Chemical reaction networks

- aTAM self-assembly describes **stateless** molecules that collide and **stick together**.
- Chemical reaction network model describes **stateful** molecules that collide and **bounce apart**, but that might **change state** as a result of the collision.
- Allow more general reactions that produce/consume molecules.

```
reactant(s)       \[ R \rightarrow P_1 + P_2 \]       product(s)

monomers          \[ M_1 + M_2 \rightarrow D \]       dimer

catalyst           \[ C + X \rightarrow C + Y \]
```

Traditionally a **descriptive modeling language**...
Let’s instead use it as a **prescriptive programming language**
What behavior is possible for chemistry in principle?

- **formally definable chemical reaction network**
  - what we’ll study

- **actual chemicals**
  - ultimate interest

- **found in biology**
  - inspiration
Computation with chemical reaction networks

• Key ideas setting chemical computation apart from others:
  • cannot control order in which molecules collide
  • can control how they react when they collide

• Related model of distributed computing called *population protocols*
  • originally motivated by mobile wireless sensor networks, e.g., attached to a birds in a flock

[Computation in networks of passively mobile finite-state sensors, Angluin, Aspnes, Diamadi, Fischer, Peralta. PODC 2004]
Example: Chemical caucusing

\[ X + Y \rightarrow U + U \]
\[ X + U \rightarrow X + X \]
\[ Y + U \rightarrow Y + Y \]

opposite opinions cancel
both opinions influence the unopinionated

distributed algorithm for “approximate majority”:
initial majority (X or Y) quickly overtakes whole population
(with high probability)

[Angluin, Aspnes, Eisenstat, A simple population protocol for fast robust approximate majority, DISC 2007]
Does chemistry compute?

[Dodd, Micheelsen, Sneppen, Thon. Theoretical analysis of epigenetic cell memory by nucleosome modification, Cell 2007]


[Cardelli, Morphisms of reaction networks that couple structure to function, BMC Systems Biology 2014]
Why compute with chemistry?

- **Cells**
  - smart drug released only in certain cellular conditions
  - slow: \(\approx 10-100 \text{ nm}\)
  - yes: compatible with “wet environments”?

- **DNA storage**
  - in-place computation replacing expensive read/write lab steps
  - fast: \(\approx 10-100 \text{ nm}\)
  - not easily

- **Bioreactors**
  - chemical controller to optimize yield of metabolically produced biofuels/drugs/etc.

**Why not compute with chemistry?**

- Speed?
  - no: \(\sim 10-100 \text{ nm}\)

- Component size?
  - no: \(\sim 10-100 \text{ nm}\)

**DNA storage**

- **DNA storage**
  - in-place computation replacing expensive read/write lab steps
  - fast: \(\approx 10-100 \text{ nm}\)
  - not easily
Can we compute with chemistry?

“Not every chemical reaction network describes real chemicals!”, i.e. “where’s the compiler?”

Response: [Soloveichik, Seelig, Winfree, *PNAS* 2010] showed how to physically implement any chemical reaction network using *DNA strand displacement*

\[ X_1 + X_2 \rightarrow X_3 \]
DNA strand displacement implementing $A + B \rightarrow C$
Experimental implementations of synthetic chemical reaction networks with DNA


What behavior is possible for chemistry in principle?

formally definable chemical reaction network

≈

actual chemicals

found in biology
Theoretical Computer Science Approach

What computations necessarily take a long time and what can be done quickly? (Computational complexity theory)

What computation is possible and what is not? (Computability theory)

**NP-complete**
- protein folding
- Boolean satisfiability
- Hamiltonian path
- integer factoring

**P**
- DNA sequence alignment
- polynomial factoring
- integer multiplication
- shortest path

- protein folding
- Boolean satisfiability
- Hamiltonian path
- integer factoring

- DNA sequence alignment
- polynomial factoring
- integer multiplication
- shortest path
Chemical Reaction Networks (formal definition)

• finite set of \textit{d species} \( \Lambda = \{ A, B, C, D, \ldots \} \)

• finite set of \textit{reactions}: \textit{e.g.} \[ A+B \xrightarrow{k_1} A+C \]
\[ C \xrightarrow{k_2} A+A \]
\[ C+2B \xrightarrow{k_3} C \]

\( k_1, k_2, k_3 \) are called rate constants; if not specified, assume \( = 1 \).

• \textit{configuration} \( \mathbf{x} \in \mathbb{N}^d \): molecular counts of each species

• reaction is \textit{applicable} to \( \mathbf{x} \) if \( \mathbf{x} \) has enough of each reactant.
What is \textbf{possible}: 

Example reaction sequence (a.k.a. \textit{execution})

\[ \alpha: \quad A + B \rightarrow A + C \]
\[ \beta: \quad C \rightarrow A + A \]

\[ x = (2, 2, 0) \]
\[ \alpha \downarrow \]
\[ (2, 1, 1) \]
\[ \beta \downarrow \]
\[ (4, 1, 0) \]
\[ \alpha \downarrow \]
\[ (4, 0, 1) \]

Formally, an execution is a sequence of configurations \( x_1, x_2, \ldots \) such that each \( x_i \Rightarrow x_{i+1} \) by a single reaction. If initial configuration \( x_1 \) is understood, the sequence of reactions is sometimes called the execution.
Some simple reactions

\[ X \xrightleftharpoons[2]{1} Y \]

Count of \( Y \) never stabilizes

start with \( n \) copies of molecule \( X \)

\[ \frac{n}{3} \]

\( \#Y = \frac{n}{2} \) expected at equilibrium

\[ X \xrightarrow{1} Y \]

Count of \( Y \) stabilizes, but not to a deterministic value based on initial count of \( X \)

\[ X \xrightleftharpoons[2]{=} Y \]

\[ \frac{n}{3} \]

\( \#Y \) stabilizes, with expected value \( \frac{n}{2} \)

Worse yet, both depend crucially on rate constants.
Examples of **stable** \((rate-independent)\) CRN computation
Examples of function computation

**division by 2:** \( f(a) = \lfloor a/2 \rfloor \)

**goal:** end up with \( a/2 \) copies of \( Y \)

**multiplication by 2:** \( f(a) = 2a \)

2A \( \rightarrow \) Y

\( A \rightarrow 2Y \)
Examples of function computation

**Multiplication by 3:** $f(a) = 3a$

$A \rightarrow 3Y$

**Division by 3:** $f(a) = \lfloor a/3 \rfloor$

$3A \rightarrow Y$
Examples of function computation

\[ f(a) = 3a \text{ using (≤ 2)-product reactions} \]

\[
\begin{align*}
A & \rightarrow Y + Y' \\
Y' & \rightarrow 2Y
\end{align*}
\]

\[ f(a) = \lfloor a/3 \rfloor \text{ using bimolecular ((≤ 2)-reactant) reactions, starting in config \{ 1 L_0, a A \} (a.k.a., leader-driven)} \]

\[
\begin{align*}
L_0 + A & \rightarrow L_1 \\
L_1 + A & \rightarrow L_2 \\
L_2 + A & \rightarrow L_0 + Y
\end{align*}
\]

ends with 1 copy of \( L_i \) for \( i = ??? \)
Examples of function computation

\[ f(a) = \lfloor a/3 \rfloor \] using bimolecular (≤ 2-reactant) reactions, starting in config \{a A\} (a.k.a., leaderless)

\[
\begin{align*}
A + A & \rightarrow A_2 \\
A_2 + A & \rightarrow Y \\
A_2 + A_2 & \rightarrow A + Y
\end{align*}
\]

Calling \( A = A_1 \), in general to divide by constant \( c \):

\[
\begin{align*}
A_i + A_j & \rightarrow A_k & \text{if } i+j < c, \text{ where } k = i + j \\
A_i + A_j & \rightarrow A_k + Y & \text{if } i+j > c, \text{ where } k = i + j - c \\
A_i + A_j & \rightarrow Y & \text{if } i+j = c
\end{align*}
\]

i.e., \( A \)'s start with 1 “ball” and pass balls to each other; whenever someone gets \( \geq c \) balls, throw away \( c \) balls and produce a \( Y \)
Examples of function computation

**addition:** \( f(a, b) = a + b \)

\[ A \rightarrow Y \]
\[ B \rightarrow Y \]

**subtraction:** \( f(a, b) = \max(0, a - b) \)

\[ A \rightarrow Y \]
\[ B + Y \rightarrow \emptyset \]
Examples of function computation

**composition:** $f(a,b) = 3a - (b/2)$

- $A \rightarrow 3Y$
- $2B + Y \rightarrow \emptyset$

**minimum:** $f(a,b) = \min(a,b)$

- $A + B \rightarrow Y$

**maximum:** $f(a,b) = \max(a,b) = a + b - \min(a,b)$

- $A \rightarrow Y + A_2$
- $B \rightarrow Y + B_2$
- $A_2 + B_2 \rightarrow K$ (addition)
- $K + Y \rightarrow \emptyset$ (subtraction)

**only linear functions computable?**
Examples of function computation

**constant:** $f(a) = 1$

$A \rightarrow Y$

$2Y \rightarrow Y$

*a.k.a. “leader election”*

**subtract constant:** $f(a) = a-1$

$2A \rightarrow A+Y$
Examples of predicate computation

**Detection:** \( \varphi(a,b) = \text{yes} \iff b > 0 \)

- \( B+A \rightarrow 2B \)

  *A votes no; B votes yes*

**Counting:** \( \varphi(a,b) = \text{yes} \iff b > 1 \)

- \( 2B \rightarrow 2Y \)
- \( Y+B \rightarrow 2Y \)
- \( Y+A \rightarrow 2Y \)

  *A,B vote no; Y votes yes*
Examples of predicate computation

**Majority:** $\varphi(a,b) = \text{yes} \iff a \geq b$

- $A + B \rightarrow A_f + B_f$ (both become “followers” but preserve difference between $A$’s and $B$’s)
- $A + B_f \rightarrow A + A_f$ (leader changes vote of follower)
- $B + A_f \rightarrow B + B_f$ (leader changes vote of follower)
- $A_f + B_f \rightarrow A_f + A_f$ (tiebreaker if no leaders left when $a=b$)


[Mertzios, Nikoletseas, Raptopoulos, Spirakis, *Determining Majority in Networks with Local Interactions and very Small Local Memory*, Distributed Computing 2015]
Examples of predicate computation

**Parity: \( \phi(a) = Y \iff a \text{ is odd} \)**

\( a = A_o \)  
(subscript o/e means ODD/EVEN, and capital A means it is leader)

- \( A_o + A_o \rightarrow A_e + a_e \)
- \( A_e + A_e \rightarrow A_e + a_e \)  
  two leaders XOR their parity,  
  and one becomes follower
- \( A_o + A_e \rightarrow A_o + a_o \)

- \( A_o + a_e \rightarrow A_o + a_o \)  
  leader overwrites
- \( A_e + a_o \rightarrow A_e + a_e \)  
  bit of follower
Formal definition of CRN computation
Modeling choices in formalizing “Computing with chemistry”

• **integer counts** (“stochastic”) or **real** concentrations ("mass-action"/“deterministic")?

