The computational power of execution bounded chemical reaction networks

David Doty, Ben Heckmann

June 2024

UC Santa Cruz Applied Mathematics Department Seminar

UCDAVIS
COMPUTER SCIENCE

Acknowledgments

Ben Heckmann
Undergraduate student
Technische Universität München, UC Davis

Matthias Köppe
Professor
UC Davis

For teaching us about
"*Theorems of the Alternative*"

Chemical reaction networks

Chemical reaction networks

reactant(s) $R \rightarrow P_1 + P_2$ *product(s)*

Chemical reaction networks

reactant(s) $R \rightarrow P_1 + P_2$ *product(s)*

monomers $M_1 + M_2 \rightarrow D$ *dimer*

Chemical reaction networks

reactant(s) $R \rightarrow P_1 + P_2$ *product(s)*

monomers $M_1 + M_2 \rightarrow D$ *dimer*

catalyst $C + X \rightarrow C + Y$

Chemical reaction networks

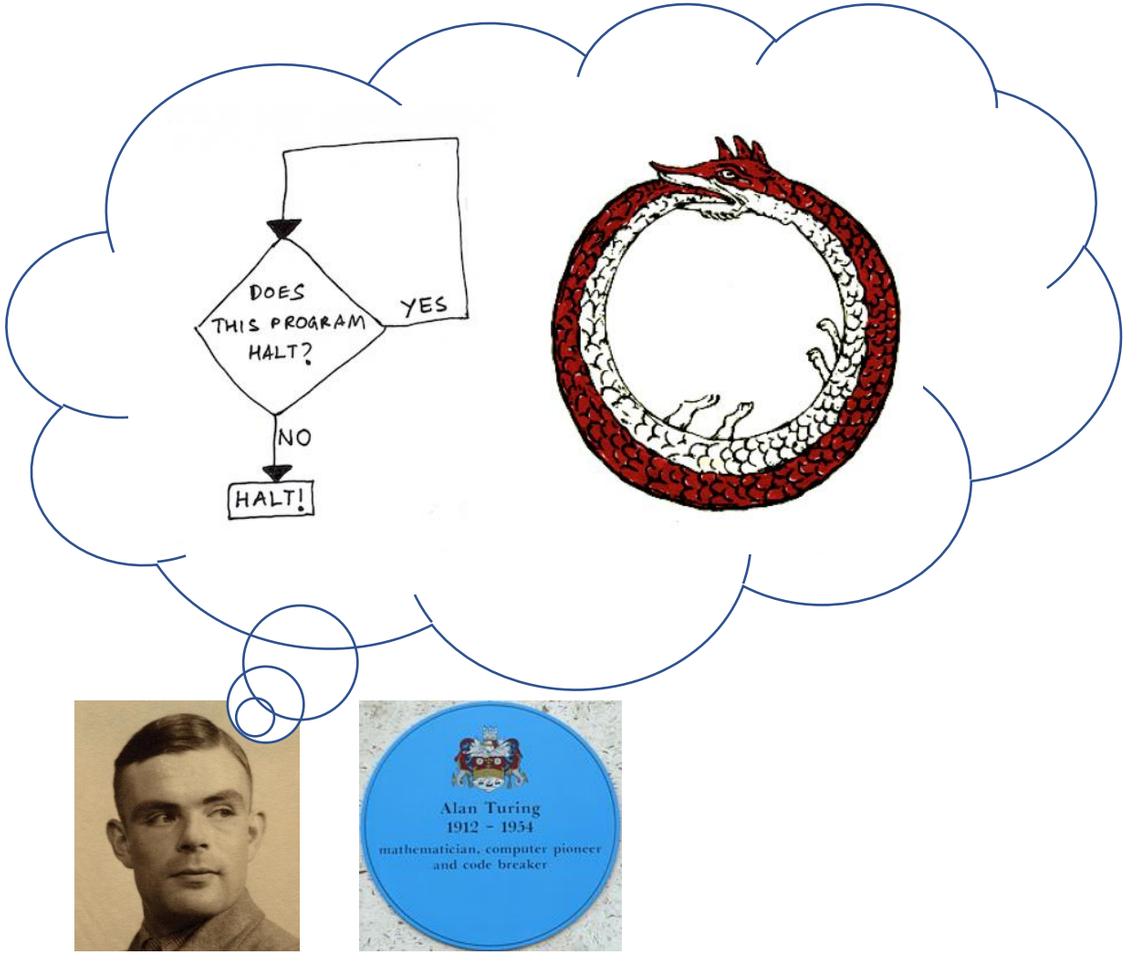
reactant(s) $R \rightarrow P_1 + P_2$ *product(s)*

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

Theoretical Computer Science Approach



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

NP

NP-complete

protein folding

Boolean satisfiability

Hamiltonian path

integer factoring

P

DNA sequence alignment

polynomial factoring

integer multiplication

shortest path

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

Outline

- **Formal definition of chemical reaction networks**
- Execution bounded chemical reaction networks and linear potential functions
- What is “computation” with chemical reactions?
- Limitations of computation with execution bounded chemical reaction networks
- Possibilities of computation with execution bounded chemical reaction networks

Chemical Reaction Network (CRN)

- finite set of *d* species $\Lambda = \{ A, B, C, D, \ldots \}$
- finite set of reactions: *e.g.*

$$A+B \rightarrow A+C$$

$$C \rightarrow A+A$$

$$C+B \rightarrow C$$

- state $\mathbf{x} \in \mathbb{N}^d$: molecular counts of each species

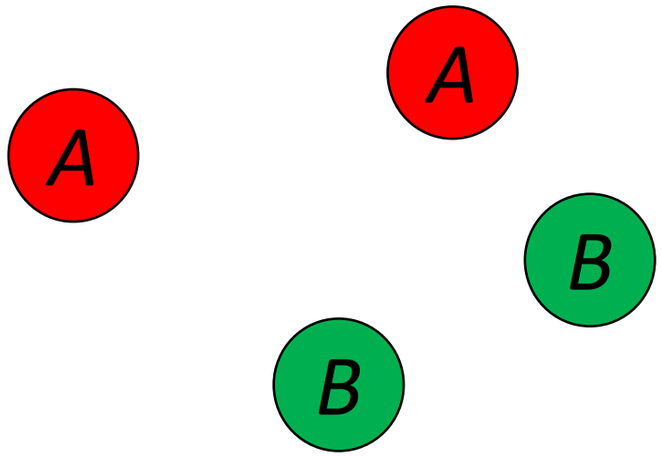
What is **possible**:

Example execution (reaction sequence)

α: $A + B \rightarrow A + C$

β: $C \rightarrow A + A$

A	*B*	*C*
x =	**(2,**	**2,**	**0)**



What is **possible**:

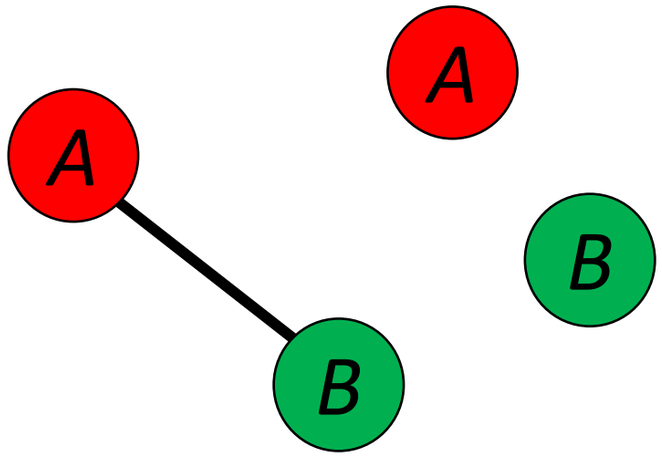
Example execution (reaction sequence)

α: $A+B \rightarrow A+C$

β: $C \rightarrow A+A$

A	*B*	*C*
2	2	0

x = (2, 2, 0)

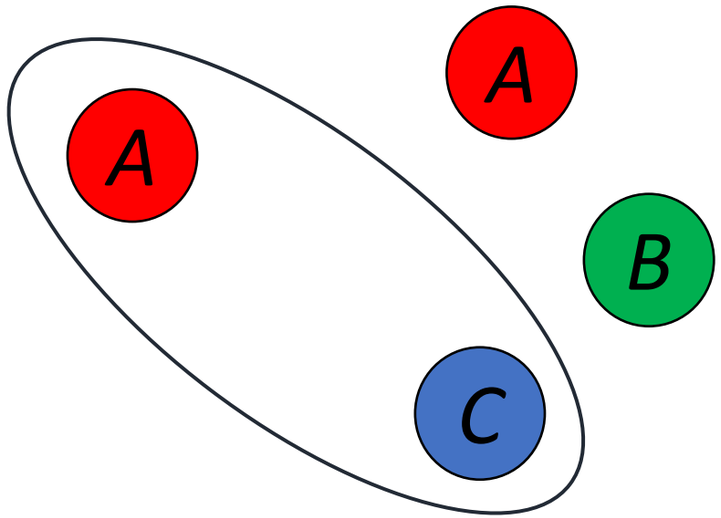


What is **possible**:

Example execution (reaction sequence)

α: $A + B \rightarrow \boxed{A + C}$

β: $C \rightarrow A + A$



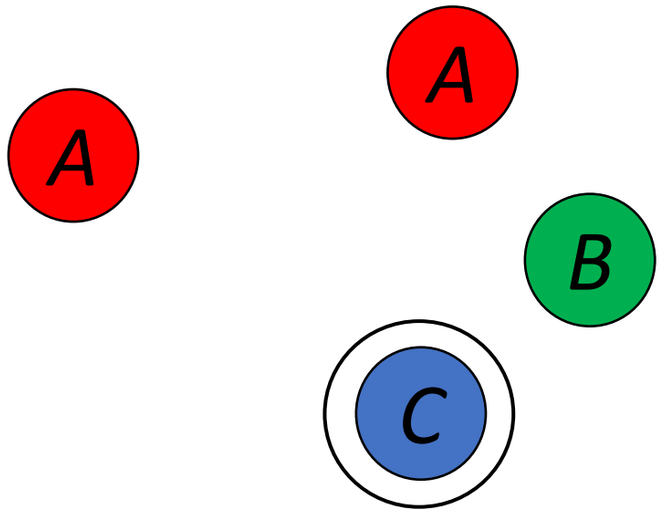
	A	*B*	*C*
x =	2,	2,	0)
	α ⇓		
	2,	1,	1)

What is **possible**:

Example execution (reaction sequence)

α: $A + B \rightarrow A + C$

β: $\boxed{C} \rightarrow A + A$



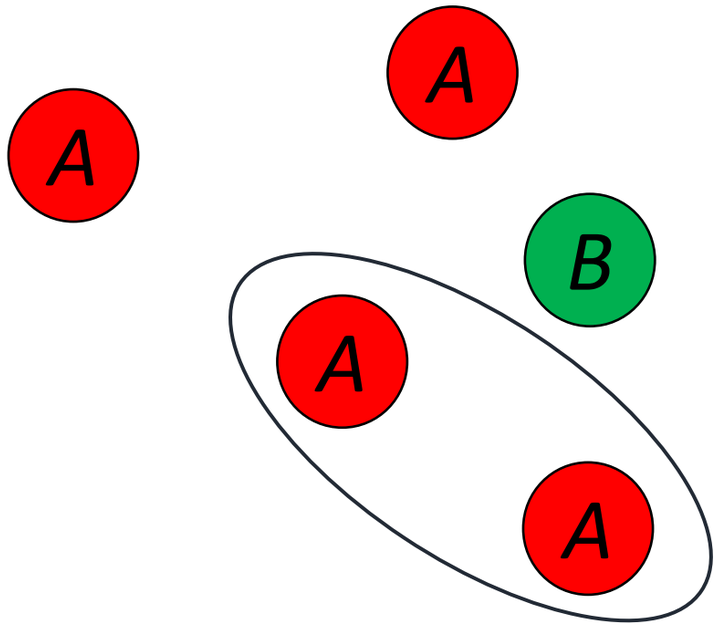
A	B	C
x = (2,	**2,**	**0)**
	α ⇓	
(2,	**1,**	**1)**

What is **possible**:

Example execution (reaction sequence)

