{i}$ to $\mathbf{o}$ reaction 4 executes a total of $f_4 + \min(n_1, n_2) - f_4 = \min(n_1, n_2)$ times, decreasing $\#Y$ each time, while each reaction $i \in \{1,2\}$ executes $f_i + n_i - f_i = n_i$ times, increasing $\#Y$ each time. Thus $\mathbf{o}(Y) = n_1 + n_2 - \min(n_1, n_2) = \max(n_1, n_2)$. By similar algebra to determine how the counts of each species change between $\mathbf{i}$ and $\mathbf{o}$, we see that $\mathbf{o}(X_1) = \mathbf{o}(X_2) = \mathbf{o}(K) = \min(\mathbf{o}(Z_1), \mathbf{o}(Z_2)) = 0$, so no reactions are applicable, so $\mathbf{o}$ is output stable. □

The above proof directly uses the definition of stable computation, which allows for CRCs that have possibly unbounded executions. The max CRC does not, however, and this leads to a somewhat simpler proof.

Alternate proof of Theorem 2.2.1. Since for each $i \in \{1,2\}$, X_i is consumed in reaction i and not produced or consumed elsewhere, reaction i can happen at most n_i times. Z_i is only produced in reaction i, so reaction 3 can happen at most $\min(n_1, n_2)$ times. K is only produced in reaction 3, so reaction 4 can happen at most $\min(n_1, n_2)$ times.

Thus, every reaction sequence is of length at most $n_1 + n_2 + 2\min(n_1, n_2)$, so any reaction sequence of that length must end in a terminal configuration $\mathbf{o}$, which is necessarily output stable (since $\mathbf{o} \Longrightarrow \mathbf{o}'$ implies that $\mathbf{o} = \mathbf{o}'$).

If each reaction $i \in \{1,2\}$ happens fewer than n_i times, the configuration is not terminal because there would be excess X_i. Assuming each reaction $i \in \{1,2\}$ happens exactly n_i times, if reaction 3 happens fewer than $\min(n_1, n_2)$ times, the configuration is not terminal because there would be excess Z_1 and Z_2. Assuming further than reaction 3 happens exactly $\min(n_1, n_2)$ times, if reaction 4 happens fewer than $\min(n_1, n_2)$ times, the configuration is not terminal because there would be excess Y and K.

This implies that every reaction sequence of length less than $\ell = n_1 + n_2 + 2\min(n_1, n_2)$ is not terminal. Since every configuration $\mathbf{c}$ reachable from $\mathbf{i}$ by $\ell' < \ell$ reactions is not terminal, this implies that there is a configuration $\mathbf{o}$ reachable from $\mathbf{c}$ by $\ell - \ell'$ additional reactions, which by the above argument must be output stable and correct. □

[9] If there is more than one reaction sequence leading from $\mathbf{i}$ to $\mathbf{c}$, pick one arbitrarily. **Challenge:** Prove that in this case, although there could be several reaction sequences that show that $\mathbf{i} \Longrightarrow \mathbf{c}$, all of them have the same number of reactions in each. Hint: note that the reaction vectors (the products minus the reactants: $\mathbf{p} - \mathbf{r}$ for a reaction $\alpha = (\mathbf{r}, \mathbf{p})$) are all linearly independent.

Not every CRC that stably computes a function has reactions that necessarily move "closer" to the correct configuration. Consider

$$\begin{aligned} X &\rightleftharpoons A \\ X + A &\to Y \end{aligned}$$

This stably computes $\lfloor n/2 \rfloor$, but unlike the previous examples, it can have arbitrarily long executions.

2.3 Stable decidability of predicates

Let's consider a related computational task: computing *predicates* $\psi : \mathbb{N}^k \to \{0, 1\}$. Of course a predicate is a type of function since we can consider the bits 0 and 1 equivalently as elements of $\mathbb{N}$, but they have a lot of special mathematical structure lacking in arbitrary functions because of the fact that there are only two possible outputs.

Informally, we say a CRN decides a predicate if we have two species that vote "yes" and "no" respectively, and it is "guaranteed" to reach a configuration in which the votes are unanimous and correct.

For example, if we have voting species Y and N, we want to decide the predicates

- n is odd

  Initial configuration: $\{nX, 1N\}$

  $$\begin{aligned} N + X &\to Y \\ Y + X &\to N \end{aligned}$$

- $n_1 \geq n_2$

  Initial configuration: $\{n_1X_1, n_2X_2, 1Y\}$

  $$\begin{aligned} Y + X_2 &\to N \\ N + X_1 &\to Y \end{aligned}$$

- $n_1 = n_2$

  Initial configuration: $\{n_1X_1, n_2X_2, 1Y\}$

  $$\begin{aligned} Y + X_1 &\to N_> \\ Y + X_2 &\to N_< \\ N_> + X_2 &\to Y \\ N_< + X_1 &\to Y \end{aligned}$$

  Intuitively there are now two NO-voting species, one that says "No, because $\#X_1 > \#X_2$", and the other that says "No, because $\#X_1 < \#X_2$"

- "n is odd" leaderless

  Input species named X_1:

$$\begin{aligned} X_1 + X_1 &\to X_0 \\ X_0 + X_1 &\to X_1 \\ X_1 + X_0 &\to X_1 \\ X_0 + X_0 &\to X_0 \end{aligned}$$

  Yes voters: X_1, No voters: X_0

- $n_1 = n_2$ leaderless

$$\begin{aligned} X_1 + X_2 &\to Y \\ Y + N &\to Y \\ X_1 + Y &\to X_1 + N \\ X_2 + Y &\to X_2 + N \end{aligned}$$

  Yes voters: Y, No voters: N, X_1, X_2

- $n_1 \geq n_2$ leaderless

  The following works if we can assume $n_1 \neq n_2$:

$$X_1 + X_2 \to \emptyset$$

  Yes voters: X_1, No voters: X_2

  What if we have to handle any input? Intuitively, below we consider a molecule of L_i to represent a "leader", and a molecule of F_i to represent a "follower", who has seen exactly i more X_1's than X_2's, i.e., it thinks $\#X_1 - \#X_2 = i$.

$$\begin{aligned} X_1 + X_2 &\to L_0 \\ L_0 + X_2 &\to L_{-1} \\ L_{-1} + X_1 &\to L_0 \\ L_0 + L_0 &\to L_0 \\ L_{-1} + L_0 &\to L_{-1} \end{aligned}$$

  Yes voters: X_1, L_0, No voters: X_2, L_{-1}

We formalize the notion of stable predicate computation below.

A *chemical reaction decider* (CRD) is a tuple $\mathcal{D} = (\Lambda, R, \Sigma, \Upsilon, \mathbf{s})$, where (Λ, R) is a CRN, $\Sigma \subseteq \Lambda$ is the *set of input species*, and $\Upsilon \subseteq \Lambda$ is the set of *yes voters*, with species in $\Lambda \setminus \Upsilon$ referred to as *no voters*.

We define a global output partial function $\Phi : \mathbb{N}^\Lambda \dashrightarrow \{0, 1\}$ as follows. $\Phi(\mathbf{c})$ is undefined if either $\mathbf{c} = \mathbf{0}$, or if there exist $S_0 \in \Lambda \setminus \Upsilon$ and $S_1 \in \Upsilon$ such that $\mathbf{c}(S_0) > 0$ and $\mathbf{c}(S_1) > 0$.

Otherwise, either $(\forall S \in \Lambda)(\mathbf{c}(S) > 0 \implies S \in \Upsilon)$ or $(\forall S \in \Lambda)(\mathbf{c}(S) > 0 \implies S \in \Lambda \setminus \Upsilon)$; in the former case, the *output* $\Phi(\mathbf{c})$ of configuration $\mathbf{c}$ is 1, and in the latter case, $\Phi(\mathbf{c}) = 0$.

A configuration $\mathbf{o}$ is *output stable* if $\Phi(\mathbf{o})$ is defined and, for all $\mathbf{c}$ such that $\mathbf{o} \Longrightarrow \mathbf{c}$, $\Phi(\mathbf{c}) = \Phi(\mathbf{o})$. We say a CRD $\mathcal{D}$ *stably decides* the predicate $\psi : \mathbb{N}^\Sigma \to \{0, 1\}$ if, for any valid initial configuration $\mathbf{i} \in \mathbb{N}^\Lambda$, for all configurations $\mathbf{c} \in \mathbb{N}^\Lambda$, $\mathbf{i} \Longrightarrow \mathbf{c}$ implies $\mathbf{c} \Longrightarrow \mathbf{o}$ where $\mathbf{o}$ is output stable and $\Phi(\mathbf{o}) = \psi(\mathbf{i})$. We say that $\mathcal{D}$ *stably decides* a set $A \in \mathbb{N}^k$ if it stably decides its indicator predicate $\psi_A : \mathbb{N}^k\{0, 1\}$ defined by $\psi_A(\mathbf{i}) = 1 \iff \mathbf{i} \in A$.

Note that we use a slightly different formal convention in which *every* species votes.

Question: Do these two formalizations define the same class of predicates?

2.4 Semilinear sets

A set $A \subseteq \mathbb{N}^k$ is *linear* if there are vectors $\mathbf{b}, \mathbf{u}_1, \ldots, \mathbf{u}_p \in \mathbb{N}^k$ such that

$$A = \left\{ \mathbf{b} + \sum_{i=1}^{p} n_i \mathbf{u}_i \;\middle|\; n_1, \ldots, n_p \in \mathbb{N} \right\}.$$

A is *semilinear* if it is a finite union of linear sets.

The following theorem is due to Angluin, Aspnes, and Eisenstat:

Theorem 2.4.1. *A set $A \subseteq \mathbb{N}^k$ is stably decidable by a CRD if and only if it is semilinear.*

We can extend the result to functions. The *graph* of a function $f : \mathbb{N}^k \to \mathbb{N}$ is the set $\mathsf{graph}(f) = \left\{ (\mathbf{x}, y) \in \mathbb{N}^{k+1} \mid f(\mathbf{x}) = y \right\}$. A function is *semilinear* if its graph is semilinear.

Examples of semilinear functions include

- $f(n) = \lfloor n/2 \rfloor$
- $f(n_1, n_2) = x_2$ if $x_1 > x_2$ and 0 otherwise
- $f(n_1, n_2) = \max(n_1, n_2)$
- $f(n) = n$ if n is odd, and $n/2$ otherwise

Examples of *non*-semilinear functions include

- $f(n) = n^2$
- $f(n_1, n_2) = n_1 \cdot n_2$
- $f(n_1, n_2) = \lfloor \sqrt{n} \rfloor$
- $f(n) = 2^n$

Theorem 2.4.2. *A function $f : \mathbb{N}^k \to \mathbb{N}$ is stably computable by a CRC if and only if it is semilinear.*

Semilinear predicates show up repeatedly in simple models of computation. For example,

- They are the predicates decidable by *reversal-bounded counter machines*.
- They are precisely the predicates definable in the first-order theory of arithmetic with only the addition operation (but no multiplication). This is called *Presburger arithmetic*.
- Perhaps the most computationally "nice" way to characterize them is that they are the predicates definable as a finite Boolean combination of *threshold* and *mod* predicates. $\phi : \mathbb{N}^k \to \{0,1\}$ is a *threshold* predicate if there are integers $a_1, \ldots, a_k, b \in \mathbb{Z}$ such that $\phi(n_1, \ldots, n_k) = 1 \iff \sum_{i=1}^k a_i n_i \geq b$ (e.g., "is $n_1 \geq 2n_2$?"). ϕ is a *mod* predicate if there are integers $a_1, \ldots, a_k, b, c \in \mathbb{Z}$ such that $\phi(n_1, \ldots, n_k) = 1 \iff \sum_{i=1}^k a_i n_i \equiv b \mod c$ (e.g., "is $n_1 + n_2$ odd?").