• what is the object being “computed”?
  - yes/no decision problem? “#A’s > #B’s?”
  - numerical function? “set #Y = #X/2”

• **guaranteed to get correct answer** or allow **small probability of error**?
  - if Pr[error] = 0, system works *no matter the reaction rates*

• to represent input \(a_1, \ldots, a_k\), what is the initial configuration?
  - only input species \(A_1, \ldots, A_k\) present
  - auxiliary species can be present?

• when is the computation **finished**? when...
  - the output *stops changing*? (convergence)
  - the output *becomes unable to change*? (stabilization)
  - a certain species \(T\) is *first produced*? (termination)

• require **exact numerical answer** or allow an **approximation**?

we’ll start with these choices
Defining **stable** computation

∀ i \rightarrow \exists x \rightarrow \forall o \rightarrow o' (assuming finite set of reachable configurations) equivalent to:
The system will reach a correct stable configuration with probability 1.
Probability-1 correctness can be characterized with only reachability

**Definition:** Let $i$ be a configuration and $Y$ be a set of configurations. Write $\Pr[i \Rightarrow Y]$ to denote the probability of the random event that, starting in configuration $i$, the CRN eventually reaches some configuration $o \in Y$.

**Definition:** For any configuration $i$, let $\text{Reach}(i)$ denote the set of configurations reachable from $i$.

**Theorem:** Let $i$ be a configuration where $\text{Reach}(i)$ is finite, and let $Y$ be a set of configurations. Then

$$\Pr[i \Rightarrow Y] = 1 \iff (\forall x \in \text{Reach}(i)) (\exists o \in \text{Reach}(x)) o \in Y.$$  

This theorem lets us use (often simpler) reachability arguments and avoid discussing probability, while still ensuring probability-1 correctness.

**Proof:**

1. ($\Rightarrow$): Assume $(\exists x \in \text{Reach}(i)) (\forall o \in \text{Reach}(x)) o \notin Y$.
2. Since $\Pr[i \Rightarrow x] > 0$, which prevents ever reaching $Y$, $\Pr[i \Rightarrow Y] < 1$. *(Note this didn’t assume $\text{Reach}(i)$ is finite.)*
3. ($\Leftarrow$): Assume $(\forall x \in \text{Reach}(i)) (\exists o \in \text{Reach}(x)) o \in Y$.
4. For each $x \in \text{Reach}(i)$, let $E_x = (x, ..., o)$ be any finite execution leading from $x$ to some $o \in Y$.
5. Let $k = \max_{x \in \text{Reach}(i)} |E_x|$ be the maximum length of any of these finite executions reaching $o$.
6. Let $p_x = \Pr[E_x \text{ occurs from } x] > 0$.
7. Let $\varepsilon = \min_{x \in \text{Reach}(i)} p_x$. Since $\text{Reach}(i)$ is finite, $\varepsilon > 0$.
8. Then for each $x \in \text{Reach}(i)$, $\Pr[E_x \text{ does not occur from } x \text{ after the next } k \text{ steps}] \leq 1 - \varepsilon < 1$.
9. So, breaking the infinite execution into segments of length $k$, the probability $E_x$ is never followed within $k$ steps after any visit to an $x \in \text{Reach}(i)$ is at most $\prod_{i=1}^{\infty} (1 - \varepsilon) = 0$. QED

To understand this slide, only need the following fact: if a reaction is applicable, then there is a positive probability it occurs.
Deterministic computation ≠ all executions correct

**False statement:** If \( \Pr[i \Rightarrow Y] = 1 \), then every sufficiently long execution starting at \( i \) reaches to some \( c \in Y \).

- Counterexample??
- Suppose \( i = \{A\} \), with reactions
  - \( A \Leftrightarrow B \)
  - \( B \rightarrow C \)
  - Then \( \Pr[\{A\} \Rightarrow \{C\}] = 1 \), but the execution \( \{A\} \Rightarrow \{B\} \Rightarrow \{A\} \Rightarrow \{B\} \Rightarrow \{A\} \Rightarrow \ldots \) avoids it forever.

- **Lesson:** it is too strict to require all sufficiently long executions to reach \( Y \).
Fair executions: Alternative characterization of stable computation

Goal of definition of fair is to make this theorem true:

**Theorem:** Let \( i \) be a configuration, and let \( Y \) be a finite set of configurations. Then

\[
\text{(every fair execution starting at } i \text{ reaches some } o \in Y) \iff \left( \forall x \in \text{Reach}(i) \right) \left( \exists o \in \text{Reach}(x) \right) o \in Y.
\]

**Definition:** An infinite execution \( x_0, x_1, \ldots \) is fair if

\[
\left( \forall o \in \mathbb{N} \right) \left[ \left( \exists \in \mathbb{N} \ x_i \Rightarrow o \right) \Rightarrow \left( \exists k \in \mathbb{N} \ x_k = o \right) \right]
\]

(“there exist infinitely many”)

**Proof:**

1. \((\Rightarrow)\): Suppose every fair execution from \( i \) reaches \( Y \).
2. Any finite execution can be extended to be fair. (why??)
3. Thus \( \left( \forall x \in \text{Reach}(i) \right) \left( \exists o \in \text{Reach}(x) \right) o \in Y \), \( Y \) is reachable from \( x \) by extending with a fair execution.
4. \((\Leftarrow)\): Suppose \( \left( \forall x \in \text{Reach}(i) \right) \left( \exists o \in \text{Reach}(x) \right) o \in Y \).
5. Let \( x_0, x_1, \ldots \) be a fair execution with \( i = x_0 \).
6. Since all \( x_j \in \text{Reach}(i) \), for each \( j \), by hypothesis \( \exists o_j \in \text{Reach}(x_j) \ o_j \in Y \).
7. Since \( Y \) is finite, some \( o \in Y \) is reachable from infinitely many \( x_j \).
8. Since \( x_0, x_1, \ldots \) is fair and \( o \) is infinitely often reachable, there is \( k \) such that \( x_k = o \in Y \), i.e., the fair execution reaches \( Y \). QED
Definition of function computation

• **goal**: compute function $f: \mathbb{N}^k \to \mathbb{N}$, e.g., $f(a,b) = 2a + b/2$

• **input specification**: designate subset $\Sigma \subseteq \Lambda$ as “input” species
  • valid initial configuration $i$: all molecules are from $\Sigma$, e.g., \{100 a, 100 b\}

• **output specification**: designate one species $Y\in\Lambda$ whose count is the *output*

• $o$ is **stable** if, for all $o'$ reachable from $o$, $o(Y) = o'(Y)$

• CRN **stably computes** $f$ if, for all valid initial configurations $i$, and all $x$ reachable from $i$, there is a stable $o$ reachable from $x$ such that $o(Y) = f(i)$.
  • **Recall**: this is equivalent to saying that $i$ reaches to a correct, stable $o$ with probability 1, and equivalent to saying that every fair execution from $i$ reaches to a correct, stable $o$. 
Definition of **predicate** (decision problem) computation

- **goal**: compute predicate $\phi: \mathbb{N}^k \rightarrow \{Y, N\}$, e.g., $\phi(a, b) = Y \iff a > b$

- **input specification**: designate subset $\Sigma \subseteq \Lambda$ as “input” species
  - in valid initial configurations $i$ all molecules are from $\Sigma$, e.g., $\{100 \ A, 55 \ B\}$

- **output specification**: partition species $\Lambda$ into “yes” voters $\Lambda_Y$ and “no” voters $\Lambda_N$
  - $\psi(o) = Y$ (configuration $o$ outputs “yes”) if vote is unanimously yes: $o(S) > 0 \implies S \in \Lambda_Y$
  - $\psi(o) = N$ (configuration $o$ outputs “no”) if vote is unanimously no: $o(S) > 0 \implies S \in \Lambda_N$
  - $\psi(o)$ **undefined** otherwise: $(\exists S \in \Lambda_N, S' \in \Lambda_Y) \ o(S) > 0 \text{ and } o(S') > 0$

- $o$ is **stable** if $\psi(o) = \psi(o')$ (and is defined) for all $o'$ reachable from $o$

- CRN **stably computes** $\phi$ if, for all valid initial configurations $i$, and all $x$ reachable from $i$, there is a stable $o$ reachable from $x$ such that $\psi(o) = \phi(i)$ (*o is correct*).
  - We say the CRN **stably decides** the set $\phi^{-1}(Y) = \text{set of inputs mapping to output } Y$

Feedforward CRNs

A class of CRNs with a simpler definition/proofs for computation
Stable versus terminal

All CRNs we’ve seen so far obey a stronger condition than stabilizing (no reaction can change the output): they reach a configuration where no reaction can happen at all. Further, they all have the property that every sufficiently long execution reaches this configuration.

**Definition**: A configuration is terminal if no reaction is applicable to it.

**Observation**: Every terminal configuration is stable.

**Note**: A configuration can be stable without being terminal. Example?
Feed-forward CRNs

**Definition:** A CRN is feed-forward if reactions can be ordered \( r_1, r_2, \ldots, r_n \) such that, for all \( k < \ell \), no reactant of \( r_k \) appears in \( r_\ell \) (as either reactant or product).

**Example:** The \( \max(A,B) \) CRN:
1. \( A \rightarrow Y + A_2 \) (\( A \) doesn’t appear below)
2. \( B \rightarrow Y + B_2 \) (\( B \) doesn’t appear below)
3. \( A_2 + B_2 \rightarrow K \) (\( A_2, B_2 \) don’t appear below)
4. \( K + Y \rightarrow \emptyset \)

**Lemma:** Suppose in a feed-forward CRN that \( i \Rightarrow c \) by execution \( P \), and \( i \Rightarrow d \) by execution \( Q \). If any reaction occurs less in \( P \) than \( Q \), then \( c \) is not terminal.

**Proof:**
1. Let \( r_k \) be first reaction in feed-forward order such that \( \#(r_k, P) < \#(r_k, Q) \).
2. For ease of exposition, assume \( r_k \) has only one reactant \( A \).
3. \( r_{k+1} \ldots r_n \) do not change \#A, by the definition of feed-forward.
4. \( r_1 \ldots r_{k-1} \) can produce but not consume \( A \). (why??)
5. So only \( r_1 \ldots r_k \) can increase \#A, and only \( r_k \) can decrease \#A.
6. Let \( m = \#(r_k, P) \); Let \( Q' \) be prefix \((i, x_1, \ldots, x_p)\) of \( Q \) such that \( x_p \Rightarrow x_{p+1} \) by the \((m+1)\)'st execution of reaction \( r_k \).
   • \( x_p \) is the config just before the first time that \( r_k \) happens more in \( Q \) than \( P \).
7. Note \( r_1 \ldots r_{k-1} \) occur least as much in \( P \) as in \( Q \). (\( \#(r_i, P) \geq \#(r_i, Q) \) for \( i=1 \) to \( k-1 \))
8. Thus \( r_1 \ldots r_{k-1} \) occur least as much in \( P \) as in \( Q' \). (since \( Q' \) is prefix of \( Q \))
9. Also, \( \#(r_k, P) = \#(r_k, Q') \) by our choice of \( Q' \).
10. So \( A \) is present in \( c \), i.e., \( c(A) > 0 \).
11. Thus \( r_k \) is applicable at \( c \), so \( c \) is not terminal. QED

We often convince ourselves a CRN works by examining just one execution that stabilizes to the correct output, and thinking, “The other executions probably/hopefully end up with the same output.” This reasoning becomes sound with feed-forward CRNs.