α: $A+B \rightarrow A+C$

β: $C \rightarrow A+A$



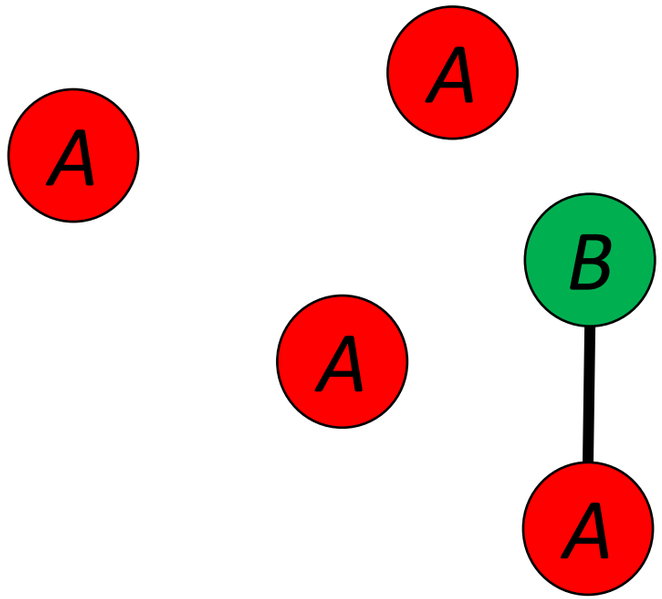
	A	*B*	*C*
x =	**(2,**	**2,**	**0)**
	α ⇓		
	(2,	**1,**	**1)**
	β ⇓		⇘α
	(4,	**1,**	**0)** ···

What is **possible**:

Example execution (reaction sequence)

α: $A+B \rightarrow A+C$

β: $C \rightarrow A+A$



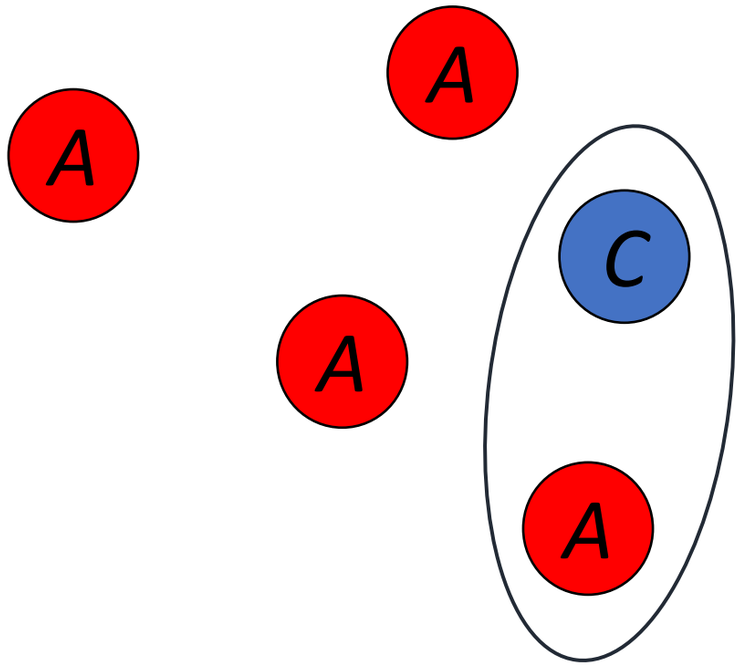
	A	*B*	*C*
x =	**(2,**	**2,**	**0)**
	α ⇓		
	(2,	**1,**	**1)**
	β ⇓		⇘α
	(4,	**1,**	**0)** ···

What is **possible**:

Example execution (reaction sequence)

α: $A+B \rightarrow \boxed{A+C}$

β: $C \rightarrow A+A$



	A	*B*	*C*
x =	**(2,**	**2,**	**0)**
	α ⇓		
	(2,	**1,**	**1)**
	β ⇓		⇘α
	(4,	**1,**	**0)** ...
	α ⇓		
	(4,	**0,**	**1)**
	...		

Key property of reachability: additivity

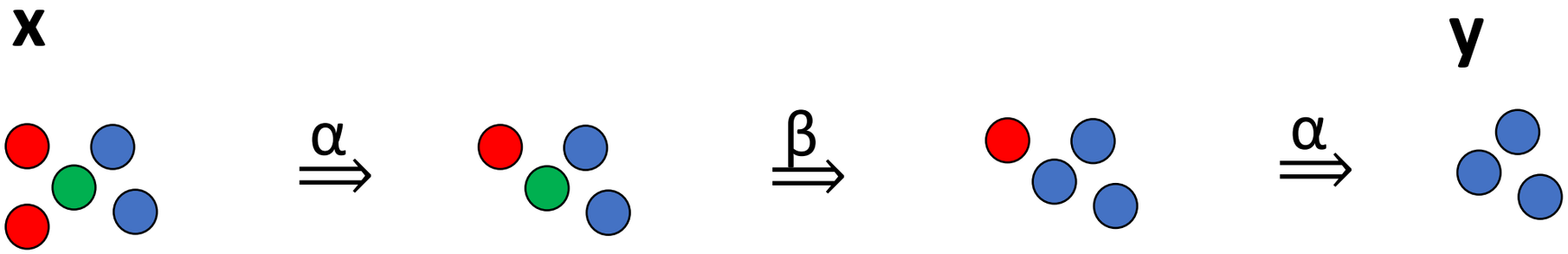
If we can reach from state **x** to **y**, written **x** $\Rightarrow$ **y**, then for all **c** $\in \mathbb{N}^d$, **x**+**c** $\Rightarrow$ **y**+**c**

The presence of extra molecules (represented by **c**) cannot *prevent* reactions from occurring.

Key property of reachability: additivity

If we can reach from state **x** to **y**, written **x** $\Rightarrow$ **y**, then for all **c** $\in \mathbb{N}^d$, **x**+**c** $\Rightarrow$ **y**+**c**

The presence of extra molecules (represented by **c**) cannot *prevent* reactions from occurring.