2.5 Kinetic model

The following model of stochastic chemical kinetics is widely used in quantitative biology and other fields dealing with chemical reactions between species present in small counts. It ascribes probabilities to execution sequences, and also defines the time of reactions, allowing us to study the computational complexity of CRN computation.

Intuition. With a reaction $A + B \xrightarrow{k} C$, the reaction happens with a certain probability (based on k) if A and B collide. The time until the next reaction is a function *only* of the current configuration. This means the correct distribution governing the time is *memoryless*; if a certain amount of time has passed without a reaction (hence the configuration has not changed), the distribution governing the time until the next reaction is the same as before the time passed. This can be expressed with conditional probability. Let T be the random variable representing the time until the next reaction, starting at time 0. Let $s, t \in [0, \infty)$. Then

$$\mathsf{Pr}[\underbrace{T > s+t}_{\substack{\text{no rxn after} \\ t \textit{ more} \text{ secs}}} \mid \underbrace{T > s}_{\substack{\text{no rxn in} \\ \text{first } s \text{ secs}}}] = \mathsf{Pr}[\underbrace{T > t}_{\substack{\text{no rxn in} \\ \text{first } t \text{ secs}}}].$$

Once you fix the expected value of such a distribution, it turns out that there is only one such continuous distribution. It's called the *exponential* distribution T_λ with rate $\lambda \in (0, \infty)$.[10]

Expected value: $1/\lambda$

PDF: $\lambda e^{-\lambda t}$

CDF: $1 - e^{-\lambda t} = \mathsf{Pr}[T < t]$

What determines the rate (a.k.a. *propensity*) λ for a reaction $A + B \xrightarrow{k} C$?

[10]The closest discrete analogy is a *geometric* distribution, T given by the number of times a coin must be flipped before seeing heads (H). For all $t, s \in \mathbb{N}$, $\mathsf{Pr}[T > s+t | T > s] = \mathsf{Pr}[T > t]$, i.e., the *additional* number of flips necessary to see H, conditioned on having not seen H in the first s flips, is the same as the number of flips needed to see an H from the start.

- molecular counts of reactants: $\lambda \propto \#A \cdot \#B$

  $\#A \cdot \#B$ counts the number of distinct ways the reaction can happen

- volume v: $\lambda \propto \frac{1}{v}$

  In a larger volume, collisions are less frequent.

- rate constant: $\lambda \propto k$

  Captures lots of stuff (diffusion rates, probability of reacting upon collision)

So in this case $\lambda = \frac{k \cdot \#A \cdot \#B}{v}$.
Other reaction types:

$$\begin{aligned} A \xrightarrow{k} \dots \qquad & \lambda = k \cdot \#A \\ A + A \xrightarrow{k} \dots \qquad & \lambda = \frac{k \cdot \#A \cdot (\#A - 1)}{v} \text{ (factor } \frac{1}{2} \text{ captured in } k) \end{aligned}$$

In the general case of a reaction $\alpha = (\mathbf{r}, \mathbf{p}, k)$ with species $S_1, \dots, S_d$, we have

$$\lambda_\alpha = k \cdot \frac{1}{v^{\|\mathbf{r}\|-1}} \cdot \prod_{i=1}^{d} \frac{\#S_i!}{(\#S_i - \mathbf{r}(S_i))!}$$

We write $\lambda_\alpha(\mathbf{c})$ to mean the rate of α in configuration $\mathbf{c}$ when $\mathbf{c}$ is not clear from context.

The kinetics of a CRN is described by a continuous-time Markov process as follows. The time until the next reaction occurs in configuration $\mathbf{c}$ is an exponential random variable with rate $\lambda(\mathbf{c}) = \sum_{\alpha \in R} \lambda_\alpha(\mathbf{c})$ (note that $\lambda(\mathbf{c}) = 0$ if no reactions are applicable to $\mathbf{c}$). Therefore, the expected time for the next reaction to occur is $\frac{1}{\lambda(\mathbf{c})}$. The probability that a particular reaction $\alpha \in R$ will be the next to occur in configuration $\mathbf{c}$ is $\frac{\lambda_\alpha(\mathbf{c})}{\lambda(\mathbf{c})}$.

The kinetic model is based on the physical assumption that the solution is well-mixed, which is valid if the solution is sufficiently dilute. Thus, we assume the *finite density constraint*, which stipulates that a volume required to execute a CRN must be *at least* proportional to the maximum molecular count obtained during execution. In other words, the total concentration (molecular count per volume) is bounded. This realistically constrains the speed of the computation achievable by CRNs.

Since

1. we are concerned with making computation as fast as possible (hence should occur in the smallest volume obeying the finite density constraint),
2. we often ignore constants by using $O()$ notation (hence it is acceptable if the volume is within a constant of the number of molecules present), and
3. many CRNs we study with initial configuration $\mathbf{i}$ have the property that for all $\mathbf{c}$ reachable from $\mathbf{i}$, $\|\mathbf{c}\| = O(\|\mathbf{i}\|)$,

we will usually assume the volume is *equal* to $\|\mathbf{i}\|$.

2.6 Stable computation and probability 1 correctness

The following theorem shows us that we really can equate the idea of stable computation with "probability 1" correctness, assuming a finite reachable state space.

For any CRN $\mathcal{N} = (\Lambda, R)$ and configuration $\mathbf{c} \in \mathbb{N}^\Lambda$, define $\mathsf{post}_\mathcal{N}(\mathbf{c}) = \{ \mathbf{c}' \mid \mathbf{c} \Longrightarrow \mathbf{c}' \}$ be the set of configurations reachable from $\mathbf{c}$.

Given two configurations $\mathbf{c}, \mathbf{c}'$, write $\mathbf{c} \hookrightarrow_\mathcal{N} \mathbf{c}'$ to denote the event that the kinetic model stated above causes the CRN to eventually reach configuration $\mathbf{c}'$ from $\mathbf{c}$.

Theorem 2.6.1. *Let $\mathcal{N} = (\Lambda, R)$ be a CRN, $\mathbf{i} \in \mathbb{N}^\Lambda$ such that $|\mathsf{post}_\mathcal{N}(\mathbf{i})| < \infty$, and $\mathbf{o} \in \mathbb{N}^\Lambda$ such that for all $\mathbf{c} \in \mathsf{post}_\mathcal{N}(\mathbf{i})$, $\mathbf{o} \in \mathsf{post}_\mathcal{N}(\mathbf{c})$. Then $\mathsf{Pr}[\mathbf{i} \hookrightarrow_\mathcal{N} \mathbf{o}] = 1$.*

Proof. For each $\mathbf{c} \in \mathsf{post}_\mathcal{N}(\mathbf{i})$, let $\mathcal{E}_\mathbf{c} = (\mathbf{c}, \ldots, \mathbf{o})$ be any execution leading from $\mathbf{c}$ to $\mathbf{o}$ (for example, the shortest one); it is well-defined since $\mathbf{o} \in \mathsf{post}_\mathcal{N}(\mathbf{c})$. Let $p_\mathbf{c} = \mathsf{Pr}[\mathcal{E}_\mathbf{c}$ occurs from $\mathbf{c}]$. Let $\epsilon = \min_{\mathbf{c} \in \mathsf{post}_\mathcal{N}(\mathbf{i})} p_\mathbf{c}$. Since $\mathsf{post}_\mathcal{N}(\mathbf{i})$ is finite, ϵ is well-defined and positive. Then for any $\mathbf{c} \in \mathsf{post}_\mathcal{N}(\mathbf{i})$, $\mathsf{Pr}[\mathcal{E}_\mathbf{c}$ does not occur from $\mathbf{c}] \leq 1 - \epsilon$. Since $\mathsf{post}_\mathcal{N}(\mathbf{i})$ is finite, in any infinite execution, some configuration $\mathbf{c}$ must be visited infinitely often, so the probability that $\mathcal{E}_\mathbf{c}$ is *never* followed after any of these visits to $\mathbf{c}$ is at most $\prod_{i=1}^{\infty}(1 - \epsilon) = 0$. □

2.7 Time complexity basics

We've seen how to determine the expected time for particular individual reactions to occur. Let's examine how to compute the expected time for various common reaction *sequences* to complete.

2.7.1 "No communication"

$\#A = n$ initially in volume n:

$$\alpha : A \to B$$

Let T_i be the time until the next reaction, when $\#A = i$. Then, it has rate $\lambda_\alpha = i$, so $\mathsf{E}[T_i] = \frac{1}{i}$.

We need to compute the time until the first reaction, plus the time until the second reaction, etc. In other words, the total time $T = \sum_{i=1}^{n} T_i$. We use a handy tool called *linearity of expectation*, which lets us conclude that

$$\mathsf{E}[T] = \mathsf{E}\left[\sum_{i=1}^{n} T_i\right] = \sum_{i=1}^{n} \mathsf{E}[T_i] = \sum_{i=1}^{n} \frac{1}{i} \approx \ln n.$$

Another way to see intuitively that $\mathsf{E}[T] = \Theta(\log n)$ is to see that, if each X decays at a constant rate, then after 1 second, we expect some constant fraction (say, $\frac{1}{2}$) of X to decay. After another second, we expect the same fraction of the *remaining* X to decay, etc. So $\#X$ gets to zero when we have cut $\#X$ in half enough times to get to less than 1; i.e., after $\log n$ seconds.

2.7.2 "Direct communication"

$\#A = \#B = 1$ in volume n:

$$\alpha : A + B \to C$$

We only need one reaction to occur, with rate $\lambda_\alpha = \frac{1 \cdot 1}{n}$, so expected time n. Note this is much slower than the previous time of $\Theta(\log n)$.

2.7.3 "Pairing off"

$\#A = \#B = n$ in volume n:

$$\alpha : A + B \to C$$

Like the first case, n reactions must occur before the CRN terminates. *Un*like the first case, this takes at *least* expected time $\Omega(n)$, since the *last* reaction to occur, when $\#A = \#B = 1$, is the same as the previous "direct communication" reaction. It turns out that in fact, asymptotically, the last reaction dominates:

When $\#A = \#B = i$, we have $\lambda_\alpha = \frac{i^2}{n}$, so the expected time for the next reaction is $\frac{n}{i^2}$. Thus the expected time for all n reactions to complete is

$$\sum_{i=1}^{n} \frac{n}{i^2} = n \underbrace{\sum_{i=1}^{n} \frac{1}{i^2}}_{< \frac{\pi^2}{6}} = \Theta(n).$$

A similar time analysis shows $A + A \to C$ takes time $\Theta(n)$, since it would be defined by a similar sum with a denominator scaling as i^2, but with half as many terms since $\#A$ goes down by 2 each time the reaction occurs.