**Stable function computation by feed-forward CRNs**

**Lemma (restated):** Suppose that in a feed-forward CRN, \( i \rightarrow c \) by execution \( P \), and \( i \rightarrow d \) by execution \( Q \). If any reaction occurs less in \( P \) than \( Q \), then \( c \) is not terminal.

**Corollary 1:** A feed-forward CRN has at most one terminal configuration reachable from any initial configuration.

**Corollary 2:** If a feed-forward CRN has at least one terminal configuration reachable from any initial configuration (i.e., exactly one), then the CRN stably computes a function.

**Question:** What’s the function?

**Answer:** Letting \( o_i = \) the unique terminal configuration reachable from \( i \), it computes \( f(i) = o_i(Y) \).

**Corollary: The CRN:**
1. \( A \rightarrow Y+A_2 \)
2. \( B \rightarrow Y+B_2 \)
3. \( A_2+B_2 \rightarrow K \)
4. \( K+Y \rightarrow \emptyset \)

stably computes the function \( f(A, B) = \max(A, B) \).

**Proof:**
1. Do the following reactions:
   1. \( \#A \) times rxn 1
   2. \( \#B \) times rxn 2
   3. \( \min(\#A,\#B) \) times rxn 3
   4. \( \min(\#A,\#B) \) times rxn 4
2. This removes all \( A, B, (at \ least \ one \ of \ A_2 \ or \ B_2, \) and \( K \), so this is terminal. By Corollary 2 it stably computes whatever \( Y \) is now, which is...
3. CRN produces \( \#A+\#B \) count of \( Y \) by rxns 1 and 2, and consumes \( \min(\#A,\#B) \) \( Y \)'s by rxn 4, so computes \( \#A+\#B-\min(\#A,\#B) = \max(\#A,\#B) \). QED
In feed-forward CRNs, if there is a terminal configuration, any long enough execution reaches it

**Lemma (restated):** Suppose that in a feed-forward CRN, $i \xrightarrow{c} c$ by execution $P$, and $i \xrightarrow{d} d$ by execution $Q$. If any reaction occurs less in $P$ than $Q$, then $c$ is not terminal.

**Proof:**

1. Let $P$ be the execution leading from $i$ to $c_i$.
2. Any execution $Q$ with $|Q| > |P|$ must have more of some reaction $r$ by the pigeonhole principle.
   1. By the Lemma, $c_i$ is not terminal, a contradiction.
   2. So no execution $Q$ is longer than $P$.
3. Any execution $Q$ with $|Q| = |P|$ must be a permutation of $P$, or else by pigeonhole $Q$ would have more of some reaction, and this would again contradict the terminality of $c_i$.
4. Finally, to rule out that we might have some shorter terminal execution, any execution $Q$ with $|Q| < |P|$ must have some reaction $r$ occurring more in $P$ than $Q$, so by the Lemma, $Q$ cannot reach a terminal configuration. **QED**

**Corollary:** In a feed-forward CRN, if there is a terminal configuration $c_i$ reachable from initial configuration $i$, then $c_i$ is reached by every sufficiently long execution from $i$. Furthermore, all of these executions are permutations of the same number of each reaction type.
Noncompetitive CRNs

**Definition:** A CRN is non-competitive if, for every species $R$, if $R$ is net consumed in some reaction (e.g., $R \rightarrow A$ or $2R \rightarrow R$), then $R$ is not a reactant in any other reaction. ($R$ can be a non-consumed catalyst in any number of reactions, e.g., $R \rightarrow 2R$ or $R+X \rightarrow R+Y$, but then no reaction can net consume it)

**Example:** The max($A,B$) CRN:
1. $A \rightarrow Y+A_2$ ($A$ isn’t a reactant elsewhere)
2. $B \rightarrow Y+B_2$ ($B$ isn’t a reactant elsewhere)
3. $A_2+B_2 \rightarrow K$ ($A_2,B_2$ aren’t reactants elsewhere)
4. $K+Y \rightarrow \emptyset$ ($K, Y$ aren’t reactants elsewhere)

**Lemma:** Suppose in a non-competitive CRN that $i \Rightarrow c$ by execution $P$, and $i \Rightarrow d$ by execution $Q$. If any reaction occurs less in $P$ than $Q$, then $c$ is not terminal.

**Proof:**
1. $Q’ = $ longest prefix $(i, x_1, ..., x_p)$ of $Q$ such that $(r,P) \geq (r,Q)$ for all reactions $r$.  
   - i.e., $x_{p+1}$ is the first time in $Q$ that some reaction exceeds its count in $P$.
2. Let $r$ be the reaction such that $x_p \Rightarrow x_{p+1}$ via $r$.
3. Note $(r,P) = (r,Q’)$ and $(t,P) \geq (t,Q’)$ for all other reactions $t \neq r$.
4. Since CRN is non-competitive, no reactant $A$ of $r$ can be consumed in $t \neq r$.
   1. Some other reactions $t$ might produce $A$.
   2. Since $(t,P) \geq (t,Q’)$, each $t \neq r$ produces at least as much $A$ in $P$ has in $Q’$.
   3. Exactly as much $A$ is consumed by $r$ in $P$ as in $Q’$.
   4. Thus $x_p(A) \leq c(A)$ for all reactants $A$ of $r$.
5. Since $r$ is applicable to $x_p$, it is applicable to $c$.
6. So $c$ is not terminal. QED

We often convince ourselves a CRN works by examining just one execution that stabilizes to the correct output, and thinking, “The other executions probably/hopefully end up with the same output.” This reasoning becomes sound with non-competitive CRNs.

Non-feedforward CRNs

Example of a non-feedforward CRN that stably computes a function?

\[ f(x) = x/2 \]
1. \( X \leftrightarrow Y + A \)
2. \( X + A \rightarrow \emptyset \)

It’s even non-non-competitive!

Not a plot of \( f! \) It's the space of reachable states.
Time complexity of CRNs
What is **probable**: Stochastic kinetic model of chemical reaction networks

Solution volume $v$

<table>
<thead>
<tr>
<th>reaction type</th>
<th>rate / propensity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \xrightarrow{k} \ldots$</td>
<td>$k \cdot #A$</td>
</tr>
<tr>
<td>$A+B \xrightarrow{k} \ldots$</td>
<td>$k \cdot #A \cdot #B / v$</td>
</tr>
</tbody>
</table>

System evolves via a continuous time Markov process:

- $\Pr[\text{next reaction is } j^{th} \text{ one}] = \text{rate of } j^{th} \text{ reaction} / (\text{sum of all reaction rates})$
- Expected time until next reaction is $1 / (\text{sum of all reaction rates})$

[McQuarrie 1967, van Kampen, Gillespie 1977]
Relationship to distributed computing

population protocol = list of transitions such as

\[ x, y \rightarrow x, x \quad a, b \rightarrow c, d \quad a, a \rightarrow a, a \quad (null \ transition) \]

• Repeatedly, two agents (molecules) are picked at random to interact (react) and change state (species).

A population protocol is a chemical reaction network with

• two reactants, two products per reaction
• unit rate constants
• volume = \( n \) = number of agents (never changes)

population protocols \( \subseteq \) chemical reactions, but “most” ideas that apply to one model also apply to the other

Time complexity in population protocols

• pair of agents picked uniformly at random to interact (possibly null interaction)

• parallel time = number of interactions / $n$
  i.e., each agent has $O(1)$ interactions per “unit time”
Speed of computation

How to fairly assess speed?

Like any respectable computer scientist...
1) as a function of input size $n$ (how required time grows with $n$)
2) ignoring constant factors

$n = \text{total molecular count}$
reasonable requirement on volume: $v = O(n)$
i.e., require bounded concentration (finite density constraint)
Full CRN time model (*Gillespie kinetics*)

- **What should** influence total rate $\lambda$ (a.k.a., propensity) of bimolecular reaction $A+B \xleftrightarrow{k} C$?
  - molecular counts of reactants: $\lambda \propto \#A \cdot \#B$ (*the more there are, the faster collisions happen*)
  - volume $v$: $\lambda \propto 1/v$ (*the bigger the volume, the slower collisions happen*)
  - rate constant $k$: $\lambda \propto k$ (*captures things not directly modeled, e.g., diffusion rates, probability that a collision results in a reaction*)
- For this example reaction $A+B \xleftrightarrow{k} C$, combining these we get $\lambda = k \cdot \#A \cdot \#B / v$
- **Other reaction types:**
  - $A+A \xleftrightarrow{k} \ldots \lambda = k \cdot \#A \cdot (\#A-1) / v$ (*symmetric bimolecular reaction*)
    - $\#A \cdot (\#A-1)/2 = \#$ ways to pick two $A$'s to react; factor $\frac{1}{2}$ by convention is put into rate constant $k$
  - $A \xleftrightarrow{k} \ldots \lambda = k \cdot \#A$ (*unimolecular reaction*)
    - no volume term since no collision required
  - $A+B+C \xleftrightarrow{k} \ldots \lambda = k \cdot \#A \cdot \#B \cdot \#C / v^2$ (*trimolecular reaction*)
    - The volume term is squared because (roughly) if we define coordinate system so position of $A$ is always at the origin, then $B$ and $C$ are randomly moving around through $v$ volume “cells”, and it takes $v^2$ expected time for them both to occupy the origin, to cause a three-way $A$-$B$-$C$ collision
  - In general, with $r$ reactants, propensity is number of ways to pick reactants, times $k$, divided by $v^{r-1}$
Discrete versus continuous time

• Time between interactions in CRN model is **exponential** random variable $T$

  - $T = \#$ of coin flips until a heads, with $\Pr[\text{heads}] = p, \ E[\text{time}] = 1/p$
  - (essentially the discrete version of an exponential random variable)
  - in population protocol, heads event = non-null interaction

• Time between interactions in PP model is **geometric** random variable $T$

  - $T = \# \text{of coin flips until a heads, with } \Pr[\text{heads}] = p, \ E[\text{time}] = 1/p$
  - (essentially the discrete version of an exponential random variable)
  - in population protocol, heads event = non-null interaction

• CRN model: time = sum of exponential random variables

  - define volume = $n$, and rate of interaction $a,b \rightarrow ...$ as $\#a \cdot \#b / n$, i.e., expected time

• PP model (time = $\#\text{interactions} / n$)

  - probability that next interaction is $a,b \rightarrow ...$ is $\#a \cdot \#b / (n \choose 2) = 2 \cdot \#a \cdot \#b / (n(n-1))$
  - expected interactions until next $a,b \rightarrow ...$ interaction = $n(n-1) / (2 \cdot \#a \cdot \#b)$, i.e., time $(n-1) / (2 \cdot \#a \cdot \#b)$
  - If we treat interactions symmetrically, (i.e., $a,b \rightarrow c,d$ is an interaction if and only if $b,a \rightarrow d,c$ is an interaction), then we have twice the probability, i.e., expected time becomes $(n-1) / \#a \cdot \#b \sim n / (\#a \cdot \#b)$, essentially the same as the CRN model