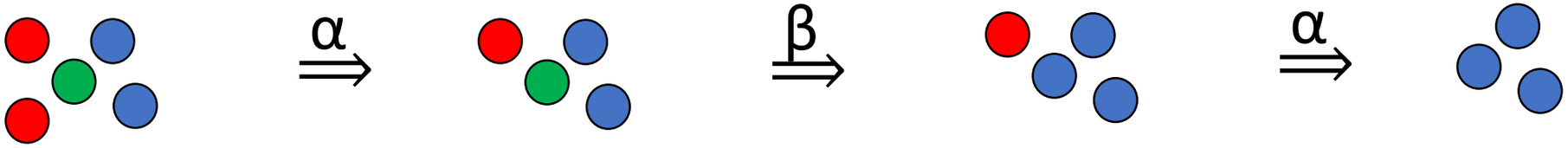


Key property of reachability: additivity

If we can reach from state **x** to **y**, written **x** $\Rightarrow$ **y**, then for all **c** $\in \mathbb{N}^d$, **x**+**c** $\Rightarrow$ **y**+**c**

The presence of extra molecules (represented by **c**) cannot *prevent* reactions from occurring.

c =



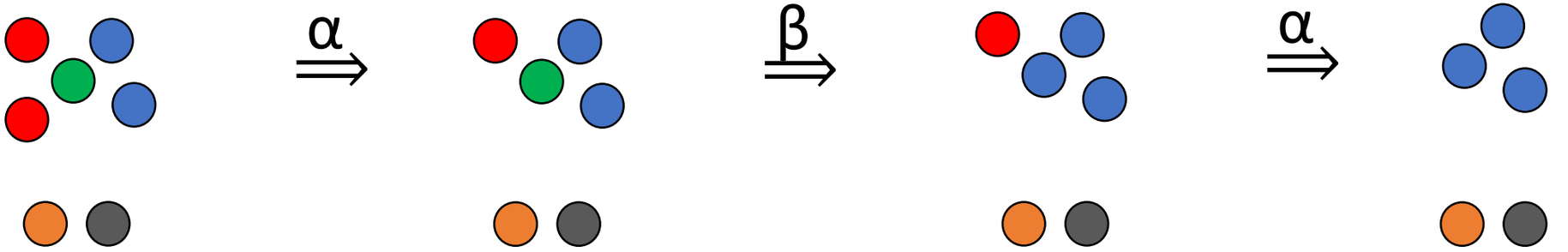
Key property of reachability: additivity

If we can reach from state **x** to **y**, written **x** $\Rightarrow$ **y**, then for all **c** $\in \mathbb{N}^d$, **x+c** $\Rightarrow$ **y+c**

The presence of extra molecules (represented by **c**) cannot *prevent* reactions from occurring.

c =

x + c



y + c

Notation

- For vectors $\mathbf{x},\mathbf{y} \in \mathbb{N}^d$
  - $\mathbf{x} \leqq \mathbf{y}$: $\mathbf{x}(i) \leq \mathbf{y}(i)$ for $1 \leq i \leq d$ $(1,2) \leqq (1,2)$
  - $\mathbf{x} \leq \mathbf{y}$: $\mathbf{x} \leqq \mathbf{y}$ and $\mathbf{x} \neq \mathbf{y}$ $(1,2) \leq (1,4)$
  - $\mathbf{x} < \mathbf{y}$: $\mathbf{x}(i) < \mathbf{y}(i)$ for $1 \leq i \leq d$ $(1,2) < (3,4)$

Notation

- For vectors $\mathbf{x},\mathbf{y} \in \mathbb{N}^d$
  - $\mathbf{x} \leqq \mathbf{y}$: $\mathbf{x}(i) \leq \mathbf{y}(i)$ for $1 \leq i \leq d$ $(1,2) \leqq (1,2)$
  - $\mathbf{x} \leq \mathbf{y}$: $\mathbf{x} \leqq \mathbf{y}$ and $\mathbf{x} \neq \mathbf{y}$ $(1,2) \leq (1,4)$
  - $\mathbf{x} < \mathbf{y}$: $\mathbf{x}(i) < \mathbf{y}(i)$ for $1 \leq i \leq d$ $(1,2) < (3,4)$
  - If $\mathbf{x} \geqq \mathbf{0}$, $\mathbf{x}$ is **nonnegative**. $(0,0)$
  - If $\mathbf{x} \geq \mathbf{0}$, $\mathbf{x}$ is **semipositive**. $(0,1)$
  - If $\mathbf{x} > \mathbf{0}$, $\mathbf{x}$ is **positive**. $(1,1)$

Outline

- Formal definition of chemical reaction networks
- **Execution bounded chemical reaction networks and linear potential functions**
- What is “computation” with chemical reactions?
- Limitations of computation with execution bounded chemical reaction networks
- Possibilities of computation with execution bounded chemical reaction networks

Execution bounded CRNs

- Definition: A CRN *C* is **execution bounded** from state **x** if all executions starting at **x** are finite.

Execution bounded CRNs

- Definition: A CRN *C* is **execution bounded** from state **x** if all executions starting at **x** are finite.
- Why prefer execution bounded CRNs?
  - Wet lab implementations of CRNs use up “fuel” to execute reactions; execution bounded CRNs limit the amount of fuel needed
  - Easier to reason about: as long as reactions keep happening, they make “progress” towards reaching a final state.

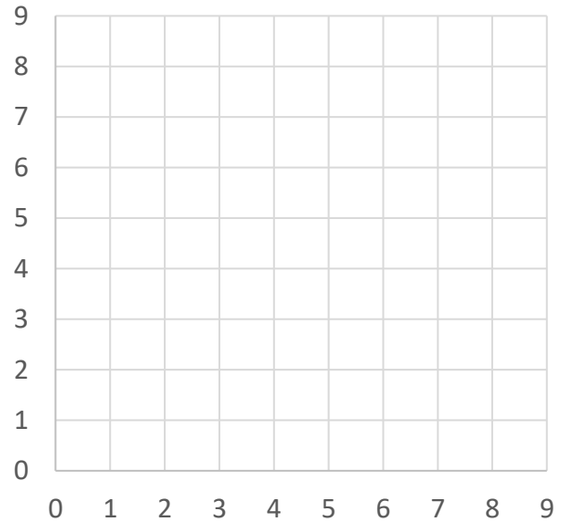
Self-covering executions

Easy Lemma: CRN *C* is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution ($\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, …) that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