2.7.4 "Rumor spreading"/"Communication by epidemic"

$\#A = 1, \#F = n$ in volume n:

$$\alpha : A + F \to A + A$$

When $\#A = i$, we have $\#F = n - i + 1$, so $\lambda_\alpha = \frac{i \cdot (n-i+1)}{n}$, so expected time is $\frac{n}{i \cdot (n-i+1)}$. Thus the total expected time is

$$\begin{aligned}
\sum_{i=1}^{n} \frac{n}{i \cdot (n-i+1)} &= n \left(\sum_{i=1}^{n/2} \frac{1}{i \cdot (n-i+1)} + \sum_{i=n/2+1}^{n} \frac{1}{i \cdot (n-i+1)} \right) \\
&= 2n \sum_{i=1}^{n/2} \frac{1}{i \cdot (n-i+1)} \\
&\leq 2n \sum_{i=1}^{n/2} \frac{1}{i \cdot n/2} = 4 \sum_{i=1}^{n/2} \frac{1}{i} = O(\log n).
\end{aligned}$$

2.7.5 Longest time for every molecule to react

Suppose we want to do a computation in which the exact count of some species A matters, not merely its presence or absence. In other words, if we had one fewer molecule of A, something different should happen. Then the only way a molecule of A can be "counted" is to react. Thus the time for the *last* molecule of A to react is a lower bound on the time necessary to do anything nontrivial. The fastest reaction is either a unimolecular reaction $A \to \ldots$, or a bimolecular reaction $A + B \to \ldots$ with $\#B = n =$ volume to obey finite density. Both have rate equal to $\#A$, so we'll just assume we have the unimolecular case, i.e., the same as the "No communication" case above. In other words, we expect to wait $\Theta(\log n)$ time for the last molecule of A to react.

Thus, we require $\Omega(\log n)$ time to do any nontrivial computation.

2.8 Time complexity of CRCs

We now examine the time complexity of some of the CRCs given earlier. In each case, we assume $n \in \mathbb{Z}^+$ is the sum of the initial counts, and also that it is the volume.

1. **Multiplication by 2.** $X \to 2Y$; As observed above in "no communication", this takes expected time $\Theta(\log n)$ no matter the volume.

2. **Division by 2.** $2X \to Y$; This is the pairing off reaction, which takes expected time $\Theta(n)$ as well.

3. **Addition.** $X_1 \to Y; X_2 \to Y$; Although the inputs have different names, there are still n of them, so this takes $\Theta(\log n)$ expected time by the exact same analysis as $X \to 2Y$.

4. **Minimum.** $X_1 + X_2 \to Y$; In the case where $\#X_1 = n_1 = n_2 = \#X_2$, this is the same as pairing off ($A + B \to C$ above), so it requires $\Theta(n)$ expected time. (In this case, n is a factor of 2 different since it is the sum $n_1 + n_2$, rather than just one of them, but ignoring constant factors we get the same time.)

   What if $\#X_1 \neq \#X_2$? We assume without loss of generality that $n_1 > n_2$. Then the expected time is

$$\begin{aligned}
\sum_{i=0}^{n_2-1} \frac{n}{(n_1-i)(n_2-i)} &= n \sum_{i=0}^{n_2-1} \frac{1}{(n_1-i)(n_2-i)} \\
&< n \sum_{i=0}^{n_2-1} \frac{1}{(n_2-i)^2} \\
&= n \sum_{i=1}^{n_2} \frac{1}{i^2} \\
&= O(n),
\end{aligned}$$

so it is no slower than in the case where $n_1 = n_2$. It is possible for it to be much faster, if the difference between the two counts is large. For example, if $n_1 \geq 2n_2$ (hence $n_1 \geq \frac{2}{3}n$), then we have expected time

$$\begin{aligned}
n\sum_{i=0}^{n_2-1}\frac{1}{(n_1-i)(n_2-i)} &< n\sum_{i=0}^{n_2-1}\frac{1}{(n_1-n_2)(n_2-i)} \\
&< n\sum_{i=0}^{n_2-1}\frac{1}{(n_1/2)(n_2-i)} \\
&< 2\frac{n}{n_1}\sum_{i=0}^{n_2-1}\frac{1}{n_2-i} \\
&\leq 2\frac{n}{\frac{2}{3}n}\sum_{i=0}^{n_2-1}\frac{1}{n_2-i} \\
&= 3\sum_{i=0}^{n_2-1}\frac{1}{n_2-i} \\
&= 3\sum_{i=1}^{n_2}\frac{1}{i} \\
&= O(\log n).
\end{aligned}$$

5. **Subtraction.** $X_1 \to Y; X_2 + Y \to \emptyset$; Now we have our first nontrivial combination of two reactions, nontrivial because the reactions are not independent: the second has a reactant produced by the first. To simplify the analysis, we upper bound the process assuming in the worst case that the second reaction does not start until the first completes. Obviously this can only appear *slower* than the actual process that allows the second reaction to start earlier.

   The first reaction completes in expected time $O(\log n)$. The analysis of $X_1 + X_2 \to Y$ above shows that the second reaction completes in expected time $O(n)$, and this is tight if $n_1 = n_2$ (i.e., if the number of Y's produced by the first reaction is equal to the initial number of X_2's).

   Since that is a worst-case analysis, which assumes the second reaction does not start until the first is complete, one might suspect that it is not tight. However, to see that the analysis is in fact tight, consider the case where $n_1 = n_2$, and recall that the expected time for just the last bimolecular reaction between two count-1 species is $\Omega(n)$. Even if the reaction $X_2 + Y \to \emptyset$ were instantaneously fast on all but the very last time the reaction occurs, observe that the final occurrence of the reaction cannot happen until the last $X_1 \to Y$ reaction occurs. Once it does, we have $\#Y = \#X_1 = 1$, and the expected time for them to react is $\Omega(n)$.

   Thus, when $n_1 = n_2$, this CRC takes expected time $\Theta(n)$.

Similarly to the case of the minimum-computing CRC above, this can be faster (as fast as $O(\log n)$ time) when $|n_1 - n_2|$ is sufficiently large.

6. **Maximum.**

$$\begin{aligned} X_1 &\to Y + Z_1 \\ X_2 &\to Y + Z_2 \\ Z_1 + Z_2 &\to K \\ K + Y &\to \emptyset \end{aligned}$$

Observe that when $n_1 = n_2$, the third reaction requires $\Omega(n)$ time, similarly to the analysis of the minimum-computing CRC above. To see that it takes $O(n)$ time no matter the input, we again simplify the analysis by assuming that each reaction does not start until the previous is complete. In this case, the expected time is

$$O(\log n) + O(\log n) + O(n) + O(n) = O(n).$$

The lesson appears to be that some CRCs are as fast as $O(\log n)$, whereas others are slower, but none of those we analyzed was slower than $O(n)$. In fact, the following are true, although we will not prove them.

Theorem 2.8.1. *Let $\psi : \mathbb{N}^k \to \{0, 1\}$ be any semilinear predicate. Then there is a CRD that stably decides ψ in expected time $O(n)$.*

Theorem 2.8.2. *Let $f : \mathbb{N}^k \to \mathbb{N}$ be any semilinear function. Then there is a CRC that stably computes f in expected time $O(n)$.*

2.9 Allowing a small probability of error

We have seen how CRNs can compute functions and predicates with a "guarantee" of getting the right answer, and saw that various ways of formalizing the idea of "guaranteed" success led to the same definition of stable computation: 1) a correct, stable configuration is reachable from every configuration reachable from the initial one, 2) with a finite reachable space of configurations, a correct stable configuration is reached with probability 1 in the standard discrete kinetic model, and 3) every fair execution reaches a correct stable configuration.

We now relax the idea of guaranteed success and ask, what computations can be done by a CRN that has a small, positive probability of error? We will see that this slight relaxation immensely expands the computational power of CRNs, allowing them to simulate a Turing-universal model of computation, and hence to compute *any* predicate/function computable by any algorithm.

2.9.1 Register machines

There is an extremely simple model of computation that surprisingly is Turing universal, called a *register machine*. Informally, a register machine is a finite-state machine whose unbounded memory consists of a fixed number of *registers*, which are nonnegative integers that the finite-state control can increment, decrement, or test for 0.

Although it is common to describe finite-state controls, such as those for Turing machines, has having named states, a state is in some sense or another equivalent to a line of code in a program. This will be slightly more convenient for notation, as each line will always go to the next line after it, unless a conditional branch occurs. The conditional branch occurs when the machine attempts to decrement a register that is already 0. Every decrement instruction must also specify what line of code to jump to if the register is already 0.

Formally, an *instruction* is one of the following types of objects:

1. an *accept*, denoted `accept`
2. a *reject*, denoted `reject`
3. an *increment*, denoted `inc` r_j, where $j \in \mathbb{Z}^+$
4. a *decrement*, denoted `dec` r_j, k, where $j, k \in \mathbb{Z}^+$

A *register machine* M is a finite sequence $c_1, c_2, \ldots, c_l$ of instructions. The semantic interpretation is that M has a *current line* i (initially 1) and register values that indicate what it will do next. Formally, a *configuration* of M with m registers is a tuple $\mathbf{c} \in \mathbb{N}^{m+1}$, where $\mathbf{c}(0)$ indicates the current line and $\mathbf{c}(1), \ldots, \mathbf{c}(m)$ indicate the values of registers r_1 through r_m, respectively.

Let $i = \mathbf{c}(0)$ be the current line. If $c_i =$ `accept` or $c_i =$ `reject`, then M halts, and we say $\mathbf{c}$ is *accepting* or *rejecting*, respectively. If $c_i =$ `inc` r_j, then M increments register r_j and goes to line $i + 1$. If $c_i =$ `dec` r_j, k, then if register r_j is positive, M decrements register r_j and goes to line $i+1$, otherwise it goes to line k. If either an `inc` r_j or a `dec` r_j, k instruction is the last line, then after executing the instruction (assuming $r_j > 0$), M halts and accepts.

M decides the language $L \subseteq \mathbb{N}$ if, given an initial configuration in which the current line is 1, register 1 has value $n \in \mathbb{N}$, and all other registers have value 0, then M eventually reaches an accepting configuration if $n \in L$, and a rejecting configuration if $n \notin L$. M computes a function $f : \mathbb{N} \to \mathbb{N}$ if it eventually reaches a halting configuration with value $f(n)$ in register r_1.

M computes a predicate/function with domain $\mathbb{N}^k$ if, to represent input $\mathbf{x} \in \mathbb{N}^k$, these are the initial values of registers $r_1 \ldots r_k$ are $\mathbf{x}(1), \ldots, \mathbf{x}(k)$, respectively.

We assume there is a special register $r_{m'}$ that is never incremented, so that `dec` $r_{m'}, k$ will unconditionally jump to line k; we use `goto` k as a shorthand for this instruction.

2.9.2 Examples of register machine computation

Let us see some examples of predicates/functions that can be computed by register machines.

$r_1 + r_2$**:**

1. `dec` $r_2, 4$
2. `inc` r_1
3. `goto` 1
4. `accept`

We frequently want to push the entire value of register r_i into r_j. The following code, denoted by `flush` $r_i \to r_j$, does this:

1. `dec` $r_i, 4$
2. `inc` r_j
3. `goto` 1
4. ...

where line 4 represents the line of code after the `flush` macro.