  - one possible convention to avoid symmetric interactions is simply define time = $2 \cdot \#\text{interactions} / n$

Can use Chernoff bounds to show it is very likely that they end up taking very close to the same amount of time for any event.
An exponential time difference

$n$ molecules
volume $v = O(n)$

one of these is always
count $\geq n/2$

propensity: $\#A \cdot \#B / v = O(1/n)$

expected time to produce $Y$:

$O(n)$  $O(\log n)$

distributed computing terms:
• epidemic
• rumor/gossip spreading

chemical term:
• autocatalysis
Time complexity analysis (basic motifs)

“direct communication”

\[ A+B \rightarrow Y+W \quad \#A=\#B=1, \ #X=n-2 \]

**population protocol time complexity:**
time until non-null interaction is geometric random variable with success probability
\[ p = \frac{1}{(n \text{ choose } 2)} = \frac{2}{n(n-1)} \]
\[ E[\# \text{ interactions}] = \frac{1}{p} = \frac{n(n-1)}{2} \]
\[ E[\text{time}] = \frac{E[\# \text{ interactions}]}{n} = \frac{n-1}{2} = O(n) \]

**CRN time complexity:**
time until reaction is exponential random variable with
rate \( \lambda = \#A \cdot \#B / n \)
\[ E[\text{time}] = \frac{1}{\lambda} = \frac{1}{n} \]

“epidemic”, “gossip”, “rumor spreading”

\[ B+X \rightarrow B+B \]

**population protocol time complexity:**
when \#B = k, we have \#X = n – k
\[ \Pr[ B+X \rightarrow B+B \text{ is next interaction } | \ #B=k ] = \frac{k(n-k)}{(n \text{ choose } 2)} = \frac{2k(n-k)}{(n(n-1))} \]
expected time until one X converted to B = \( 1/(n \cdot \text{probability}) \)
\[ = \frac{n-1}{2k(n-k)} \]
expected time until all X converted to B =
\[
\approx \frac{1}{2} \left( \sum_{k=1}^{n} \frac{1}{k} + \sum_{k=n}^{1} \frac{1}{k} \right) = \sum_{k=1}^{n} \frac{1}{k} \approx \ln n
\]
Time complexity analysis (*basic motifs*)

“no communication”
? here means “every species” (including A)

\[
\begin{align*}
A +? & \rightarrow B +? \\
\text{#A} & = n, \quad \text{#B} = 0
\end{align*}
\]

*population protocol time complexity:*
When \(\text{#A}=k\), time until non-null interaction is geometric random variable with success probability
\[
p = \frac{k(n-1)}{(n \text{ choose } 2)} = \frac{k}{(2n)}
\]
\[
E[\text{# interactions}] = \frac{1}{p} = \frac{n}{k}
\]
\[
E[\text{time until non-null interaction}] = E[\text{# interactions}] / n = \frac{1}{k}
\]
\[
E[\text{time to convert all A}] = \frac{1}{2} \sum_{k=1}^{n} \frac{1}{k} \approx \frac{1}{2} \ln n
\]

“no communication/ unimolecular decay”
(unimolecular CRN version)

\[
A \rightarrow B \\
\text{#A} = n, \quad \text{#B} = 0
\]

*CRN time complexity:*
When \(\text{#A}=k\), time until next reaction is exponential random variable with rate \(\lambda = k\)
\[
E[\text{time until next reaction}] = \frac{1}{\lambda} = \frac{1}{k}
\]
\[
E[\text{time for all } n \text{ reactions}] = \sum_{k=1}^{n} \frac{1}{k} \approx \ln n
\]
Time complexity analysis (*basic motifs*)

“pairing off”

\[
A + B \rightarrow C
\]

\[\#A=n, \ #B=n, \ \text{total volume} = O(\text{total count}) = n\]

**CRN time complexity:**
When \(#A=#B=k\), next reaction has rate \(\lambda = k^2/n\)

\[
E[\text{time until next reaction}] = 1/\lambda = n/k^2
\]

\[
E[\text{time for all } n \text{ reactions}] = \sum_{k=1}^{n} \frac{n}{k^2}
\]

\[
< n \sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{n \cdot \pi^2}{6} = \Theta(n)
\]

“pairing off” (symmetric version)

\[
A + A \rightarrow C
\]

similar analysis
Time complexity analysis (basic motifs)

“coupon collecting”

$L+A \rightarrow L+B$  
#$L=1$, #$A=n$, #$B=0$, total volume = $O$(total count) = $n$

**CRN time complexity:**
When #$A=k$, next reaction has rate $\lambda = k/n$
E[time until next reaction] = $1/\lambda = n/k$
E[time for all $n$ reactions] = $\sum_{k=1}^{n} \frac{n}{k}$
$< n \sum_{k=1}^{\infty} \frac{1}{k}$
$= \Theta(n \log n)$
Time complexity analysis of stably computing CRNs
Time complexity analysis of stably computing CRNs

**multiplication by 2:** $f(a) = 2a$

$A \rightarrow 2Y$

$O(\log n)$ “unimolecular decay”

**division by 2:** $f(a) = a/2$

$2A \rightarrow Y$

$O(n)$ “pairing off”

**addition:** $f(a, b) = a + b$

$A \rightarrow Y$

$B \rightarrow Y$

$O(\log n)$: same as unimolecular decay, just with two names for decaying species

**minimum:** $f(a, b) = \min(a, b)$

$A + B \rightarrow Y$

$O(n)$: “pairing off”

... worst case if $a = b$

Suppose $a > b$.  

$E[\text{time}] = \sum_{i=0}^{b-1} \frac{n}{(a-i)(b-i)}$

$= n\sum_{i=0}^{b-1} \frac{1}{(a-i)(b-i)}$

$< n\sum_{i=0}^{b-1} \frac{1}{(b-i)^2}$

$= n\sum_{i=1}^{b} \frac{1}{i^2}$

$= O(n)$

So it’s no slower... can it be faster in some cases?

Suppose $a > 2b$, so $a > 2n/3$.

$E[\text{time}] = \sum_{i=0}^{b-1} \frac{1}{(a-i)(b-i)}$

$< n\sum_{i=0}^{b-1} \frac{1}{(a/2-b-i)}$

$= \frac{2n}{a} \sum_{i=0}^{b-1} \frac{1}{(b-i)}$

$= \frac{2n}{a} \sum_{i=1}^{b} \frac{1}{i} \approx \frac{2n}{a} \ln b$

$\leq \frac{2n}{3n} \ln b = 3 \ln b$

Intuitively, there’s always a large $\Omega(n)$ excess of $A$, so “acts like” unimolecular decay of $B$. 
Time complexity analysis of stably computing CRNs

**Subtraction:** $f(a,b) = a - b$

$A \rightarrow Y$

$B + Y \rightarrow \emptyset$

- Unlike addition, this is a nontrivial combination of reactions: rate of second reaction depends how many times first has happened.
- To simplify, we assume second reaction cannot happen until first has finished.
  - This simpler process "stochastically dominates" the real process: it takes even longer than the real process, so suffices to show a time upper bound.

$$E[\text{time}] = E[\text{time for first to finish}] + E[\text{time for second to finish}]$$

$E[\text{time for first to finish}] = O(\log n)$ (unimolecular decay)

$E[\text{time for second to finish}] = O(n)$ in worst case: similar to minimum, worst case when $a=b$, but $O(\log n)$ time if $|a-b| = \Omega(n)$.

$$E[\text{time}] = O(\log n) + O(n) = O(n)$$
Time complexity analysis of stably computing CRNs

**maximum**: \( f(a, b) = \max(a, b) \)

1. \( A \rightarrow Y + A_2 \)
2. \( B \rightarrow Y + B_2 \)
3. \( A_2 + B_2 \rightarrow K \)
4. \( K + Y \rightarrow \emptyset \)

- Assume reaction 3 waits for reactions 1 and 2 before starting, and reaction 4 waits for reaction 3.
- \( E[\text{time for 1 and 2}] = O(\log n) \)
- \( E[\text{time for 3}] = O(n) \)
- \( E[\text{time for 4}] = O(n) \)
- So \( E[\text{time}] = O(\log n) + O(n) + O(n) = O(n) \)
Possibilities of stable computation

What can be stably computed?
Summary: Possibilities and limits of stable computation

Predicates

• φ is stably computable if and only if φ is semilinear.

• semilinear = Boolean combination of threshold and mod predicates: take weighted sum \( s = w_1 \cdot a_1 + \ldots + w_d \cdot a_d \) of inputs and ask if
  - \( s > t \)? (threshold)
  - \( s \equiv c \mod m \)? (mod)

<table>
<thead>
<tr>
<th>a&gt;b?</th>
<th>a=b?</th>
<th>a is odd?</th>
<th>a&gt;0?</th>
<th>a&gt;1?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOT</th>
<th>a=b^2?</th>
<th>a is a power of 2?</th>
<th>a is prime?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Functions

• \( f \) is stably computable if and only if
  - \( \text{graph}(f) = \{ (a,y) \mid f(a)=y \} \) is semilinear.
  - piecewise affine, with semilinear predicate to determine which piece.

\[
\begin{align*}
  a+b & \quad a-b & \quad 2a & \quad a/2 & \quad \min(a,b) & \quad a+1 & \quad a-1 \\
  f(a) = 2a-b/3 & \quad \text{if } a+b \text{ is odd, else } f(a) = a/4+5b \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>NOT</th>
<th>( a^2 )</th>
<th>( 2^a )</th>
<th>( 2a ) if a is prime, else 3a</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All semilinear predicates/functions are known to be computable in \( O(n) \) time.

[Angluin, Aspnes, Diamadi, Fischer, Peralta, Computation in networks of passively mobile finite-state sensors, PODC 2004]
[Angluin, Aspnes, Eisenstat, Stably computable predicates are semilinear, PODC 2006]
[Chen, Doty, Soloveichik, Deterministic function computation with chemical reaction networks, DNA 2012]
[Doty, Hajiaghayi, Leaderless deterministic chemical reaction networks, DNA 2013]
Linear sets

Definition: A set $X \subseteq \mathbb{N}^d$ is linear if there are vectors $b, u_1, \ldots, u_p \in \mathbb{N}^d$ such that

$$X = \{ b + n_1 \cdot u_1 + \ldots + n_p \cdot u_p \mid n_1, \ldots, n_p \in \mathbb{N} \}$$

Example in dimension $d=2$:

- $b = (2,1)$
- $u_1 = (4,1)$
- $u_2 = (2,2)$

multi-dimensional generalization of eventually periodic
Semilinear sets

**Definition:** A set $X \subseteq \mathbb{N}^d$ is semilinear if it is a finite union of linear sets.
Equivalent definitions of semilinear

**Definition 1**: \( X \subseteq \mathbb{N}^d \) is **semilinear** if it is Boolean combination (through finite unions, intersections, and complements) of threshold and mod sets.