Self-covering executions

Easy Lemma: CRN C is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

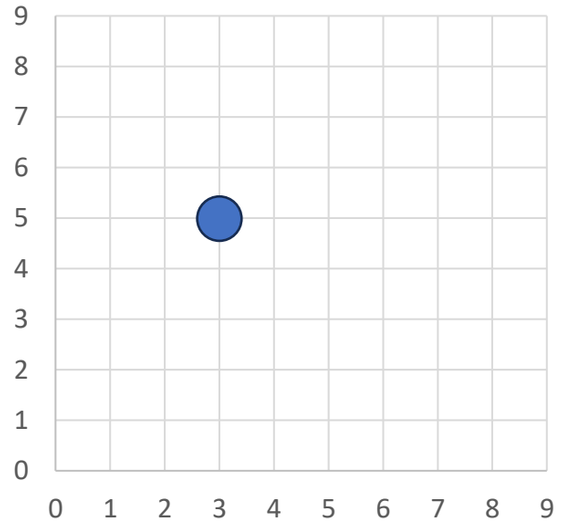
⇒: *Dickson's Lemma*: If $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ is any infinite sequence of vectors from $\mathbb{N}^d$, then for some $i < k$, $\mathbf{x}_i \leqq \mathbf{x}_k$. (easy to show by induction on dimension d) So if C has an infinite execution, it is self-covering.



Self-covering executions

Easy Lemma: CRN *C* is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution ($\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, …) that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

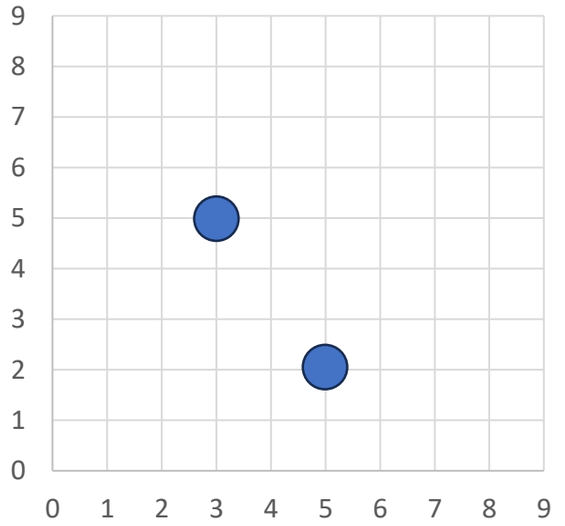
⇒: *Dickson's Lemma*: If ($\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, …) is any infinite sequence of vectors from $\mathbb{N}^d$, then for some $i < k$, $\mathbf{x}_i \leqq \mathbf{x}_k$. (easy to show by induction on dimension *d*) So if *C* has an infinite execution, it is self-covering.



Self-covering executions

Easy Lemma: CRN C is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

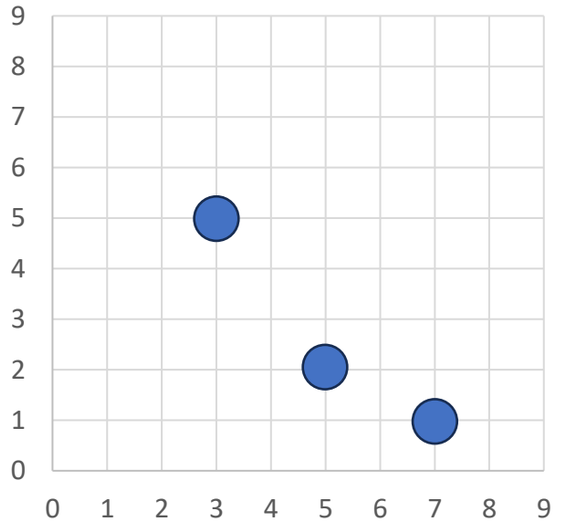
$\Rightarrow$: *Dickson's Lemma*: If $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ is any infinite sequence of vectors from $\mathbb{N}^d$, then for some $i < k$, $\mathbf{x}_i \leqq \mathbf{x}_k$. (easy to show by induction on dimension d) So if C has an infinite execution, it is self-covering.



Self-covering executions

Easy Lemma: CRN C is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

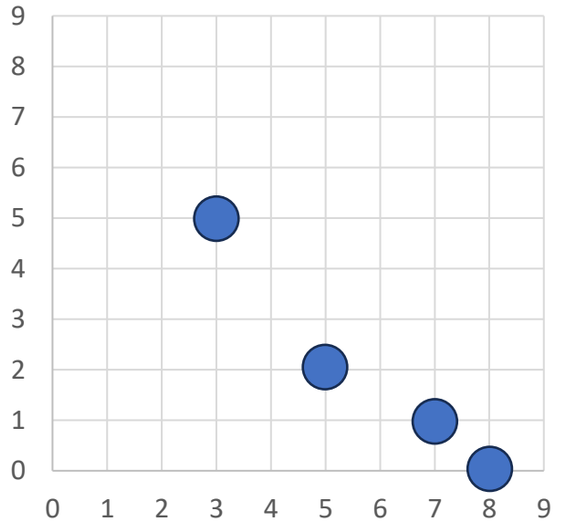
$\Rightarrow$: *Dickson's Lemma*: If $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ is any infinite sequence of vectors from $\mathbb{N}^d$, then for some $i < k$, $\mathbf{x}_i \leqq \mathbf{x}_k$. (easy to show by induction on dimension d) So if C has an infinite execution, it is self-covering.



Self-covering executions

Easy Lemma: CRN C is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

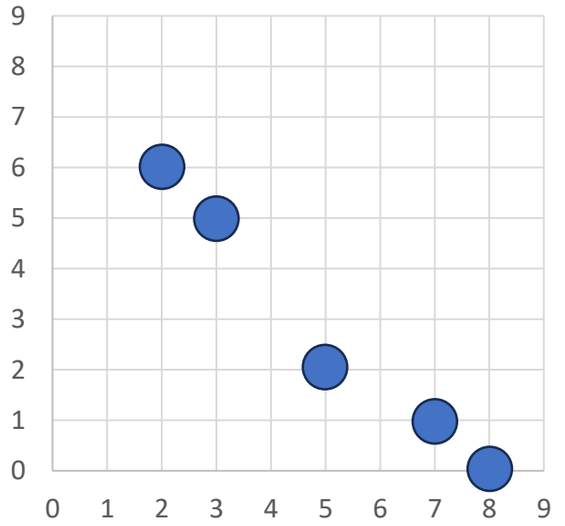
$\Rightarrow$: *Dickson's Lemma*: If $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ is any infinite sequence of vectors from $\mathbb{N}^d$, then for some $i < k$, $\mathbf{x}_i \leqq \mathbf{x}_k$. (easy to show by induction on dimension d) So if C has an infinite execution, it is self-covering.