$\max(r_1 - r_2, 0)$**:**

1. `dec` $r_2, 4$
2. `dec` $r_1, 4$
3. `goto` 1
4. `accept`

$2r_1$**:**

1. `dec` $r_1, 5$
2. `inc` r_2
3. `inc` r_2
4. `goto` 1
5. `flush` $r_2 \to r_1$

$\lfloor r_1/2 \rfloor$**:**

1. `dec` $r_1, 5$
2. `dec` $r_1, 5$
3. `inc` r_2
4. `goto` 2
5. `flush` $r_2 \to r_1$

r_1 **is odd?**

1. `dec` $r_1, 5$
2. `dec` $r_1, 4$
3. `goto` 1
4. `accept`
5. `reject`

$r_1 > r_2$**?**

1. `dec` $r_1, 5$
2. `dec` $r_2, 4$
3. `goto` 1
4. `accept`
5. `reject`

$r_1 \cdot r_2$**:**

1. `dec` $r_1, 5$
2. `flush` $r_2 \to r_3, r_4$
3. `flush` $r_3 \to r_2$
4. `goto` 1
5. `flush` $r_4 \to r_1$

Although this looks a whole lot like the sort of computation we were able to do with stably-computing CRNs, the last example is a hint that something a bit more sophisticated is going on, since provably no CRN can stably compute $f(r_1, r_2) = r_1 \cdot r_2$.

In fact, the next section shows that register machines are powerful enough to do any computation that a Turing machine can do.

2.9.3 Register machines are (inefficiently) Turing universal

We want to show that a register machine M_R can simulate a Turing machine M_T. Suppose M_T has tape alphabet $\{0, 1, \sqcup\}$, and that it never writes $\sqcup$, and that whenever it encounters a $\sqcup$, it writes a 0 over it immediately. It's not difficult to see that such a Turing machine is as powerful as one without that restriction. Furthermore, it means that the tape contents of M_T can be represented as a binary string $x \in \{0, 1\}^*$, giving the symbols from the leftmost tape cell until the first $\sqcup$.

A first attempt might be to observe that the tape contents x can be interpreted as the binary expansion of a natural number n, represented in a register of M_R. Operations on x, such as changing a bit, or adding a new bit to one end, are expressible as arithmetic operations on n, but awkwardly so. Also, although the current state of M_T can be represented in the finite state control of M_R, the final ingredient in M_T's configuration, the tape head position,

is unbounded and requires its own register. Although it may be possible to do a simulation directly on such a representation, it could be length and awkward to describe.

There is a more elegant way. The key idea is to represent x with *two* natural numbers r_1 and r_2, represented in binary as the strings to the left and right of the tape cell, respectively, with the least significant bit *closest* to the tape head. To avoid the ambiguity introduced by leading 0's, we will say that the number r_1 is actually represented by the bit string to the left of the tape cell, with a 1 appended to its left end, and similarly for r_2. For example to represent the tape contents 00110$\underline{1}$0001101, with the tape head on the underlined bit (which we call the *current bit*, we break the string into two parts x_1 and x_2 (assigning the underlined bit to x_1), and reverse x_2 so that its most significant bit is on the left:

$$\begin{aligned} x_1 &= 001101 \\ \mathsf{rev}(x_2) &= 1011000 \end{aligned}$$

We then let y_1 and y_2 be strings obtained by prepending a 1 to each of x_1 and $\mathsf{rev}(x_2)$:

$$\begin{aligned} y_1 &= 1001101 \\ y_2 &= 11011000 \end{aligned}$$

Finally, we let r_1 and r_2 be the positive integers represented respectively by y_1 and y_2, and store these integers in registers r_1 and r_2 of M_R. Now, the standard Turing machine operations can be expressed as the computation of predicates and functions as in Section 2.9.2. Let b represent the current bit.

M_T operation	M_R implementation
test if $b = 0$	test if r_1 is even
flip b	if $b = 0$, then $r_1 := r_1 + 1$, else $r_1 := r_1 - 1$
move tape head left	$r_1 := \lfloor r_1/2 \rfloor$, $r_2 := 2r_2 + b$
move tape head right	$r_2 := \lfloor r_2/2 \rfloor$, $r_1 := 2r_1 + b$

We omit the details of dealing with moving the tape head off the right end.

2.9.4 CRN computation with a large probability of error

We now describe how to simulate a register machine with a CRN. The CRN $\mathcal{C}$ will be non-deterministic in the sense that for some configurations of $\mathcal{C}$, two reactions will be applicable, but the correctness of the simulation depends on one of them happening instead of the other. We then describe how to reduce the probability of the incorrect reaction.

For simplicity we describe only the case that the input is a single integer $n \in \mathbb{N}$.

For a register machine M with l lines of instructions and m registers, to define the simulating CRN $\mathcal{N}$, create molecular species $L_1, \ldots, L_l$ and $R_1, \ldots, R_m$, and to represent input n, the initial configuration of $\mathcal{N}$ is $\{nR_1, 1L_1\}$. The presence of molecule L_i is used to indicate that the current line is i. Since the register machine can only be on one line at a time, there will be exactly one molecule of the form L_i present in the solution at any time.

The count of species R_j represents the current value of register r_j. If M halts and accepts (respectively, rejects), we want $\mathcal{N}$ to produce a Y (resp. N) molecule and enter a terminal configuration.[11]

The following table shows the reactions to simulate an instruction of the register machine, assuming the instruction occurs on line i:

`accept`	$L_i \to Y$
`reject`	$L_i \to N$
`goto` k	$L_i \to L_k$
`inc` r_j	$L_i \to L_{i+1} + R_j$
`dec` r_j, k	$L_i + R_j \to L_{i+1}$ $L_i \to L_k$

The first four reactions are error-free simulations of the corresponding instructions. The final two reactions are an error-prone way to decrement register r_j. If $r_j = 0$, then only the latter reaction is possible, and when it occurs it is a correct simulation of the instruction. However, if $r_j > 0$ (hence there are a positive number R_j molecules in solution), then either reaction is possible. While only the former reaction is correct, the latter reaction could still occur. The semantic effect this has on the register machine being simulated is that, when a decrement `dec` r_j, k is possible because $r_j > 0$, the machine may nondeterministically jump to line k anyway.

We could imagine trying to make the last reaction slower by assigning it a lower rate constant $k_s \ll 1$. In this case, the last two reactions have rates $\lambda_c = \frac{1}{v} \cdot \#L_i \cdot \#R_j \geq \frac{1}{v}$ (correct reaction) and $\lambda_i = k_s \cdot \#L_i = k_s$ (incorrect reaction). The probability of error is

$$\frac{\lambda_i}{\lambda_i + \lambda_c} \leq \frac{k_s}{k_s + 1/v}.$$

Problems with this scheme:

1. **Probability of error increases with longer computations.** If d decrements occur in total, then the best we could conclude from the above analysis is that the probability that *any* of them experience an error is at most $\frac{dk_s}{k_s+1/v}$. Once $d \geq \frac{k_s+1/v}{k_s}$, this gives the trivial bound 1. In fact, a more careful analysis shows that the probability of error in any one of d decrements grows exponentially with d (although making k_s small decreases the constants involved).

   In particular, to ensure small chance of error over the whole computation, we need to know in advance how many decrements will occur and set k_s such that $d \ll \frac{k_s+1/v}{k_2}$. This really means that we need to know in advance how many steps the computation will take. Thus this cannot be called "universal" computation; a universal machine

[11] Note that we are referring to a CRN, not a CRD or CRC. As long as the CRN enters a terminal configuration, it can be thought to compute either a predicate $\psi : \mathbb{N}^k \to \{0, 1\}$ or a function $f : \mathbb{N}^k \to \mathbb{N}$, depending on whether we take the presence of Y/N or the count of Y in the terminal configuration as representing the output.

should be able to simulate *any* algorithm, without knowing in advance anything about the algorithm (including how long it will run).

2. **Adjusting rate constants means designing new chemicals.** The idea of reducing the error probability by adjusting the rate constants requires that we design new chemicals that have different properties. It would be better if, in order to reduce the error probability, we need only adjust initial counts of existing chemicals, rather than design new chemicals.

3. **Reducing error slows down the computation.** Decreasing k_s decreases the error linearly (e.g., cutting it in half cuts the error probability in half), but it also increases the expected time linearly (doubling the time if k_s is cut in half).

4. **Register machines are exponentially slower than Turing machines.**

5. **To store b bits we need $\Omega(2^b)$ molecules.**

Problem 5 is fundamental to the idea that the configuration of a CRN is a constant-length vector of counts. However, problems 1-4 are artifacts of this particular implementation. In fact, all of the problems can be eliminated, giving the following theorem:

Theorem 2.9.1. *For every Turing machine M, there is a CRN $\mathcal{N} = (\Lambda, R)$ such that, for every input $x \in \{0,1\}^*$ to M and every $\epsilon > 0$, there is an initial configuration $\mathbf{x} \in \mathbb{N}^\Lambda$ for $\mathcal{N}$ such that $\mathcal{N}$ simulates $M(x)$ with probability at least $1 - \epsilon$, and the expected time for $\mathcal{N}$ to halt is $O(t \cdot s^5)$, where t and s are respectively the running time and space usage of M on input x.*

Of these, problem 4 is the most difficult to deal with. It requires that we more directly simulate a Turing machine (or at least simulate something that is itself faster than a register machine at simulating a Turing machine). Such a simulation is possible, but known CRNs that do it are complex.

In the next section we describe how to do a relatively simple modification to the register machine construction to alleviate problems 1-3.

2.9.5 Turing-universal CRN computation with a small probability of error

We now construct a CRD $\mathcal{D}$ to simulate M, while reducing the probability of an error each time an error could potentially occur. Besides the species described in Section 2.9.4, we introduce the following new species: "clock" species C_1, C_2, and C_3, and "forward" and "backward" species F and B that control the other clock species.

The clock species will be used to reduce the probability of error each time a decrement is simulated.

The initial state of $\mathcal{D}$ on input $n \in \mathbb{N}$ is $\{nR_1, 1L_1, 1C_1, 1F, n_0B\}$ — i.e., start with register $r_1 = n$, initialize the register machine M at line 1, and start the "clock module" in the first of its three stages. Also start with count n_0 of B and count 1 of F; we will see later

that by choosing n_0 sufficiently large, we can make the probability of error arbitrarily close to 0.

Recall that the only source of error in the CRN simulation is from the decrement instruction `dec` r_j, k when R_j is present, but the jump reaction $L_i \to L_k$ occurs instead of the decrement reaction $L_i + R_j \to L_{i+1}$. This would cause the CRN to erroneously perform a jump when it should instead decrement register r_j. To decrease the probability of this occurring, we can slow down the jump reaction, thus decreasing the probability of it occurring before the decrement reaction when R_j is present.