**Definition 1a**: \( X \subseteq \mathbb{N}^d \) is a **threshold** set if there are integers \( t \) and \( w_1 \ldots w_k \) such that \( X = \{ (x_1, \ldots, x_d) \in \mathbb{N}^d \mid w_1 \cdot x_1 + \ldots + w_d \cdot x_d > t \} \)

Examples:
- is \( x_1 > x_2 \)?
- is \( x_1 - 3x_2 > x_2 + 5 \)?

Example semilinear set:
- is \( x_1 > x_2 \) and \( x_1 + x_2 \) is odd?

**Definition 1b**: \( X \subseteq \mathbb{N}^d \) is a **mod** set if there are integers \( c, m \) and \( w_1 \ldots w_k \) such that \( X = \{ (x_1, \ldots, x_d) \in \mathbb{N}^d \mid w_1 \cdot x_1 + \ldots + w_d \cdot x_d \equiv c \mod m \} \)

Examples:
- is \( x \) odd?
- is \( x \) 2 more than a multiple of 3? = \{2, 5, 8, 11, 14, \ldots\}
- is \( x_1 - 3x_2 \) odd?

Example semilinear set:
- is \( x_1 + x_2 \) is not a multiple of 3?
Equivalent definitions of semilinear

**Definition 3**: $X \subseteq \mathbb{N}^d$ is semilinear if it is definable in the first-order theory of Presburger arithmetic. *(original definition, hardest to understand; we won’t use it.)*

**Other places semilinear sets show up in computer science:**
- Sets decidable by *reversal-bounded counter machines*.
- In 2D, they are conjectured to be the sets weakly self-assembled by temperature $\tau=1$ tile systems.
Theorem 1: A set $X \subseteq \mathbb{N}^d$ is stably decided by some CRN if and only if it is semilinear.

Full proof is too complex to do in this course. But we’ll show:
1. All semilinear sets can be stably decided.
2. The non-semilinear “squaring” set $X = \{ (a,y) \in \mathbb{N}^2 \mid a^2 = y \}$ cannot be stably decided.

Definition: A function $f: \mathbb{N}^d \to \mathbb{N}$ is semilinear if $\text{graph}(f) = \{ (a,y) \mid f(a) = y \}$ is a semilinear set.

Example of function graph: The squaring set $X$ to the right is the graph of the function $f(a) = a^2$.

Theorem 2: A function $f: \mathbb{N}^d \to \mathbb{N}$ is stably computed by some CRN if and only if it is semilinear.
Possibilities of stable computation

All semilinear functions/predicates can be stably computed by CRNs
Stably decidable sets are closed under Boolean operations

**Theorem:** If sets $X_1, X_2 \subseteq \mathbb{N}^d$ are stably decided by some CRN, then so are $X_1 \cup X_2$, $X_1 \cap X_2$, and $\overline{X_1}$.

**Proof:**

1. To stably decide $\overline{X_1}$, swap the yes and no voters.
2. For $\cup$ and $\cap$, let $C_1$ and $C_2$ stably decide $X_1$ and $X_2$.
3. Add the reaction $A \rightarrow A_1 + A_2$ for each input species $A$, and let $A_i$ be the input species for $C_i$.
4. Add four new species $V_{NN}$, $V_{NY}$, $V_{YN}$, and $V_{YY}$.
5. To "record" the votes of $C_1$ and $C_2$:
   1. If $S_b$ votes $b \in \{N,Y\}$ in $C_1$, add reaction $S_b + V_{\overline{b}} \rightarrow S_b + V_b$ (i.e., $S_b$ changes the first vote of $V$)
   2. If $T_b$ votes $b \in \{N,Y\}$ in $C_2$, add reaction $T_b + V_{\overline{b}} \rightarrow T_b + V_b$ (i.e., $T_b$ changes the second vote of $V$)
6. To stably decide $X_1 \cup X_2$, let yes voters be $V_{NN}$, $V_{YN}$, $V_{YY}$
7. To stably decide $X_1 \cap X_2$, let yes voter be $V_{YY}$

For this proof, we assume that the voting species can be a strict subset of all species.

What if all species are required to vote??
Mod and threshold sets are stably decidable

**Theorem:** Every mod set
\[ M = \{ (x_1, \ldots, x_d) \mid w_1 \cdot x_1 + \ldots + w_d \cdot x_d \equiv c \mod m \} \]
is stably decidable by a CRN.

**Proof:**
1. Start with 1 \( L_0 \) leader.
   The leader will “count the (weighted) input mod m.”
2. For each \( 1 \leq i \leq d \) and \( 0 \leq j < m \), add the reaction
   \[ X_i + L_j \rightarrow L_{j+w_i \mod m} \]
3. Let \( L_c \) vote yes and all others vote no.

**Corollary** (since stably decidable sets are closed under Boolean combinations): Every semilinear set is stably decided by some CRN.

Also true for leaderless CRNs.

[Computation in networks of passively mobile finite-state sensors, Angluin, Aspnes, Diamadi, Fischer, Peralta. PODC 2004]

**Theorem:** Every threshold set
\[ T = \{ (x_1, \ldots, x_d) \mid w_1 \cdot x_1 + \ldots + w_d \cdot x_d > t \} \]
is stably decidable by a CRN.

**Proof:**
1. If \( w_i > 0 \), add reaction \( X_i \rightarrow w_i P \)
2. If \( w_i < 0 \), add reaction \( X_i \rightarrow (-w_i) N \)
3. Need to decide if \((#P \text{ produced}) > (#N \text{ produced}) + t\)
4. Start with 1 \( L_N \) leader and
   1. \( t N \) if \( t > 0 \).
   2. \( (-t) P \) if \( t < 0 \).
5. Now need to decide if \#P > \#N (including those present initially)
6. Add reactions
   1. \( L_Y + N \rightarrow L_N \)
   2. \( L_N + P \rightarrow L_Y \)
Semilinear functions are stably computable

**Lemma**: If \( f: \mathbb{N}^d \rightarrow \mathbb{N} \) is a semilinear function, then it is **piecewise affine**: a finite union of partial affine functions \( g_i: \mathbb{N}^d \rightarrow \mathbb{N} \).

Each \( g_i \) is **affine** (linear with constant offsets): there are \( w_1, \ldots, w_d \in \mathbb{Q} \) and \( b, c_1, \ldots, c_d \in \mathbb{N} \) such that each \( g_i(x_1, \ldots, x_d) = w_1 \cdot (x_1 - c_1) + \ldots + w_d \cdot (x_d - c_d) + b \).

Furthermore, each “piece” \( \text{dom } g_i \) is a linear set.

We won’t prove this; see Chen, Doty, Soloveichik, *Deterministic function computation with chemical reaction networks*. DNA 2012
Semilinear function examples

\[ g_1(x) = x_1 \]
\[ g_2(x) = x_2 \]
\[ \text{dom } g_1 = \{ x_1 > x_2 \} \]

\[ g_1(x) = x_2 \]
\[ g_2(x) = 0 \]
\[ \text{dom } g_1 = \{ x_1 > x_2 \} \]
Computing affine functions (by example)

linear:
\[ f(a,b,c) = 2a + (4/3)b - (5/6)c \]
\[ A \rightarrow 2Y \]
\[ 3B \rightarrow 4Y \]
\[ 6C + 5Y \rightarrow \emptyset \]

General form:
\[ w_1 \ldots w_d \in \mathbb{Q} \text{ and } b, c_1, \ldots, c_d \in \mathbb{N} \]
\[ g_i(x_1, \ldots, x_d) = w_1(x_1-c_1) + \ldots + w_d(x_d-c_d) + b. \]

subtract constant offset \( c_i \) from input \( x_i \):
start with 1 \( L \), a \( A \)'s, b \( B \)'s
\[ f(a,b) = 2a + 3b + 4 \]
\[ L \rightarrow 6Y + L_{a0} + L_{b0} \]
create \( d \) offset, and one leader for each input
remember 3 copies of \( A \)
\[ L_{a0} + A \rightarrow L_{a1} \]
\[ L_{a1} + A \rightarrow L_{a2} \]
\[ L_{a2} + A \rightarrow L_{a3} \]
\[ L_{a3} + A \rightarrow L_{a3} + A' \]
convert remaining \( A \) to \( A' \)
\[ A' \rightarrow 2Y \]
compute \( 2(a-3) \) by doubling \( A' \)
\[ L_{b0} + B \rightarrow L_{b1} \]
remove 1 copy of \( B \)
\[ L_{b1} + B \rightarrow L_{b1} + B' \]
convert remaining \( B \) to \( B' \)
\[ 4B' + 5Y \rightarrow \emptyset \]
compute \( -(5/4)(b-1) \) on \( B' \)
Combining all affine function computations

**Theorem:** If \( f: \mathbb{N}^d \rightarrow \mathbb{N} \) is a semilinear function, then some CRN stably computes \( f \).

**Lemma:** If \( f: \mathbb{N}^d \rightarrow \mathbb{N} \) is a semilinear function, then it is piecewise affine: a finite union of partial affine functions \( g_i: \mathbb{N}^d \rightarrow \mathbb{N} \).

Furthermore, each “piece” \( \text{dom} \ g_i \) is a linear set.

**Proof sketch:**

1. To compute whole semilinear function \( f \), compute all these affine functions \( g_i \) in parallel, storing output of \( g_i \) in species \( Y_i \).
   - *in parallel* means: split each input species \( A \) via reaction \( A \rightarrow A_1 + A_2 + \ldots \), where \( A_i \) is used as input for computing \( g_i \).
2. Also in parallel, for each domain \( \text{dom} \ g_i \), compute the predicate \( [x \in \text{dom} \ g_i?] \).
3. Yes-voters \( T_i \) and no-voters \( F_i \) for \( [x \in \text{dom} \ g_i?] \) do:
   1. \( T_i + Y_i \rightarrow T_i + Y + \hat{Y}_i \) convert \( g_i \)'s output \( Y_i \) to “global” output \( Y \)
   2. \( F_i + Y + \hat{Y}_i \rightarrow F_i + Y_i \) convert back

**Question 1:** what’s the point of species \( \hat{Y}_i \)?

**Question 2:** Something else doesn’t work as described... what is it?

**Answer 2:** Consuming \( Y_i \) can disrupt computation of \( g_i \). Can be solved using *dual-rail encoding*. (not shown)
Limits of stable computation

Non-semilinear functions/predicates cannot be stably computed by CRNs
Impossibility of stably deciding non-semilinear sets

**Theorem**: Every stably decidable set $X \subseteq \mathbb{N}^d$ is semilinear.

We won’t prove this in full generality, but we will prove the simpler corollary that the “squaring set” $X = \{(a, b) \in \mathbb{N}^2 \mid a^2 = b\}$ is not stably decidable.

To start, we use the above theorem to prove the following:

**Theorem**: Every stably computable function $f: \mathbb{N}^d \to \mathbb{N}$ is semilinear.
Theorem: Every stably computable function $f: \mathbb{N}^k \to \mathbb{N}$ is semilinear.