Self-covering executions

Easy Lemma: CRN C is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

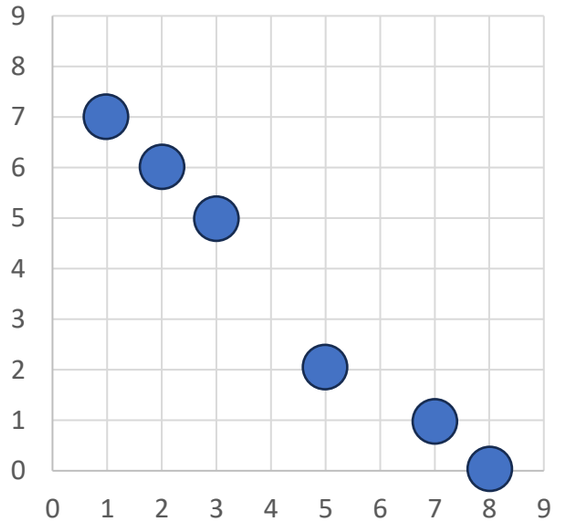
$\Rightarrow$: *Dickson's Lemma*: If $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ is any infinite sequence of vectors from $\mathbb{N}^d$, then for some $i < k$, $\mathbf{x}_i \leqq \mathbf{x}_k$. (easy to show by induction on dimension d) So if C has an infinite execution, it is self-covering.



Self-covering executions

Easy Lemma: CRN *C* is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution ($\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, …) that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

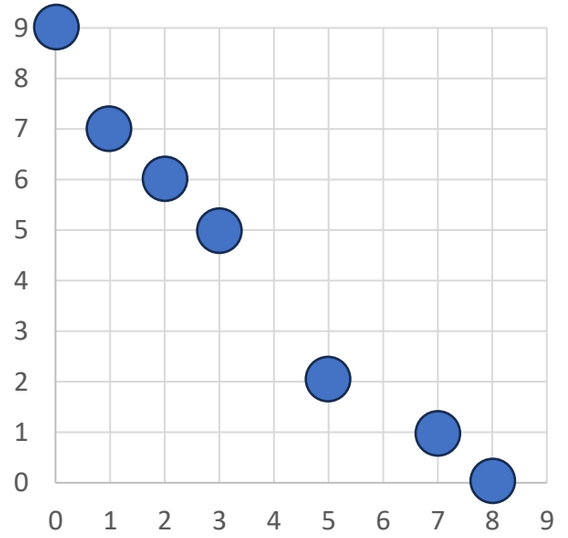
⇒: *Dickson's Lemma*: If ($\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, …) is any infinite sequence of vectors from $\mathbb{N}^d$, then for some $i < k$, $\mathbf{x}_i \leqq \mathbf{x}_k$. (easy to show by induction on dimension *d*) So if *C* has an infinite execution, it is self-covering.



Self-covering executions

Easy Lemma: CRN C is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

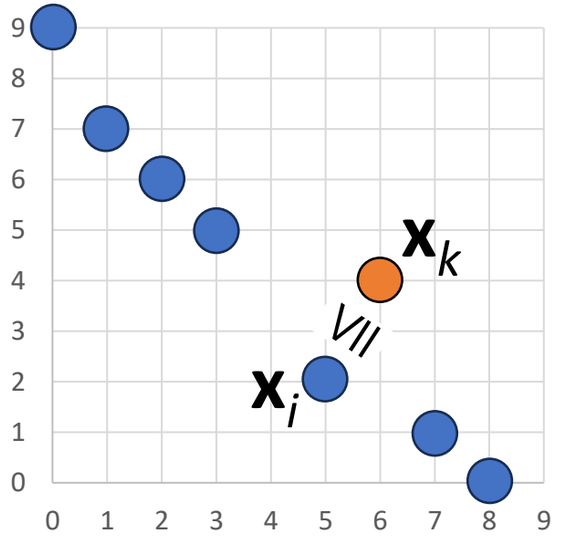
$\Rightarrow$: *Dickson's Lemma*: If $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ is any infinite sequence of vectors from $\mathbb{N}^d$, then for some $i < k$, $\mathbf{x}_i \leqq \mathbf{x}_k$. (easy to show by induction on dimension d) So if C has an infinite execution, it is self-covering.



Self-covering executions

Easy Lemma: CRN *C* is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

⇒: *Dickson's Lemma*: If $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ is any infinite sequence of vectors from $\mathbb{N}^d$, then for some $i < k$, $\mathbf{x}_i \leqq \mathbf{x}_k$. (easy to show by induction on dimension *d*) So if *C* has an infinite execution, it is self-covering.

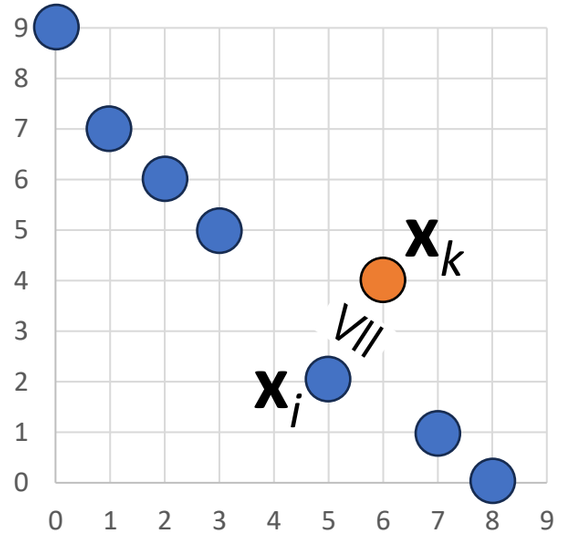


Self-covering executions

Easy Lemma: CRN C is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

$\Rightarrow$: *Dickson's Lemma*: If $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ is any infinite sequence of vectors from $\mathbb{N}^d$, then for some $i < k$, $\mathbf{x}_i \leqq \mathbf{x}_k$. (easy to show by induction on dimension d) So if C has an infinite execution, it is self-covering.