The following reactions, which we call the "clock module," implement a random walk that is biased in the reverse direction, so that C_3 is present infrequently, the expected frequency controlled by the relative counts of F and B:

$$\begin{aligned} F + C_1 &\to F + C_2, & B + C_2 &\to B + C_1, \\ F + C_2 &\to F + C_3, & B + C_3 &\to B + C_2. \end{aligned}$$

We modify the conditional jump reaction to require a molecule of C_3 as a reactant, which also "resets the clock" (converts C_3 to C_1).:

`dec` r_j, k	$L_i + R_j \to L'_i$
	$L'_i + C_3 \to L_{i+1} + C_1 + B$
	$L_i + C_3 \to L_k + C_1 + B$

Increasing the count of species B decreases the expected time until the reaction $B + C_{s+1} \to B + C_s$ occurs, while leaving the expected time until reaction $F + C_s \to F + C_{s+1}$ unchanged. This has the effect that C_3 is present less frequently, delaying the conditional jump reaction.[12]

We refer to the first reaction implementing `dec` r_j, k as the *decrement* reaction and the third as the *jump* reaction; note that if $\#R_j > 0$ then both reactions are possible so they probabilistically compete (the "correct" reaction in that case is the decrement, and the jump reaction executing instead would constitute a simulation error). An additional B molecule is produced to lower the probability of simulation error on the next decrement instruction. Each simulation of `dec` r_j, k converts a C_3 molecule to C_1, which restarts the clock. The second reaction is present to ensure that this occurs after every successful decrement. An additional B molecule is produced to lower the probability of simulation error on the next decrement instruction. If the decrement reaction occurs, then the second reaction above is the only one possible so it is guaranteed that, on each simulation of a `dec` r_j, k instruction, exactly one of the jump reaction or the second reaction occurs. As we continue to perform decrements, the random walk from C_1 to C_3 acquires a stronger reverse bias due to the increase in $\#B$, so the conditional jump becomes less likely to occur erroneously. The reason we convert C_3 to C_1 is to ensure that the increase in $\#B$ has an immediate effect of lowering the probability of C_3 being present.

The `accept`, `reject`, `goto` , and `inc` commands cannot result in errors for the CRD simulation, so we keep their reactions unchanged from Section 2.9.4.

[12] Intuitively, with an ℓ-stage clock, if there are b molecules of B, the frequency of time that C_ℓ is present is less than $\frac{1}{b^{\ell-1}}$.

For an error to occur, the two reactions $L_i + R \to L_i'$ and $L_i + C_3 \to L_k + A + C_1$ compete. After d decrement instructions, $\#B = n_0 + d$. The frequency of time in which C_3 is present is $\leq \frac{1}{\#B^2}$.[13] Thus if an error is possible (i.e., if at least one R is present), then the probability of error on that step — i.e., that L_i encounters C_3 before R — is $\leq \frac{1}{\#B^2}$. Thus, by the union bound, the probability of an error ever occurring is at most

$$\sum_{b=n_0}^{\infty} \frac{1}{b^2}.$$

By choosing n_0 sufficiently large, we can make this error probability arbitrarily close to 0.

2.10 Stably deciding semilinear sets

This section that any semilinear set can be stably decided by a CRD.

First, we observe that the sets decidable by CRDs are closed under union, intersection, and complement.

Theorem 2.10.1. *Let $\mathcal{C}_1 = (\Lambda_1, R_1, \Sigma, \Upsilon_1, \sigma_1)$ and $\mathcal{C}_2 = (\Lambda_2, R_2, \Sigma, \Upsilon_2, \sigma_2)$ be CRDs stably deciding sets $A_1, A_2 \subseteq \mathbb{N}^\Sigma$, respectively. Then there are CRDs stably deciding the sets $A_1 \cup A_2$, $A_1 \cap A_2$, and $\overline{A_1}$, respectively.*

Proof. Let $\Upsilon' = \Lambda_1 \setminus \Upsilon_1$ to decide $\overline{A_1}$. For the other two operations, add reaction $X \to X_1 + X_2$ for each $X \in \Sigma$, where X_i serves as the "true" input species for $\mathcal{C}_i$. Add 4 species $V_{00}, V_{01}, V_{10}, V_{11}$. To "record" the vote of $\mathcal{C}_1$, for each species S_b in $\mathcal{C}_1$ voting $b \in \{0,1\}$ and each $b' \in \{0,1\}$, add the reaction $S_b + V_{(1-b)b'} \to S_b + V_{bb'}$. Similarly, to record the vote of $\mathcal{C}_2$, for each species S_b in $\mathcal{C}_2$ voting $b \in \{0,1\}$ and each $b' \in \{0,1\}$, add the reaction $S_b + V_{b'(1-b)} \to S_b + V_{b'b}$. For $\cup$, let $\Upsilon = \{V_{01}, V_{10}, V_{11}\}$, and for $\cap$, let $\Upsilon = \{V_{11}\}$. □

The following characterization of semilinear sets (due to Ginsberg and Spanier) is helpful.

Theorem 2.10.2. *A set $A \subseteq \mathbb{N}^d$ is semilinear if and only if it is a finite union, intersection, and complement of sets $X \subseteq \mathbb{N}^d$ that are either*

1. threshold sets*: for some constants $b, a_1, \ldots, a_d \in \mathbb{Z}$,*

$$X = \left\{ \mathbf{x} \in \mathbb{N}^d \;\middle|\; \sum_{i=1}^{d} a_i \mathbf{x}(i) < b \right\},$$

or

2. mod sets*: for some constants $a_1, \ldots, a_d \in \mathbb{Z}$, $b, c \in \mathbb{N}$,*

$$X = \left\{ \mathbf{x} \in \mathbb{N}^d \;\middle|\; \sum_{i=1}^{d} a_i \mathbf{x}(i) \equiv b \mod c \right\}.$$

[13] This is not too hard to prove but requires more probability theory than we will cover in this course.

By Theorems 2.10.2 and 2.10.1, to show that all semilinear sets can be stably decided by a CRD, it suffices to show that CRDs can stably decide any threshold set and any mod set.

Theorem 2.10.3. *Any threshold set can be stably computed by a CRD.*

Proof. Let the set be $A = \left\{ \mathbf{x} \in \mathbb{N}^d \;\middle|\; \sum_{i=1}^d a_i \mathbf{x}(i) < b \right\}$. If $a_i > 0$, add the reaction $X_i \to a_i P$, and if $a_i < 0$, add the reaction $X_i \to (-a_i)N$. Start with initial context $\{1L_0, bN\}$ if $b > 0$ and $\{1L_0, (-b)P\}$ otherwise. The CRD must decide if $\#P < \#N$, which is done by the reactions $L_0 + N \to L_1$ and $L_1 + P \to L_0$. □

Theorem 2.10.4. *Any mod set can be stably computed by a CRD.*

Proof. Let the set be $A = \left\{ \mathbf{x} \in \mathbb{N}^d \;\middle|\; \sum_{i=1}^d a_i \mathbf{x}(i) \equiv b \mod c \right\}$. In addition to input species, we start with a single copy of L_0 and have species $L_1, \ldots, L_{c-1}$. For each input species X_i and $j \in \{0, 1, \ldots, c-1\}$, add the reaction $X_i + L_j \to L_{j+a_i \bmod c}$. Let L_b vote YES and all other L_j species vote NO, and no other species votes. □

2.11 Impossibility of stably deciding "$y = x^2$?"

The full proof that only semilinear sets can be stably decided by CRDs is quite involved. In this section we will show some of the basic ideas by showing that no CRD can stably decide the "squaring" set $S = \{ (x, y) \in \mathbb{N}^2 \mid y = x^2 \}$.

We will crucially use the fact that reachability is "additive": If $\mathbf{c} \Longrightarrow \mathbf{d}$, then for all $\mathbf{e} \in \mathbb{N}^\Lambda$, $\mathbf{c} + \mathbf{e} \Longrightarrow \mathbf{d} + \mathbf{e}$.

Recall the partial order $\leq$ defined on $\mathbb{N}^d$ is $\mathbf{c} \leq \mathbf{d}$ if $\mathbf{c}(i) \leq \mathbf{d}(i)$ for all $i \in \{1, \ldots, d\}$. We say a sequence $\mathbf{c}_0, \mathbf{c}_1, \ldots \in \mathbb{N}^d$ is *nondecreasing* if $\mathbf{c}_i \leq \mathbf{c}_{i+1}$ for all i. Given $A \subseteq \mathbb{N}^d$, we say $\mathbf{y} \in A$ is *minimal* if, for all $\mathbf{x} \in A$, $\mathbf{x} \leq \mathbf{y} \implies \mathbf{x} = \mathbf{y}$. Let $\min(A)$ denote the minimal elements of A.

Observation 2.11.1. *For all* $\mathbf{u} \in A$*, then there is* $\mathbf{m} \in \min(A)$ *such that* $\mathbf{m} \leq \mathbf{u}$.

Proof. If $\mathbf{u} \in \min(A)$ then we are done, so assume not. Since $\mathbf{u}$ is not minimal, there is some $\mathbf{u}' \in A$ such that $\mathbf{u}' < \mathbf{u}$. If $\mathbf{u}' \in \min(A)$ then we are done, otherwise we can repeat this process to find $\mathbf{u}'' < \mathbf{u}'$. But there are only a finite number of elements strictly less than $\mathbf{u}$, so this repeated process must terminate with some minimal element $\mathbf{m} < \mathbf{u}$. □

If $(X, \leq)$ is any partially ordered set and $\mathbf{x} \in X$, let $\nabla(\mathbf{x}) = \{ \mathbf{y} \mid \mathbf{y} \geq \mathbf{x} \}$ be the *upper cone* of $\mathbf{x}$.

The following is known as *Dickson's Lemma*, and we will use it repeatedly:

Lemma 2.11.2. *Every infinite sequence* $\mathbf{x}_0, \mathbf{x}_1, \ldots \in \mathbb{N}^d$ *has an infinite nondecreasing subsequence, and every set* $A \subseteq \mathbb{N}^d$ *has a finite number of minimal elements.*

Proof. We prove the first condition by induction on d.

For the base case $d = 1$, let $n_0, n_1, \ldots \in \mathbb{N}$ be an infinite sequence. If the set $A = \{ n_i \mid i \in \mathbb{N} \}$ is finite, then by the pigeonhole principle, there is $c \in A$ such that $n_i = c$ for infinitely many i, defining a nondecreasing subsequence.

Otherwise, for all $n \in \mathbb{N}$, there exists a smallest $i(n) \in \mathbb{N}$ such that $n_{i(n)} > n$. Define the sequence of indices $i_0 < i_1 < i_2 < \ldots$ recursively as $i_0 = 0$ and, for $j > 0$, $i_j = i(n_{i_{j-1}})$, i.e., n_{i_j} is chosen to be the next element of sequence that exceeds $n_{i_{j-1}}$. This establishes the base case for the first condition.

For the inductive case, assume the inductive hypothesis holds for d dimensions and let $A \subseteq \mathbb{N}^{d+1}$ be infinite. For each $\mathbf{x} \in A$, $(\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(d))$ is a vector in $\mathbb{N}^d$, so by the inductive hypothesis for d dimensions, there is an infinite subsequence $\mathbf{x}_0, \mathbf{x}_1, \ldots$ from A such that, for all $i \in \mathbb{N}$ and all $j \in \{1, \ldots, d\}$, $\mathbf{x}_i(j) \leq \mathbf{x}_{i+1}(j)$. Applying the base case on one dimension, there is an infinite subsequence $\mathbf{y}_0, \mathbf{y}_1, \ldots$ of $\mathbf{x}_0, \mathbf{x}_1, \ldots$ such that, for all $i \in \mathbb{N}$, $\mathbf{y}_i(d+1) \leq \mathbf{y}_{i+1}(d+1)$. Then $\mathbf{y}_i \leq \mathbf{y}_{i+1}$.