Proof:
1. Let $C$ be a CRN stably computing $f$.
2. We convert $C$ to a CRN $D$ stably deciding $\text{graph}(f) = \{ (x_1,x_2,\ldots,x_k,y) \in \mathbb{N}^{k+1} \mid f(x_1,x_2,\ldots,x_k) = y \}$.
3. Then $\text{graph}(f)$ must be semilinear, since a CRN stably decides it. (By first theorem on previous slide.)
   1. **Key challenge:** $D$ will run $C$ with $D$'s first $k$ inputs. But $D$ has to test the count of $C$'s output $Y$ to compare it to $D$'s last input, without consuming $Y$, since consuming $Y$ could disrupt the correctness of $C$.
   2. **Solution:** we introduce two new species $Y_P$ and $Y_C$ representing $\#Y$ that are only produced by $C$, never consumed, so we are free to add reactions consuming them. This is called dual-rail encoding: $\#Y = \#Y_P - \#Y_C$.
4. For each reaction in $C$ changing the count of output $Y$, add $Y_P$ or $Y_C$ as products to track the change:
   • $X \rightarrow Z + 2Y$ becomes $X \rightarrow Z + 2Y + 2Y_P$ since there are net 2 $Y$'s produced.
   • $X + 6Y \rightarrow 2Y$ becomes $X + 6Y \rightarrow 2Y + 4Y_C$ since there are net 4 $Y$'s consumed.
5. For concreteness, assume $k=1$.
   • CRN $D$ deciding $\text{graph}(f)$ has 2 input species. The first is $A$. Let the second input species be $Y_C$.
6. Since $C$ stably computes $f$, eventually $f(\text{initial } \#A)$ more $Y_P$ are produced than $Y_C$.
7. If and only if initially $f(\#A) = \#Y_C$, then eventually $\#Y_P = \#Y_C$.
8. Add reactions to test for equality between $\#Y_P$ and $\#Y_C$. (not shown, but easy)
Impossibility of stably deciding a non-semilinear set

Theorem: The “squaring set” \( S = \{ (x,y) \in \mathbb{N}^2 \mid x^2 = y \} \) is not stably decidable by any CRN.
Additivity, nondecreasing sequences, minimal elements

**Observation:** Reachability is *additive*: if \( c \Rightarrow d \), then for all \( e \in \mathbb{N}^d \), \( c+e \Rightarrow d+e \), i.e., the presence of extra molecules \( e \) cannot prevent reactions from being applicable.

**Definition:** An infinite sequence of vectors \( c_1, c_2, \ldots \) is *nondecreasing* if \( c_i \leq c_{i+1} \) for all \( i \). (\( c_i \leq c_{i+1} \) means \( c_i(S) \leq c_{i+1}(S) \) for all species \( S \)).

**Definition:** Given \( A \subseteq \mathbb{N}^d \), we say \( y \in A \) is *minimal* if, for all \( x \in A \), \( x \leq y \) implies \( x = y \), i.e., nothing in \( A \) is strictly smaller than \( y \). Let \( \text{min}(A) = \text{minimal elements of } A \).
All vectors have a minimal vector under them

**Observation:** For all $x \in A$, there is a minimal vector $m \in \text{min}(A)$ such that $m \leq x$.

**Proof:**
1. If $x \in \text{min}(A)$ then we’re done.
2. Otherwise, since $x \notin \text{min}(A)$, there is $x_1 \in A$ such that $x_1 < x$.
3. If $x_1 \in \text{min}(A)$ then we’re done since $x_1 \leq x$.
4. Otherwise, since $x_1 \notin \text{min}(A)$, there is $x_2 \in A$ such that $x_2 < x_1$.
5. ...
6. Since there are only a finite number of $y$ in $\mathbb{N}^d$ such that $y < x$, this process must terminate with a minimal vector $m \in \text{min}(A)$. QED
Dickson’s Lemma: Nondecreasing subsequences

Dickson’s Lemma: (1) Every infinite sequence \((x_0, x_1, \ldots)\) of vectors in \(\mathbb{N}^d\) has an infinite nondecreasing subsequence, and (2) every set \(A \subseteq \mathbb{N}^d\) has a finite number of minimal elements.

Proof:
1. We’ll show condition (1) by induction on \(d\).
2. Base case \(d = 1\): Let \(X = x_0, x_1, \ldots\) be an infinite sequence of nonnegative integers.
   1. case 1: some \(x \in \mathbb{N}\) appears infinitely often. Let the subsequence be \((x, x, \ldots)\), e.g. 1,1,5,3,4,3,4,3,4,3,4,3,4,...
   2. case 2: every \(x \in \mathbb{N}\) appears finitely many times.
      1. First element of subsequence is \(y_0 = x_0\).
      2. Assuming we have finite increasing subsequence \(y_0 < y_1 < \ldots < y_{k-1}\), let \(x_j\) be last occurrence of any integer \(\leq y_{k-1}\) in original sequence \(x_0, x_1, \ldots\), and let \(y_k = x_{j+1}\), e.g., 2,1,0, 3,2,1, 4,3,2, 5,4,3, 6,5,4, 7,6,5, 8,7,6, 9,8,7, 10,9,8, 11,10,9...
3. Inductive case \(d > 1\):
   1. Inductively pick infinite subsequence \(X'\) such that the length-(\(d-1\)) prefix vectors are nondecreasing.
   2. Pick an infinite subsequence of \(X'\) such that the \(d\)th elements are also nondecreasing, as in base case.
4. For condition (2), suppose that \(\min(A)\) is infinite; put them in any order to make an infinite sequence.
5. By first condition, there’s an infinite nondecreasing subsequence \(m_1 \leq m_2 \leq \ldots\) of distinct vectors in \(\min(A)\).
6. Since they are distinct, \(m_1 < m_2 < \ldots\), but \(m_1 < m_2\) contradicts the minimality of \(m_2\). QED
Properties of stable configurations

• For convenience, assume every species votes.
• Thus a **stable** YES-output configuration $o$ with output $\varphi(o) = \text{YES}$ is one in which, for all $o' \in \text{Reach}(o)$, all NO voters are absent from $o'$. (Similarly for stable NO-output.)
• Conversely, an **unstable** configuration $c$ is one in which at least one of the following holds:
  • YES and NO voters both exist already in $c$ (output undefined)
  • Only YES voters exist, but a NO voter is producible in some $c' \in \text{Reach}(c)$.
  • Only NO voters exist, but a YES voter is producible in some $c' \in \text{Reach}(c)$.
• By additivity, for all $\delta \in \mathbb{N}^q$, $c+\delta$ is unstable as well, since $c'+\delta \in \text{Reach}(c+\delta)$ (*since $c'$ has the contradictory voter, so does $c'+\delta$*), leading to the following observation:

**Observation:** The **unstable configurations are closed upwards:** for all unstable $c$ and all $d \geq c$, $d$ is also unstable.

**Corollary:** The **stable configurations are closed downwards:** for all stable $c$ and all $b \leq c$, $b$ is also stable.
Definition: For all $c \in \mathbb{N}^d$, let $\nabla(c) = \{ d \in \mathbb{N}^d | c \leq d \}$ denote the upper cone of $c$.

Observation (reworded from previous slide): For all unstable $c$ and all $d \in \nabla(c)$, $d$ is also unstable.
Set of unstable configurations is finite union of cones

Recall:

**Observation 1:** Unstable configs are closed upwards:
For all unstable \(c\) and all \(d \in \nabla(c)\), \(d\) is also unstable.

**Observation 2:** For all \(x \in A\), there is a minimal \(m \in \min(A)\) such that \(m \leq x\).

**Lemma:** Let \(U\) be the set of unstable configurations. Then \(U = \bigcup_{m \in \min(U)} \nabla(m)\). \((U\ is\ a\ finite\ union\ of\ cones.)\)

**Proof:**
1. Let \(C = \bigcup_{m \in \min(U)} \nabla(m)\). Need to show \(C = U\).
2. To see that \(C \subseteq U\), let \(x \in C\), i.e., \(x \in \nabla(m)\) for some \(m \in \min(U)\).
3. By Observation 1, since \(m \in U\), also \(x \in U\), so \(C \subseteq U\).
4. To see that \(U \subseteq C\), let \(x \in U\).
5. By Observation 2, for some \(m \in \min(U)\), \(m \leq x\).
6. Thus \(x \in \nabla(m) \subseteq C\), so \(U \subseteq C\). QED

![Diagram showing cones and minimal points](image-url)
Stable configurations are closed upwards for species that are already “large”

Recall stable configs are closed downward. They are also closed upward for “already large” species.

**Definition:** Let \( \tau = \max \{ m(S) \mid m \in \min(U), S \in \Lambda \} \). The hypercube with corner \((\tau, \tau, ..., \tau) \in \mathbb{N}^d \) (and other corner at origin) contains every minimal \( m \) defining \( U \).

**Lemma:** Let \( c \) be stable such that for some species \( S \in \Lambda, c(S) \geq \tau \). Let \( d = c + \{ \text{any amount of } S \} \). Then \( d \) is also stable.

**Proof:** By picture. \( \tau = 6, c(S) = 6, d(S) = 8 \). If \( c \) is not already in a cone \( \nabla(m) \) defining the unstable configurations \( U \), we cannot enter any cone by adding more \( S \).
A pumping lemma

**Pumping Lemma**: Suppose a CRN stably decides infinite set $A \subseteq \mathbb{N}^d$. Then there are $c<d$ such that, letting $\delta = d - c$, for all $n \in \mathbb{N}$, $c + n\delta \in A$.

**Proof:**

1. By Dickson’s Lemma there is infinite nondecreasing subsequence $c_0 \leq c_1 \leq \ldots$, each $c_i \in A$. Let $\delta_i = c_{i+1} - c_i$.
2. Define sequence of stable $o_0, o_1, \ldots$ inductively as follows.
3. **Base case**: $c_0 \Rightarrow o_0$ for some stable $o_0$.
4. **Inductive case**: By additivity $c_{i+1} = c_i + \delta_i \Rightarrow o_i + \delta_i$.
5. By correctness $o_i + \delta_i \Rightarrow o_{i+1}$ for some stable $o_{i+1}$.
6. By Dickson’s Lemma pick infinite nondecreasing subsequence $o_0' \leq o_1' \leq \ldots$ of $o_i$’s. For the sake of readability let’s assume this is just the original sequence $o_0 \leq o_1 \leq \ldots$.
7. Let $\Gamma = \{ S \mid \lim_{j \to \infty} o_i(S) = \infty \}$ (*species with unbounded counts*).
8. For large enough $i$, if $S \in \Gamma$, then $o_i(S) \geq \tau$, and if $S \not\in \Gamma$, then $o_i(S) = c_S$ where $c_S$ is the largest $S$ ever gets in the $o_i$’s.
9. Then $o_{i+1}(S) = o_i(S)$ if $S \not\in \Gamma$ and $o_i(S) \geq \tau$ otherwise.
A pumping lemma (proof continued)

**Pumping Lemma**: Suppose a CRN stably decides infinite set $A \subseteq \mathbb{N}^d$. Then there are $c < d$ such that, letting $\delta = d - c$, for all $n \in \mathbb{N}$, $c + n\delta \in A$.