$\Leftarrow$: If an execution is self-covering, by additivity we can repeat indefinitely the reactions leading from $\mathbf{x}_i$ to $\mathbf{x}_k$, so C is not execution bounded from $\mathbf{x}_0$.

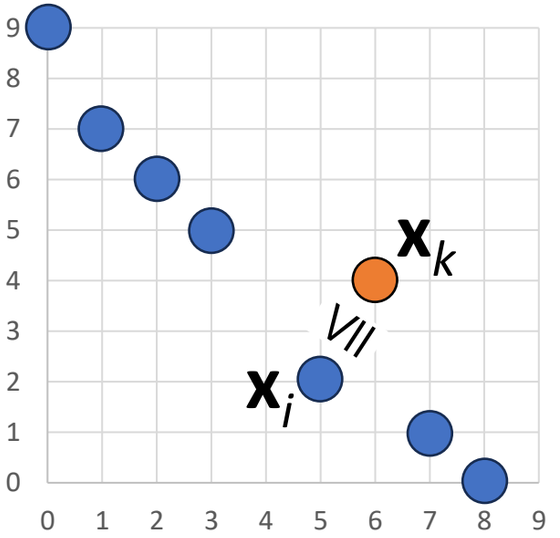


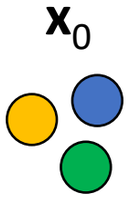
Self-covering executions

Easy Lemma: CRN C is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

⇒: *Dickson's Lemma*: If $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ is any infinite sequence of vectors from $\mathbb{N}^d$, then for some $i < k$, $\mathbf{x}_i \leqq \mathbf{x}_k$. (easy to show by induction on dimension d) So if C has an infinite execution, it is self-covering.

⇐: If an execution is self-covering, by additivity we can repeat indefinitely the reactions leading from $\mathbf{x}_i$ to $\mathbf{x}_k$, so C is not execution bounded from $\mathbf{x}_0$.



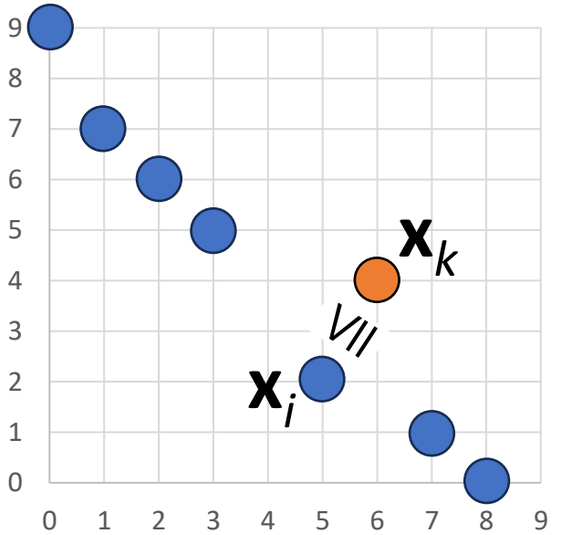


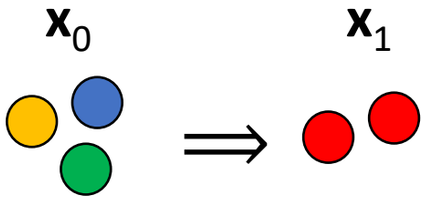
Self-covering executions

Easy Lemma: CRN *C* is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

⇒: *Dickson's Lemma*: If $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ is any infinite sequence of vectors from $\mathbb{N}^d$, then for some $i < k$, $\mathbf{x}_i \leqq \mathbf{x}_k$. (easy to show by induction on dimension *d*) So if *C* has an infinite execution, it is self-covering.

⇐: If an execution is self-covering, by additivity we can repeat indefinitely the reactions leading from $\mathbf{x}_i$ to $\mathbf{x}_k$, so *C* is not execution bounded from $\mathbf{x}_0$.



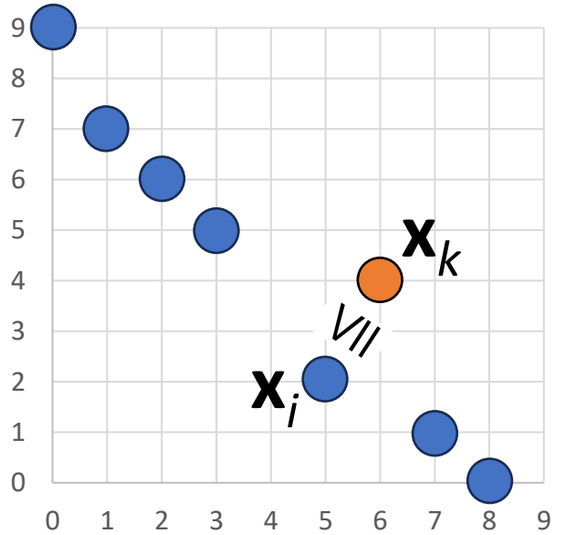


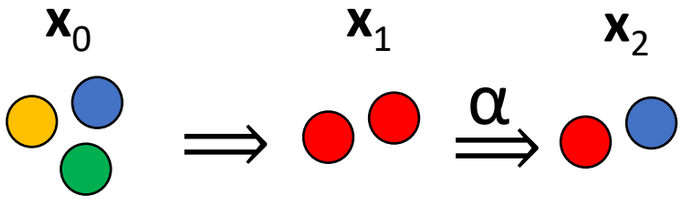
Self-covering executions

Easy Lemma: CRN *C* is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

⇒: *Dickson's Lemma*: If $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ is any infinite sequence of vectors from $\mathbb{N}^d$, then for some $i < k$, $\mathbf{x}_i \leqq \mathbf{x}_k$. (easy to show by induction on dimension *d*) So if *C* has an infinite execution, it is self-covering.

⇐: If an execution is self-covering, by additivity we can repeat indefinitely the reactions leading from $\mathbf{x}_i$ to $\mathbf{x}_k$, so *C* is not execution bounded from $\mathbf{x}_0$.