To see the second condition, let $A \subseteq \mathbb{N}^d$. Suppose for the sake of contradiction that $\min(A)$ is infinite. By the first condition, there is an infinite nondecreasing sequence $\mathbf{m}_0 \leq \mathbf{m}_1 \leq \ldots \in \min(A)$, and since every element of it is distinct, we have $\mathbf{m}_0 < \mathbf{m}_1$, which contradicts the minimality of $\mathbf{m}_1$. Thus $\min(A)$ is finite. □

For convenience, assume that every species votes. This means that a stable configuration $\mathbf{o}$ with output $\Phi(\mathbf{o})$ is one in which, for every $\mathbf{o}'$ such that $\mathbf{o} \Longrightarrow \mathbf{o}'$, no species voting $\neg\Phi(\mathbf{o})$ is present in $\mathbf{o}'$. Conversely a configuration $\mathbf{c}$ is *unstable* if either $\Phi(\mathbf{c})$ is undefined, or there exists $\mathbf{c}'$ such that $\mathbf{c} \Longrightarrow \mathbf{c}'$ and $\mathbf{c}'$ contains a $\neg\Phi(\mathbf{c})$ voter. Because adding more molecules to $\mathbf{c}$ cannot make the preceding false (because reachability is additive), we have the following observation, which we state as "instability is closed upwards".

Observation 2.11.3. *If* $\mathbf{c}$ *is unstable, then all* $\mathbf{d} \geq \mathbf{c}$ *are unstable.*

The upward closure of instability gives some very regular structure to the set of unstable configurations (and hence to its complement, the set of stable configurations).

Lemma 2.11.4. *For any CRD* $\mathcal{D}$*, let* U *be its set of unstable configurations. Then* $U = \bigcup_{\mathbf{m} \in \min(U)} \nabla(\mathbf{m})$.

(Draw picture of U as a finite union of cones in 2D.)

Proof. By Lemma 2.11.2, $\min(U)$ is finite. To see that $\bigcup_{\mathbf{m} \in \min(U)} \nabla(\mathbf{m}) \subseteq U$, let $\mathbf{u} \in \nabla(\mathbf{m})$ for some $\mathbf{m} \in \min(U)$. Since $\mathbf{u} \geq \mathbf{m} \in U$, Observation 2.11.3 implies that $\mathbf{u} \in U$. To see the reverse containment, let $\mathbf{u} \in U$. Observation 2.11.1 implies that there is some $\mathbf{m} \in \min(U)$ such that $\mathbf{m} \leq \mathbf{u}$, so $\mathbf{u} \in \nabla(\mathbf{m})$, implying $U \subseteq \bigcup_{\mathbf{m} \in \min(U)} \nabla(\mathbf{m})$. □

For any CRD $\mathcal{D}$, let $\tau_{\mathcal{D}} = \max_{\mathbf{m} \in M, S \in \Lambda} \mathbf{m}(S)$, where M is the finite set of minimal elements of the unstable configurations of $\mathcal{D}$ as above.

Lemma 2.11.5. *For any CRD* $\mathcal{D} = (\Lambda, \mathbb{R}, \Sigma, \Upsilon, \mathbf{s})$*, if* $\mathbf{c} \leq \mathbf{d}$ *and* $(\forall S \in \Lambda)\ \mathbf{d}(S) - \mathbf{c}(S) > 0 \implies \mathbf{c}(S) \geq \tau_{\mathcal{D}}$,[14] *and* $\mathbf{c}$ *is stable, then* $\mathbf{d}$ *is stable and* $\Phi(\mathbf{d}) = \Phi(\mathbf{c})$.

Proof. Suppose without loss of generality that $\Phi(\mathbf{c}) = 1$, then $\mathbf{c}(Y) > 0$ for some $Y \in \Upsilon$. Thus $\mathbf{d}(Y) \geq \mathbf{c}(Y) > 0$ since $\mathbf{d} \geq \mathbf{c}$, so if $\mathbf{d}$ is stable, then $\Phi(\mathbf{d}) = 1$. To see that $\mathbf{d}$ is stable, let $S \in \Lambda$ such that $\mathbf{d}(S) > \mathbf{c}(S)$ By hypothesis we have that $\mathbf{c}(S) \geq \tau_{\mathcal{D}}$. By Lemma 2.11.4, for any S such that $\mathbf{c}(S) \geq \tau_{\mathcal{D}}$, if $\mathbf{c} \notin U$, then $\mathbf{c}' \notin U$ for any $\mathbf{c}'$ equal to $\mathbf{c}$ and strictly greater on S. Applying this iteratively to every S such that $\mathbf{d}(S) > \mathbf{c}(S)$ implies that $\mathbf{d} \notin U$, i.e., $\mathbf{d}$ is stable. □

(Draw picture of τ-bounding box to see that increasing $\#S$ beyond τ cannot move us into the set of unstable configurations.)

The following "pumping lemma" is the main technical tool to prove that the squaring set S cannot be stably decided.

Lemma 2.11.6. *Let* $\mathcal{D} = (\Lambda, R, \Sigma, \Upsilon, \mathbf{s})$ *be a CRD stably deciding the infinite set* $A \subseteq \mathbb{N}^k$*. Then there are* $\mathbf{c}, \mathbf{d} \in A$ *with* $\mathbf{c} < \mathbf{d}$ *such that* $\{\ \mathbf{c} + m \cdot (\mathbf{d} - \mathbf{c})\ \mid m \in \mathbb{N}\ \} \subseteq A$.

That is, if we write $\mathbf{d} = \mathbf{c} + \delta$ for $\delta = (\mathbf{d} - \mathbf{c})$, then $\mathcal{C}$ must accept any input obtained by pumping additional δ into $\mathbf{c}$, i.e., it also accepts $\mathbf{c} + m \cdot \delta$ for all $m \in \mathbb{N}$.

Proof. Since A is infinite, by Dickson's Lemma there is an infinite nondecreasing sequence $\mathbf{c}_0, \mathbf{c}_1, \ldots \in A$. (For simplicity we are assuming here that the initial context is $\mathbf{0}$, so that an initial configuration $\mathbf{c}_i$ where the correct output is 1 is also considered an element of $A \subseteq \mathbb{N}^k$.) Let $\delta_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.

Let $(\mathbf{o}_i)_{i=0}^{\infty}$ to be an infinite sequence of stable configurations with $\Phi(\mathbf{o}_i) = 1$, defined inductively as follows. For the base case, since $\mathcal{D}$ stably decides A, there exists stable $\mathbf{o}_0$ such that $\mathbf{c}_0 \Longrightarrow \mathbf{o}_0$ and $\Phi(\mathbf{o}_0) = 1$. For the inductive case, the fact that reachability is additive implies that for all i, $\mathbf{c}_{i+1} = \mathbf{c}_i + \delta_i \Longrightarrow \mathbf{o}_i + \delta_i$. The fact that $\mathcal{D}$ stably decides A implies that $\mathbf{o}_i + \delta_i \Longrightarrow \mathbf{o}_{i+1}$ for some stable $\mathbf{o}_{i+1}$ with $\Phi(\mathbf{o}_{i+1}) = 1$, which defines $\mathbf{o}_{i+1}$.

By Dickson's Lemma there is an infinite nondecreasing subsequence of $(\mathbf{o}_i)_{i=0}^{\infty}$; call it $(\overline{\mathbf{o}}_i)_{i=0}^{\infty}$. Let $\Gamma = \{\ S \in \Lambda\ \mid \lim_{i\to\infty} \overline{\mathbf{o}}_i(S) = \infty\ \}$ be the set of species with unbounded counts in $(\overline{\mathbf{o}}_i)$. Thus, if $S \in \Gamma$, then for large enough i, $\overline{\mathbf{o}}_i(S) > \tau_{\mathcal{D}}$. If $S \notin \Gamma$, then there exists $c_S \in \mathbb{N}$ such that $\lim_{i\to\infty} \overline{\mathbf{o}}_i(S) = c_S$.

Let i be large enough that $\overline{\mathbf{o}}_i(S) = c_S$ if $S \notin \Gamma$ (thus $\overline{\mathbf{o}}_{i+1}(S) = \overline{\mathbf{o}}_i(S)$) and $\overline{\mathbf{o}}_i(S) > \tau_{\mathcal{D}}$ if $S \in \Gamma$. Define $\overline{\delta}_i = \delta_j + \delta_{j+1} + \ldots + \delta_k$ (a sum of δ_j's from the original sequence, before we took an infinite nondecreasing subsequence) such that $\overline{\mathbf{o}}_i + \overline{\delta}_i \Longrightarrow \overline{\mathbf{o}}_{i+1}$.

Since $(\overline{\mathbf{o}}_i)$ is nondecreasing, we can write $\overline{\mathbf{o}}_{i+1} = \overline{\mathbf{o}}_i + \gamma$ for $\gamma \geq \mathbf{0}$, where $\gamma(S) > 0 \implies \overline{\mathbf{o}}_i(S) > \tau_{\mathcal{D}}$ (since we chose i large enough that every unbounded species has count $> \tau_{\mathcal{D}}$) and $\gamma(S) > 0 \implies S \in \Gamma$ (since we chose i large enough that every bounded species has a converged count).

[14] In other words, $\mathbf{d}$ is at least $\mathbf{c}$ and $\mathbf{d}$ is strictly larger than $\mathbf{c}$ only on species on which $\mathbf{c}$ is already "large" ($\geq \tau$).

Thus $\overline{\mathbf{o}}_i + \overline{\delta}_i \Longrightarrow \overline{\mathbf{o}}_i + \gamma$, and applying this same reaction sequence m times gives

$$\overline{\mathbf{o}}_i + m \cdot \overline{\delta}_i \Longrightarrow \overline{\mathbf{o}}_i + m \cdot \gamma.$$

By Lemma 2.11.5, no N molecule can be produced from $\overline{\mathbf{o}}_i + m \cdot \gamma$, since $\overline{\mathbf{o}}_i + m \cdot \gamma$ is larger than $\overline{\mathbf{o}}_i$ only on species with count $> \tau_{\mathcal{D}}$ in $\overline{\mathbf{o}}_i$.

We may not have that $\overline{\mathbf{o}}_i + m \cdot \gamma$ is a stable configuration, but since it cannot produce an N molecule, if $\mathcal{D}$ stably decides the set A, the only other possibility is that a stable configuration $\mathbf{o}$ with $\Phi(\mathbf{o}) = 1$ is reachable, which means $\mathcal{D}$ accepts any input that can reach to $\overline{\mathbf{o}}_i + m \cdot \overline{\delta}_i$.