**Proof**: (continued)

1. Fix large enough $i$ that $o_{i+1}(S) = o_i(S)$ if $S \notin \Gamma$ and $o_i(S) \geq \tau$ otherwise.
2. Write $\epsilon = o_{i+1} - o_i$.
3. Note that $\epsilon(S) > 0$ implies $o_i(S) \geq \tau$.
4. Then $o_i + \delta_i \Rightarrow o_{i+1} = o_i + \epsilon$, i.e., $o_i$ is like a “catalyst” that transforms $\delta_i$ into $\epsilon$.
5. Apply the same execution to $n$ copies of $\delta_i$: $o_i + n\delta_i \Rightarrow o_i + n\epsilon$.
6. $o_i + n\epsilon$ is larger than $o_i$ only on species $S$ with count $o_i(S) \geq \tau$.
7. By closure of stable configurations upwards for “already large” species, since $o_i$ is stable, $o_i + n\epsilon$ is also stable, with the same output YES, since they have the same species present.
8. In other words, we can reach from $c_i + n\delta_i$ to a stable YES configuration, so $c_i + n\delta_i \in A$ for all $n \in \mathbb{N}$.
9. Let $c = c_i$ and $d = c_{i+1}$, with $\delta = \delta_i$. QED
Impossibility of stably deciding squaring set

**Pumping Lemma**: Suppose a CRN stably decides infinite set $A \subseteq \mathbb{N}^d$. Then there are $c<d$ such that, letting $\delta = d-c$, for all $n \in \mathbb{N}$, $c+n\delta \in A$.

**Theorem**: The “squaring set” $S = \{ (x,y) \mid x^2=y \}$ is not stably decidable by any CRN.

**Proof**:  
1. By our Pumping Lemma, there are points $c=(x,x^2)$ and $d=(z,z^2)$, $x < z$, such that, letting $\delta = d-c$, for all $n \in \mathbb{N}$, $c+n\delta \in S$.  
2. **Claim**: the point $c+2\delta \notin S$, contradicting our Pumping Lemma.  
3. Proof: by picture. (straight line intersects a parabola at ≤ 2 points)  
4. Formally, suppose otherwise: $c+2\delta = (2z-x, 2z^2-x^2) \in S$.  
5. Then $(2z-x)^2 = (2z^2-x^2)$, so  
   
   $0 = (2z-x)^2 - (2z^2-x^2)$  
   $= (4z^2-4xz+x^2) - (2z^2-x^2)$  
   $= 2z^2 - 4xz + 2x^2$  
   $= 2(z-x)^2$, which contradicts $x \neq z$. QED
Limits of efficient stable computation
What is known to be computable in less than time $O(n)$?

**Predicates**

Boolean combination of detection predicates

“detection” means $\varphi(a) = [a > 0]$?

$\varphi(a,b,c) = a>0 \text{ OR } (b>0 \text{ AND } c=0)$

i.e., constant except when a variable changes from 0 to positive

**Functions**

$\mathbb{N}$-linear functions (coefficients are nonnegative integers)

E.g., $f(a,b) = 2a + 3b$

$a \rightarrow y+y$

$b \rightarrow y+y+y$

Both computable in $O(\log n)$ time

[Angluin, Aspnes, Eisenstat, Fast computation by population protocols with a leader, DISC 2006]
[Chen, Doty, Soloveichik, Deterministic function computation with chemical reaction networks, DNA 2012]
Known time lower bounds: leader election/majority

**Leader election**

Leader election (computing the constant function $f(a)=1$) requires $\Omega(n)$ time

**Majority (and other “explicit” predicates)**

Majority (and many other “explicit” predicates such as equality) require $\Omega(n / \text{polylog } n)$ time, even with up to $\frac{1}{2} \log \log n$ states.*

If the protocol satisfies a technical condition called “output dominance”, then even with up to $\log n$ states, $\Omega(n^{0.999})$ time is required.**

* [Alistarh, Aspnes, Eisenstat, Gelashvili, Rivest, SODA 2017]

** [Alistarh, Aspnes, Gelashvili, SODA 2018]: “output dominance” = changing positive counts of states in a stable configuration leaves it able to reach a stable configuration with the same output
Known time lower bounds: “most” predicates/functions

• **Informal**: “most” semilinear predicates and functions not known to be computable in $o(n)$ time, actually require at least $\Omega(n)$ time to compute

• **Definition**: $\varphi : \mathbb{N}^k \rightarrow \{Y,N\}$ is **eventually constant** if there is $m \in \mathbb{N}$ so that $\varphi(a) = \varphi(b)$ for all $a, b$ with all components $\geq m$

• **Definition**: $f : \mathbb{N}^k \rightarrow \mathbb{N}$ is **eventually $\mathbb{N}$-linear** if there is $m \in \mathbb{N}$ so that $f(a)$ is $\mathbb{N}$-linear for all $a$ with all components $\geq m$
  • Both definitions allow exceptions “near a face of $\mathbb{N}^k$”

• **Formal theorem**: Every predicate that is not eventually constant, and every function that is not eventually $\mathbb{N}$-linear, requires at least time $\Omega(n)$ to compute.
  • They’re all computable in at most $O(n)$ time, so this settles their time complexity.

[Belleville, Doty, Soloveichik, *Hardness of computing and approximating predicates and functions with leaderless population protocols, ICALP 2017*]
# What is currently known/unknown

<table>
<thead>
<tr>
<th>Predicates</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>computable in (O(\log n)) time</td>
<td>\text{detection} (constant unless changing between 0 and positive) (a &gt; 0 \text{ AND } (b &gt; 0 \text{ OR } c = 0))</td>
</tr>
<tr>
<td>not computable in less than (\Omega(n)) time</td>
<td>non-eventually constant (a &gt; b? \quad a = b? \quad a\text{ is odd?})</td>
</tr>
<tr>
<td>unknown (best known protocol is (O(n)) time)</td>
<td>eventually constant but not constant on all positive values (a &gt; 1?)</td>
</tr>
</tbody>
</table>

**Diagram:**
- \(f(a)\) is defined as a piecewise function handling different cases based on the value of \(a\).
Other modeling choices?
Modeling choices in formalizing “Computing with chemistry”

- **integer counts** ("stochastic") or **real concentrations** ("mass-action")?
- what is the object being “computed”?
  - yes/no decision problem? “number of A’s > number of B’s?”
  - numerical function? “make Y become double the amount of X”
- **guaranteed to get correct answer** or allow **small probability of error**
  - if Pr[error] = 0, system works no matter the reaction rates
- to represent an input \( n_1, \ldots, n_k \), what is the initial configuration?
  - only input species present
  - auxiliary species can be present
- when is the computation **finished**? when...
  - the output stops changing? (convergence)
  - the output becomes unable to change? (stabilization)
  - a certain species \( T \) is first produced? (termination)
- require **exact numerical answer**? or allow an **approximation**?

first part of slides

summarized in next few slides
Auxiliary species present initially ≈ “initial leader”

Instead of starting with \{ 100 A \} to represent input value 100, start with \{ 1 L, 100 A \}

some predicates/functions get “easier” (i.e., it’s easy to think of the reactions)

parity: \( \varphi(a) = \text{“a is odd”} \)

\[
\begin{align*}
\text{without a leader} & \\
A_o + A_o & \rightarrow A_e + a_e \\
A_e + A_e & \rightarrow A_e + a_e \\
A_o + A_e & \rightarrow A_o + a_o \\
A_o + a_e & \rightarrow A_o + a_o \\
A_e + a_o & \rightarrow A_e + a_e \\
\end{align*}
\]

\[
\begin{align*}
\text{with a leader} & \\
L_e + A & \rightarrow L_o \\
L_o + A & \rightarrow L_e \\
\end{align*}
\]

But fundamental computability doesn’t change: exactly the semilinear predicates/functions can be computed (same as without a leader).

[Angluin, Aspnes, Diamadi, Fischer, Peralta, PODC 2004] [Angluin, Aspnes, Eisenstat, PODC 2006]
[Chen, Doty, Soloveichik, DNA 2012] [Doty, Hajijaghayi, DNA 2013]
**Theorem:** Without a leader, all non-eventually constant predicates and non-eventually $\mathbb{N}$-linear functions require at least $\Omega(n)$ stabilization time. [Belleville, Doty, Soloveichik, ICALP 2017]

**Previous work:** With a leader, all semilinear predicates/functions can be computed in at most $O(\log^5 n)$ convergence time. [Angluin, Aspnes, Eisenstat, DISC 2006]

**Conjecture:** With a leader, all non-detection predicates and non-$\mathbb{N}$-linear functions require at least $\Omega(n)$ stabilization time.

**False conjecture:** Without a leader, all non-detection predicates and non-$\mathbb{N}$-linear functions require at least $\Omega(n)$ convergence time.

[resolved negatively by Kosowski, Uznański, Population Protocols are Fast, PODC Brief Announcement 2018]
What if we use real-valued concentrations?

**Theorem:** A function is stably computable by an integer-valued chemical reaction network if and only if it is semilinear.

A function is stably computable by a real-valued chemical reaction network if and only if it is continuous and piecewise linear.

---

**Semilinear example**

\[ f(n) \]

\[ n \]

[Angluin, Aspnes, Eisenstat, *PODC* 2006]

[Chen, Doty, Soloveichik, *DNA* 2012]

---

**Continuous piecewise linear example**

\[ \min(x_1, \max(2^x_1 - x_2, 2^x_1 + x_2)) \]

[Chen, Doty, Reeves, Soloveichik, *JACM* 2023]
What if we allow a small probability of error? (i.e., allow reaction rates to influence outcome)

**Theorem**: A function is computable with probability of error < 1% by an integer-valued chemical reaction network if and only if it is computable by any algorithm whatsoever... [Soloveichik, Cook, Bruck, Winfree, *Natural Computing* 2008]

Furthermore, computation doesn’t merely converge to the correct answer eventually, but can be made “terminating”: producing a molecule $T$ signaling when the computation is done. (provably impossible when $Pr[error] = 0$)

**Conjecture**: Even without a leader, any computable function can be efficiently computed with high probability.
What if we use real-valued concentrations... and allow reaction rates to influence outcome??

Theorem: A function is computable by a real-valued chemical reaction network using mass-action kinetics if and only if it is computable by any algorithm whatsoever.


mass-action kinetics:

\[
\begin{align*}
X \xrightarrow{k_1} & \ Y + Y \\
Y + Z \xrightarrow{k_2} & \ X \\
\end{align*}
\]

\[
\begin{align*}
\dot{[X]} & = -k_1[X] + k_2[Y][Z] \\
\dot{[Y]} & = 2k_1[X] - k_2[Y][Z] \\
\dot{[Z]} & = -k_2[Y][Z] \\
\end{align*}
\]

... with only a polynomial-time slowdown.

Fast approximate division by 2

initial configuration: 
\{ n \ X, \ \varepsilon n \ A, \ \varepsilon n \ B \} 

\[ X + A \rightarrow B + Y \]
\[ X + B \rightarrow A \]

guaranteed to get
\[ Y = n/2 \pm \varepsilon n \]
\[ E[\text{time}] = O(\log n) / \varepsilon \]

[Belleville, Doty, Soloveichik, Hardness of computing and approximating predicates and functions with leaderless population protocols, ICALP 2017]
CRN computation with a small chance of error
Counter (register) machine

If empty goto 6

1) dec r
2) inc s
3) inc s
4) inc s
5) dec t
6) inc s

HALT

"input" counter

computes $f(n) = 3n+1$
Counter machines

• Finite state machine with a fixed number of counters $c_1, c_2, \ldots, c_k$, each holding a nonnegative integer.

• Start with inputs $n_1, n_2, \ldots, n_l \in \mathbb{N}$ as values of $c_1, c_2, \ldots, c_l$, and $c_{l+1}, \ldots, c_k$ start 0.

• Finite-state machine, where each state is one of:
  • $\text{inc } c$: increment counter $c$
  • $\text{dec } c$: decrement counter $c$; no effect if $c = 0$
  • $\text{if } c=0 \text{ goto } i$: if counter $c$ is 0, then jump to state $i$
  • $\text{goto } i$ (can be shorthand for $\text{if } c=0 \text{ goto } i$ for unused $c$)

• may also have accept/reject semantics, or interpret the final value of some counter as the output
Example counter machines

input a  \( f(a) = 2a \)

1. if \( a=0 \) goto 6
2. dec a
3. inc b
4. inc b
5. goto 1
6. end

input a  \( f(a) = \lfloor a/2 \rfloor \)

1. while \( a>0 \):
2. dec a
3. dec a
4. inc b
5. goto 1
6. accept
7. reject

inputs a,b  \( f(a,b) = ab \)

1. while \( a>0 \):
2. dec a
3. while \( b>0 \):
4. dec b
5. inc c
6. inc d
7. while \( c>0 \):
8. dec c
9. inc b

input a  \( f(a) = 2^a \)

1. inc b
2. while \( a>0 \):
3. dec a
4. while \( b>0 \):
5. dec b
6. inc c
7. inc c
8. while \( c>0 \):
9. dec c
10. inc b

... i-1. goto 1
i. ...

input a  \( \varphi(a) = \text{"a is odd"} \)

1. if \( a=0 \) goto 7
2. dec a
3. if \( a=0 \) goto 6
4. dec a
5. goto 1
6. accept
7. reject

\( f(\varphi(a)) = \varphi(f(a)) \)

\( f(\varphi(a)) = \varphi(f(a)) \)

\( f(\varphi(a)) = \varphi(f(a)) \)
3-counter machines are Turing universal

Assume Turing machine
- has a single blank on rightmost cell
- if rightmost blank overwritten, it grows a new blank cell to right

Need a third “work” counter $c$ to help do the following operations on counters $a$ and $b$:

<table>
<thead>
<tr>
<th>Turing machine operation</th>
<th>Counter machine implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>read bit under tape head</td>
<td>is $a$ odd?</td>
</tr>
<tr>
<td>change bit under tape head</td>
<td>inc/dec $a$</td>
</tr>
<tr>
<td>move tape head right</td>
<td>set $a = 2a \ (\ + \ 1) ; \ set \ b = \lfloor b/2 \rfloor$</td>
</tr>
<tr>
<td>move tape head left</td>
<td>set $b = 2b \ (\ + \ 1) ; \ set \ a = \lfloor a/2 \rfloor$</td>
</tr>
<tr>
<td>test if tape head is on blank and if so, change it to 1</td>
<td>if $b=1$ then \ set \ $a = 2a + 1$ \</td>
</tr>
</tbody>
</table>

Interpret tape on each side of tape head as binary number; append new leading 1 to make this mapping 1-1, in case the binary string has no leading 1 already, since 00111_2, 0111_2, and 111_2 are all considered the number 7.

1-counter machines are not Turing-universal... why?
2-counter machines are (sort of) Turing universal

[Minsky 1967, Computation: Finite and Infinite Machines]

• To represent counter values \((a,b,c)\) in a single counter \(x\), let \(x = 2^a \cdot 3^b \cdot 5^c\) and \(y = 0\).
  - To increment \(b\), set \(x = 3x\). (using \(y\) as a work counter)
  - To decrement \(a\), set \(x = \lfloor x/2 \rfloor\).
  - To test if \(c = 0\), test if \(x \equiv 0 \) mod 5.

• To start with \(a = n\) and \(b = c = 0\), start with \(x = 2^n \cdot 3^0 \cdot 5^0 = 2^n\).

• If \(f: \mathbb{N} \rightarrow \mathbb{N}\) is any computable function, this machine can start with \(x=2^n\) and halt with \(x=2^{f(n)}\).

• **Caveat about encoding:** there is no 2-counter machine that starts with \(x=n\) and halts with \(x=2^n\).

  [Schroeppel 1972, A Two Counter Machine Cannot Calculate \(2^n\)]
  “Theorem: Any counter machine can be simulated by a 2-counter machine, provided an obscure coding is accepted for the input and output.”

• 2-counter machines can do universal computation on encoded inputs (\(n\) encoded as \(2^n\)), but they cannot compute the encoding/decoding themselves.

• However, the fact that 2-counter machines can simulate arbitrary 3-counter machines implies that the Halting Problem for 2-counter machines is undecidable.
Finite automaton occupying a point \((x, y) \in \mathbb{N}^2\).

It cannot write anything, or see anything.

It can sense if it is touching the southern wall, or western wall (or both).

It can move north, south, east, or west based on its current state and 2 “wall bits”, and of course change state:

\[
\delta: S \times \{\text{wall, no wall}\}^2 \rightarrow S \times \{\uparrow, \downarrow, \leftarrow, \rightarrow\}
\]

There is an automaton \(A\) so that this problem is undecidable: given \((x, y) \in \mathbb{N}^2\), if started at \((x, y)\), will \(A\) ever visit the lower-left corner?
CRNs can simulate counter machines with probability < 1

<table>
<thead>
<tr>
<th>Counter machine:</th>
<th>CRN:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r = \text{input } n, \text{ start line 1} )</td>
<td>initial state { ( n \ R ), 1 ( L_1 ) }</td>
</tr>
</tbody>
</table>

1) **inc** \( r \)  
2) **dec** \( r \)  \text{ if zero goto 1}  
3) **inc** \( s \)  
4) **dec** \( s \)  \text{ if zero goto 2}  

1) \( L_1 \rightarrow L_2 + R \)  
2) \( L_2 + R \rightarrow L_3 \)  
3) \( L_3 \rightarrow L_4 + S \)  
4) \( L_4 + S \rightarrow L_5 \) 

Error occurs when \( R \) is present, but reaction \( L_2 \rightarrow L_1 \) occurs instead of \( L_2 + R \rightarrow L_3 \).  
Semantic effect on register machine: when \( r > 0 \), it may jump from line 2 to 1 without decrementing.  
There’s a positive probability of error; how to reduce it? Need to slow down \( L_2 \rightarrow L_1 \).

Problem with adjusting rate constant to slow down reactions for achieving Turing-universal computation

**Could make rate constant $k$ very small**

- If correct reaction $r_c: L_2 + R \rightarrow L_3$ has rate constant 1, how small should $k$ be to achieve $\Pr[r_i \text{ occurs instead of } r_c] = \Pr[\text{error}] = \varepsilon$?
- rate of $r_c = \lambda_c = \#L_2 \cdot \#R/v = \#R/v \geq 1/v$
- rate of $r_i = \lambda_i = k \cdot \#L_2 = k$
- $\Pr[\text{error}] = \lambda_i / (\lambda_i + \lambda_c) \leq k / (k + 1/v)$
- For $\Pr[\text{error}] = \varepsilon$, set $k = \varepsilon / (v - v\varepsilon) \approx \varepsilon / v$
Problems with simulation scheme so far

1. Adjusting rate constants means designing new chemicals.
   • Easier to adjust counts of existing molecules than to design new ones.

2. \( \text{Pr[error in any time step]} \) increases for longer computations.
   • By union bound we can only say \( \text{Pr[error in any time step]} \leq \varepsilon \cdot t \) \((t = \text{running time})\), so to achieve total error probability \( \leq \delta \) over all the computation requires setting \( \varepsilon \leq \delta/t \), i.e., \( k \leq \delta/(t \cdot v) \).
   • Universal computation requires that we can simulate a program without knowing in advance how many steps it will take.

3. Reducing error slows down the computation “significantly”.
   • Halving rate constant \( k \) decreases \( \text{Pr[error]} \) by half, but doubles expected running time of all jump steps.

4. Register machines are exponentially slower than Turing machines.

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

   • Problem 5 is fundamental in CRNs: they necessarily store a “unary” encoding of any integer.
   • **Theorem(ish):** There is a CRN solving problems 1–4.
   • We’ll see how to solve problems 1–3 by simulating a register machine more efficiently.
   • To handle Problem 4, see [Soloveichik, Cook, Winfree, Bruck, *Computation with Finite Stochastic Chemical Reaction Networks*, \( \text{NaCo} \ 2008 \)]
How to slow down reaction $L_2 \rightarrow L_1$?

Use a clock:
1 $C_1$, 1 $F$, $n$ $B$

$F + C_1 \rightarrow F + C_2$
$F + C_2 \rightarrow F + C_3$

$C_k + L_2 \rightarrow C_1 + L_1$

$B + C_2 \rightarrow B + C_1$
$B + C_3 \rightarrow B + C_2$

$\vdots$

$C_k$ appears after expected time $\approx n^{k-1}$

$E[\text{time for } L_2 + R \rightarrow L_3] \leq n$

reverse-biased random walk
How to handle the three problems

Recall three problems we claimed we would solve:
1. Adjusting rate constants means designing new chemicals.
2. \( \Pr[\text{error in any time step}] \) increases for longer computations.
3. Reducing error slows down the computation “significantly”.

**Problem 1:** Now all rate constants = 1.

**Problem 2:** How to make \( \Pr[\text{error in any time step}] < \varepsilon \), no matter how long the computation goes?

\[
\begin{align*}
F + C_1 & \rightarrow F + C_2 \\
F + C_2 & \rightarrow F + C_3 \\
B + C_2 & \rightarrow B + C_1 \\
B + C_3 & \rightarrow B + C_2
\end{align*}
\]

Two competing reactions, \( r_i \) incorrect, and \( r_c \) correct:
\( r_i: C_k + L_2 \rightarrow C_1 + L_1 + B \)
\( r_c: L_2 + R \rightarrow L_3 + B \)

So \( \Pr[r_i \text{ ever occurs when it shouldn’t}] \leq \sum_{n=1}^{\infty} 1/n^2 = \pi^2/6 \).

Still not a great probability bound, but we can scale that to any constant error probability \( \varepsilon \) by setting starting value of \( B \):

For \( \varepsilon = 1/100 \), set initial \#\( B \) = 102, since \( \sum_{n=102}^{\infty} 1/n^2 < 0.01 \).

**Solution:** increase \( B \) after every decrement and jump:
\( r_i: C_k + L_2 \rightarrow C_1 + L_1 + B \)
\( r_c: L_2 + R \rightarrow L_3 + B \)

**Problem 3:** Also solved! i.e., halving error probability no longer doubles computation time (*derivation not shown*)