Recall that $\overline{\delta}_i = \delta_j + \delta_{j+1} + \ldots + \delta_k$ for some $j, k \in \mathbb{N}$. Thus $\overline{\delta}_i = \mathbf{c}_k - \mathbf{c}_j$. Let $\mathbf{c} = \mathbf{c}_j$ and $\mathbf{d} = \mathbf{c}_k$. Then $\{\ \mathbf{c} + m \cdot (\mathbf{d} - \mathbf{c})\ \mid m \in \mathbb{N}\ \} \subseteq A$. □

Finally, we can prove that the squaring set cannot be stably decided.

Theorem 2.11.7. *The squaring set $S = \{\ (x, y) \in \mathbb{N}^2\ \mid y = x^2\ \}$ cannot be stably decided by any CRD.*[15]

Proof. Lemma 2.11.6 says that there exist points $(x, y), (x', y') \in S$ with $x < x'$ or $y < y'$ (but since they are in S, if one is strictly smaller then both are) such that, defining $\delta = (x' - x, y' - y)$, the point $(x, y) + n\delta \in S$ for every $n \in \mathbb{N}$. But the point $(x, y) + 2\delta \notin S$. To see why, suppose for the sake of contradiction that $(x, y) + 2\delta = (2x' - x, 2y' - y) \in S$. Then $(2x' - x)^2 = (2y' - y)$, so

$$\begin{aligned} 0 &= (2(x') - x)^2 - (2y' - y) \\ &= (4(x')^2 - 4xx' + x^2) - (2(x')^2 - x^2) \\ &= 2(x')^2 - 4xx' + 2x^2 \\ &= 2(x' - x)^2, \end{aligned}$$

a contradiction since $x' > x$. □

2.12 Reachability

We define $=$-reachability as the decision problem, given a CRN $\mathcal{N} = (\Lambda, R)$ and two configurations $\mathbf{x}, \mathbf{y} \in \mathbb{N}^\Lambda$, is it the case that $\mathbf{x} \Longrightarrow_{\mathcal{N}} \mathbf{y}$? We define $\geq$-reachability as the decision problem, given a CRN $\mathcal{N} = (\Lambda, R)$ and two configurations $\mathbf{x}, \mathbf{y} \in \mathbb{N}^\Lambda$, is it the case that there exists $\mathbf{y}' \geq \mathbf{y}$ such that $\mathbf{x} \Longrightarrow_{\mathcal{N}} \mathbf{y}'$?

[15] A fast way to see this is true is to observe the graph of $y = x^2$; the point $(x, y) + \delta$ corresponds to drawing a line between two points on the graph and then finding a third point on that line; since the graph is convex, the third point cannot also intersect the graph.

2.12.1 =-reachability and ≥-reachability are hard for EXPSPACE

The following proof is due to David Soloveichik, based on Lipton's original proof, but modified to use CRN notation and simplified somewhat.

Notion of "simulation": Here when we say that a particular CRN simulates a (halting) register machine we mean the following: The register machine has some halting states and we are guaranteed that if we reach the halting state then the computation is correct. Further, we are guaranteed that there is some sequence of reactions that takes us to a halting state. However, it's possible that we get trapped in some CRN configuration from which it's impossible to get to the halting state.

Suppose we are trying to simulate a register machine whose maximum register value is 4. We know how to do everything except for test for 0. But we can have two registers x and x' such that $x + x' = 4$ (this invariant is maintained by the computation) and test for 0 as follows:

$$\begin{aligned} L_1 + X &\to L_2 + X && \text{[goto line 2 if } x > 0\text{]} \\ L_1 + 4X' &\to L_3 + 4X' && \text{[goto line 3 if } x = 0\text{]} \end{aligned}$$

(In fact this satisfies a stronger notion of simulation: there is only one terminal sequence of reactions and it gets the correct answer.)

What if we need a larger value than 4? Suppose each register is bounded by 2^k; then we can use a number of reactions that grows with k.

$$\begin{aligned} X' &\rightleftharpoons X'_k \\ 2X'_i &\rightleftharpoons X'_{i-1} && \text{[for each } i \in \{2, \ldots, k\}\text{]} \\ L_1 + X'_1 &\to L_3 + X'_1 && \text{[goto } L_3 \text{ if } x = 0\text{]} \end{aligned}$$

Note that the last reaction can happen if and only if $\#X' = 2^k$ (since we assume it cannot get larger), so if $x + x' = 2^k$ is maintained as an invariant by the register machine, L_1 becomes L_3 only if $\#X = 0$

But since the space usage of the Turing machine simulated by the register machine is order $\log 2^k = k$, we get only PSPACE-hardness.

For EXPSPACE-hardness we need to be more clever. For any k, we'll built a CRN such that to get from L_1 to L_3, some reaction must fire $3^{2^{k-2}} - 1$ times, and the number of species/reactions of the CRN is only $O(k)$. Then by coupling the consumption of X' to this reaction, we can implement [goto L_3 if $x = 0$] above using only $O(k)$ reactions.

For $c : \mathbb{N} \to \mathbb{N}$, we say that a register machine M is $c(n)$-count-bounded if, for every $n \in \mathbb{N}$, M halts on input n, and M's registers never exceed the value $c(n)$.

Lemma 2.12.1. *Let $k \in \mathbb{N}$. Define $\mathbf{o} = \{1Y\}$ and $\mathbf{i}_n$ by*

- $\mathbf{i}_n(V_i) = 1$ *for each* $i \in \{2, \ldots, k\}$,

- $\mathbf{i}_n(X_2) = 2$,
- $\mathbf{i}_n(X_i) = 3^{2^{i-3}} - 1$ *for each* $i \in \{3, \ldots, k\}$,
- $\mathbf{i}_n(S_k) = 1$

For each $3^{2^{k-3}}$*-count-bounded register machine* M*, there is a CRN* $\mathcal{C} = (\Lambda, R)$ *such that, for each input* $n \in \mathbb{N}$ *to* M*,* $\mathbf{i}_n \Longrightarrow_{\mathcal{C}} \mathbf{o}$ *if and only if* M *accepts* n*.*

Proof. We will use the notation $A \xrightarrow{C} B$ to denote the reaction $A + C \to B + C$, i.e., C is a catalyst.

The idea is that to get from 1 S_k (plus carefully chosen other stuff) to 1 H_k (plus other stuff) necessarily requires calling the base case (subroutine S_1) $W_k = 3^{2^{k-2}} - 1$ times. There is no shorter path that successfully produces H_k.

Let $q_{i-1} = 3^{2^{i-3}} - 1$ for each $i \in \{3, \ldots, k\}$.

The i'th subroutine is "called" by producing an S_i, and it signals that it should "return" by producing an H_i. We maintain the invariant $\#V_i + \#W_i = 1$, so we think of the ith subroutine as being in one of two "modes" V_i or W_i, and they all have mode V_i at the start of the whole computation.

(We'll relax the requirement of starting with q_{i-1} of X_i later.)

Base case: (subroutine S_1)

$$S_1 \to T_1$$
$$T_k \to H_1$$

For $i \in \{1, \ldots, k-1\}$:

$$T_i \xrightarrow{V_{i+1}} T_{i+1}$$
$$T_i + X'_{i+1} \xrightarrow{W_{i+1}} T_{i+1} + X_{i+1}$$

Pseudocode:

```
for i ∈ {2, ..., k} do
    if S_i is in W_i mode then
        decrement X'_i;
        increment X_i;
    end
end
```

Algorithm 1: Subroutine S_1

Recursive case: For $i \in \{2, \ldots, k\}$ (subroutine S_i)

$$\begin{aligned} S_i &\to S_{i-1} \\ W_i + H_{i-1} &\to V_i + H_i \\ V_i + H_{i-1} &\to S_{i-1} + W_i \\ X_i + H_{i-1} &\xrightarrow{V_i} S_{i-1} + X_i' \end{aligned}$$

Pseudocode:

call s_{i-1} // (the return from this subroutine is signaled by presence of h_{i-1});
if *in* W_i *mode* **then**
  switch to V_i mode;
  return;
else `// in` V_i `mode`
  Non-deterministically decide between two options;
  1. Switch to W_i mode;
  2. Decrement X_i, increment X_i';
  Call S_{i-1} and goto if statement above;
end

Algorithm 2: Subroutine S_i

The intended path has S_i make the non-deterministic choice (2) unless all X_i have been converted to X_i', and only then make non-deterministic choice (1) switching to W_i mode.

Claim 1. *To reach H_k from the start state of S_k, S_1 must be called $q_k = 3^{2^{k-2}} - 1$ times (i.e., there is a path with this many calls of s_1, but no path with fewer). Further, if H_k is reached, then at that point the configuration is the same as the start configuration (except for $S_k = 0$ and $H_k = 1$).*

Proof. By induction on i. Clearly true for S_2: S_1 must be called 2 times.

Suppose it is true for S_{k-1}. Observe that every time S_{k-1} is called from S_k in the larger system, S_{k-1} acts as in the base case, except for every time S_1 is called, it converts a molecule of X_k' to X_k if W_k mode is on. Thus, in W_k mode, S_{k-1} can only return to S_k when $\#X_k' \geq q_{k-1}$. This can only happen if previously all of X_k is moved to X_k'. This requires S_{k-1} getting called $q_{k-1}+1$ times. Thus, the total number of times S_{k-1} must be called from S_k is $q_{k-1}+2$. Since each call causes q_{k-1} calls to S_1 by the induction hypothesis, the total number of S_1 calls that S_k causes before H_k is produced is $q_{k-1} \cdot (q_{k-1}+2)$ □

□

email from David S:

Ok, finally, do we really need to start with $\#X_i = q_{i-1}$? First, of all I claim that there is a recursive way to build up this initial condition using a construction similar to above, but this email is already kind of long. But I think you will believe me that there is a way to

generate some amount of X_i such that q_{i-1} is the maximum that can be produced. This can be done similar to the exponentiation example that we know and love [in Cook, Soloveichik, Winfree, Bruck]. Then S_k will only reach H_k if the maximum amount was generated in this initiation phase.

2.12.2 $\geq$-reachability is contained in EXPSPACE

TODO: deal with catalysts

The following is due to Rackoff (1978).

Lemma 2.12.2. *Let $\mathcal{N} = (\Lambda, R)$ be a CRN and let $d = |\Lambda|$. There is a constant c such that, for all $\mathbf{x}, \mathbf{y} \in \mathbb{N}^\Lambda$, if $\mathbf{x} \Longrightarrow_{\mathcal{N}} \mathbf{y}$, then there is $\mathbf{y}' \geq \mathbf{y}$ such that $\mathbf{x} \Longrightarrow_{\mathcal{N}} \mathbf{y}'$ via an execution sequence of length at most $2^{2^{cd}}$.*

Proof. Recall a reaction is a pair $\alpha = (\mathbf{r}, \mathbf{p}) \in R$; for convenience we consider the associated *reaction vector* $\mathbf{v} = \mathbf{p} - \mathbf{r}$, which represents the vector that must be added to a configuration $\mathbf{c}$ to denote the configuration $\mathbf{c} + \mathbf{v}$ resulting from applying α. Let V_R denote the set of reaction vectors of R.

For any $\mathbf{c} \in \mathbb{Z}^d$, let $\mathbf{c} \upharpoonright k$ denote the prefix vector $(\mathbf{c}(1), \mathbf{c}(2), \ldots, \mathbf{c}(k)) \in \mathbb{Z}^k$.

Let $d = |\Lambda|$; we will consider $\mathbb{N}^d$ equivalent to $\mathbb{N}^\Lambda$ by considering some arbitrary order to the species $S_1, S_2, \ldots, S_d \in \Lambda$. Let $k \in \{0, 1, \ldots, d\}$. We say a vector $\mathbf{c} \in \mathbb{Z}^d$ is *k-valid* if $\mathbf{c}(j) \geq 0$ for all $1 \leq j \leq k$. Note that a configuration $\mathbf{c} \in \mathbb{N}^d$ is a d-valid vector. Let $r \in \mathbb{N}$. A vector $\mathbf{c}$ is *k-r-bounded* if $0 \leq \mathbf{c}(j) \leq r$ for all $1 \leq j \leq k$; in other words, the first k species have nonnegative count at most r. If $\mathbf{c}$ is k-valid, then it is k'-valid for all $k' \leq k$, and if $\mathbf{c}$ if k-r-bounded, then it is also k-valid, and it is k'-r'-bounded for $k' < k$ and $r' \geq r$. A sequence $\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_m$ is *k-valid* (resp. *k-r-bounded*) if, for $1 \leq j \leq m$, $\mathbf{c}_j$ is k-valid (resp. k-r-bounded).

We write $\mathbf{c} \widehat{\Longrightarrow}^r_k \mathbf{u}$ if there is a k-r-bounded sequence of configurations $C = (\mathbf{c}_1, \ldots, \mathbf{c}_m)$, where $\mathbf{c}_1 = \mathbf{c}$, such that $\mathbf{c}_m \upharpoonright k \geq \mathbf{u} \upharpoonright k$ and, for all $1 \leq j \leq m$, there is a reaction vector $\mathbf{v} \in V_R$ such that $\mathbf{c}_{j-1} + \mathbf{v} = \mathbf{c}_j$.[16]

In other words, $\mathbf{c} \widehat{\Longrightarrow}^r_k \mathbf{u}$ means that we can reach from $\mathbf{c}$ to a $\mathbf{c}_m$ that is at least $\mathbf{u}$ on the first k components, by reactions that possibly take elements greater than index k negative, as long as all elements in the first k indices remain nonnegative and upper-bounded by r along the whole sequence. We write $\mathbf{c} \widehat{\Longrightarrow}_k \mathbf{u}$ replacing "k-r-bounded" above with "k-valid." In either case we say that $\mathbf{c}_m$ *k-covers* $\mathbf{u}$.

Note in particular that the lemma concerns configurations $\mathbf{x}, \mathbf{y}$ such that $\mathbf{x} \widehat{\Longrightarrow}_d \mathbf{y}$.; we write $\mathbf{x} \widehat{\Longrightarrow} \mathbf{y}$ to denote that $\mathbf{x} \widehat{\Longrightarrow}_d \mathbf{y}$. Also, note that if $k = 0$, since any vector vacuously 0-covers any other vector, then for all $\mathbf{c}, \mathbf{c}'$, $\mathbf{c} \widehat{\Longrightarrow}_0 \mathbf{c}'$ by the trivial sequence $(\mathbf{c})$.

[16] Note that if $\mathbf{v}$ is the reaction vector of reaction α, α may not be applicable at $\mathbf{c}_{j-1}$, and indeed $\mathbf{c}$ may not even be a nonnegative configuration since for $k' > k$ we could have $\mathbf{c}_{j-1}(k') < 0$. So $\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_m$ is not necessarily an execution sequence.

For $\mathbf{c}, \mathbf{u} \in \mathbb{N}^\Lambda$ and $k \in \{1, \ldots, d\}$, if $\mathbf{c} \widehat{\Longrightarrow}_k \mathbf{u}$ let $\ell_k(\mathbf{c}, \mathbf{u})$ denote the shortest k-valid sequence from $\mathbf{c}$ to some $\mathbf{c}_m$ that k-covers $\mathbf{u}$. To prove the lemma, it suffices to show that for some constant c, for all $\mathbf{c}, \mathbf{u}$ such that $\mathbf{c} \widehat{\Longrightarrow} \mathbf{u}$, $\ell_d(\mathbf{c}, \mathbf{u}) \leq 2^{2^{cn}}$.

This follows from the following claim.

Claim 2. *Let $s \in \mathbb{Z}^+$ be the maximum stoichiometric coefficient of any reactant. For $1 \leq k \leq d$, $\ell_k(\mathbf{c}, \mathbf{u}) \leq (s \cdot \ell_{k-1}(\mathbf{c}, \mathbf{u}))^k + \ell_{k-1}(\mathbf{c}, \mathbf{u})$.*

We now prove this claim by induction on k. Let $\mathbf{c}, \mathbf{u}$ such that $\mathbf{c} \widehat{\Longrightarrow} \mathbf{u}$.

For the base case $k = 0$, $\ell_k(\mathbf{c}, \mathbf{u}) = 1$ since $\mathbf{c}$ 0-covers $\mathbf{u}$.

For the inductive case for $k > 0$, since $\mathbf{c} \widehat{\Longrightarrow} \mathbf{u}$, there is a k-valid sequence $C = (\mathbf{c}_1, \ldots, \mathbf{c}_m)$, where $\mathbf{c}_1 = \mathbf{c}$, that k-covers $\mathbf{u}$. Let $r = s \cdot \ell_{k-1}(\mathbf{c}, \mathbf{u})$. We consider two cases:

1. C is k-r-bounded. This case will actually not require the inductive hypothesis. We consider two subcases:

   (a) For all $1 \leq i < i' \leq m$, $\mathbf{c}_i \upharpoonright k \neq \mathbf{c}_{i'} \upharpoonright k$. In other words, the subvector consisting of first k integers never repeat in the sequence. Then since $0 \leq \mathbf{c}_i(j) \leq r$ for all $1 \leq i \leq m$ and $1 \leq j \leq k$, we have $\ell_k(\mathbf{c}, \mathbf{u}) \leq m \leq r^k = (s \cdot \ell_{k-1}(\mathbf{c}, \mathbf{u}))^k$.

   (b) Otherwise, for some $1 \leq i < i' \leq m$, $\mathbf{c}_i \upharpoonright k = \mathbf{c}_{i'} \upharpoonright k$. Recall that $\mathbf{c}_1 = \mathbf{c}$, and $\mathbf{c}_{i+1} = \mathbf{c}_i + \mathbf{v}_i$ for some $\mathbf{v}_i \in V_R$. Then cut out configurations between $\mathbf{c}_i$ and $\mathbf{c}_{i'}$, as well as the reaction vectors. In other words, redefine $\mathbf{c}_{i+1} = \mathbf{c}_i + \mathbf{v}_{i'}$, $\mathbf{c}_{i+2} = \mathbf{c}_{i+1} + \mathbf{v}_{i'+1}$, ... Then this possibly changes coordinates $k+1 \ldots, d$ in $\mathbf{c}_{i+1}, \ldots, \mathbf{c}_m$, but since $\mathbf{c}_i \upharpoonright k = \mathbf{c}_{i'} \upharpoonright k$, coordinates $1, \ldots, k$ remain unaffected in all $\mathbf{c}_{i+1}, \ldots, \mathbf{c}_m$. By repeating this for any pair of vectors that agree on the first k coordinates, we eventually arrive at a sequence satisfying subcase (1a).

2. C is not k-r-bounded. Since it is k-valid, for some $i \in \{1, \ldots, m\}$ and $1 \leq j \leq k$, $\mathbf{c}_i(j) > r$. Consider the smallest p such that $\mathbf{c}_{p+1}(j) > r$, so that $\mathbf{c}_1, \ldots, \mathbf{c}_p$ is a k-r-bounded prefix sequence. Then by subcase (1a), $p \leq r^k$. Let $C_1 = (\mathbf{c}_1, \ldots, \mathbf{c}_p)$ and $C_2 = (\mathbf{c}_{p+1}, \ldots, \mathbf{c}_m)$.

   Assume without loss of generality that $j = k$. Then C_2 is a k-valid sequence from $\mathbf{c}_{p+1}$ that k-covers $\mathbf{u}$. By the induction hypothesis, there is a $(k-1)$-r-bounded sequence $C_2' = (\mathbf{c}_1', \mathbf{c}_2', \mathbf{c}_3', \ldots, \mathbf{c}_t')$ where $\mathbf{c}_1' = \mathbf{c}_{p+1}$, that $(k-1)$-covers $\mathbf{u}$, and $t \leq \ell_{k-1}(\mathbf{c}_{p+1}, \mathbf{u})$.

   Since $\mathbf{c}_{p+1}(k) > r = s \cdot \ell_{k-1}(\mathbf{c}, \mathbf{u}) \geq s \cdot t$, each reaction consumes at most s copies of species k. Thus $\mathbf{c}_{p+1}(k)$ is sufficiently large that it stays positive in all subsequent vectors, i.e., $\mathbf{c}_{p'}(k) > 0$ for all $p' \in \{1, \ldots, t\}$. Therefore the concatenated sequence $C_1 C_2'$ is k-r-bounded, and

   $$\begin{aligned} |C_1 C_2'| &= p + t \\ &\leq r^k + \ell_{k-1}(\mathbf{c}_{p+1}, \mathbf{u}) \\ &= (s \cdot \ell_{k-1}(\mathbf{c}, \mathbf{u}))^k + \ell_{k-1}(\mathbf{c}_{p+1}, \mathbf{u}), \end{aligned}$$

   which proves the claim.

Let $S = s + 1$. To complete the proof, note that

$$\begin{aligned}(s \cdot \ell_{k-1}(\mathbf{c}, \mathbf{u}))^k + \ell_{k-1}(\mathbf{c}, \mathbf{u}) &\leq ((s+1) \cdot \ell_{k-1}(\mathbf{c}, \mathbf{u}))^k \\ &= (s+1)^k \cdot (\ell_{k-1}(\mathbf{c}, \mathbf{u}))^k, \\ &= S^k \cdot (\ell_{k-1}(\mathbf{c}, \mathbf{u}))^k,\end{aligned}$$

Recall that $\ell_0(\mathbf{c}, \mathbf{u}) = 1$. A simple induction (see first few k for the intuition):

$$\begin{aligned}\ell_1 &\leq S^k \\ \ell_2 &\leq S^k \cdot (S^k)^k = S^{k^2+k} \\ \ell_3 &\leq S^k \cdot (S^{k^2+k})^k = S^{k^3+k^2+k} \\ \ell_4 &\leq S^k \cdot (S^{k^3+k^2+k})^k = S^{k^4+k^3+k^2+k} \\ &\dots\end{aligned}$$

shows that

$$\begin{aligned}\ell_d &\leq S^{k^{d-1}+k^{d-2}+\ldots+k} \\ &\leq S^{k^d} \\ &= 2^{k^d \log S} \\ &= 2^{2^{d \log k} \log S} \\ &= 2^{2^{d \log k} 2^{\log \log S}} \\ &= 2^{2^{d \log k + \log \log S}}\end{aligned}$$

Choosing c sufficiently large that $c \cdot d \geq d \log k + \log \log S$ shows that $\ell_d \leq 2^{2^{cd}}$, completing the proof. □

2.12.3 =-reachability is decidable

I don't understand this proof at all.