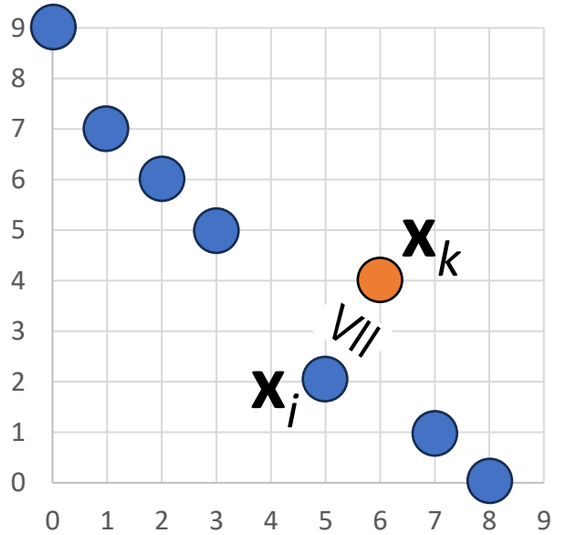


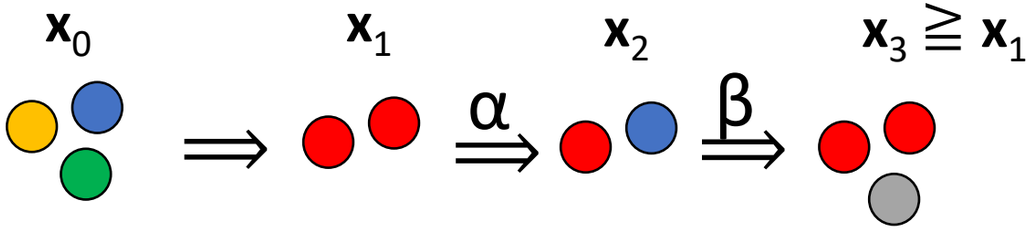
Self-covering executions

Easy Lemma: CRN *C* is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

⇒: *Dickson's Lemma*: If $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ is any infinite sequence of vectors from $\mathbb{N}^d$, then for some $i < k$, $\mathbf{x}_i \leqq \mathbf{x}_k$. (easy to show by induction on dimension *d*) So if *C* has an infinite execution, it is self-covering.

⇐: If an execution is self-covering, by additivity we can repeat indefinitely the reactions leading from $\mathbf{x}_i$ to $\mathbf{x}_k$, so *C* is not execution bounded from $\mathbf{x}_0$.



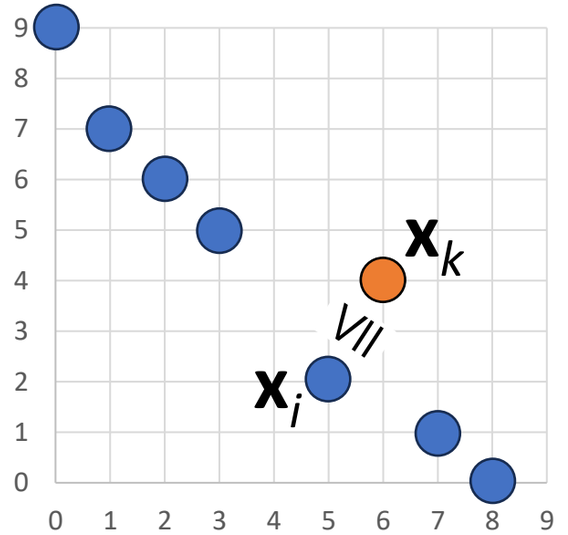


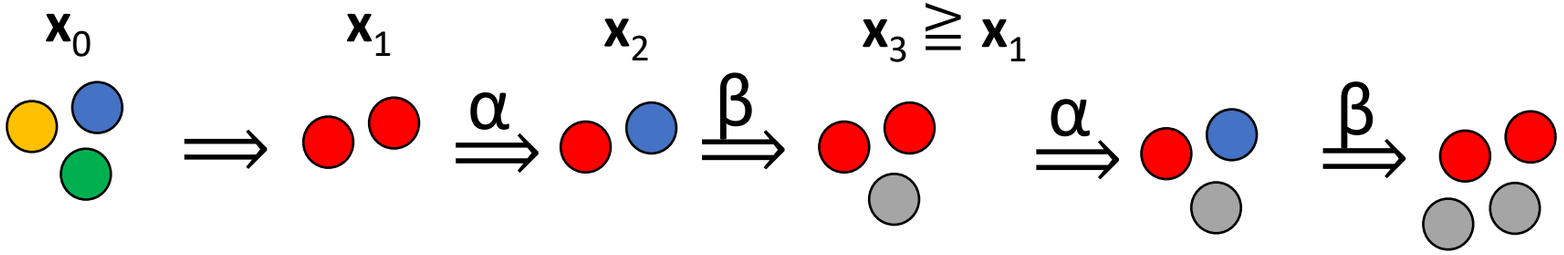
Self-covering executions

Easy Lemma: CRN *C* is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

⇒: *Dickson's Lemma*: If $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ is any infinite sequence of vectors from $\mathbb{N}^d$, then for some $i < k$, $\mathbf{x}_i \leqq \mathbf{x}_k$. (easy to show by induction on dimension *d*) So if *C* has an infinite execution, it is self-covering.

⇐: If an execution is self-covering, by additivity we can repeat indefinitely the reactions leading from $\mathbf{x}_i$ to $\mathbf{x}_k$, so *C* is not execution bounded from $\mathbf{x}_0$.



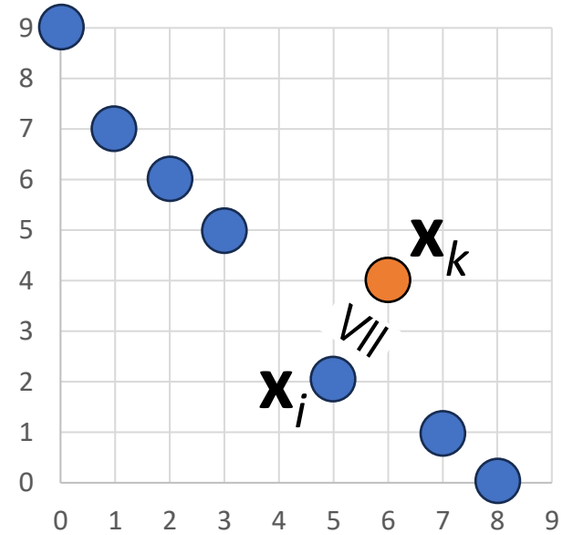


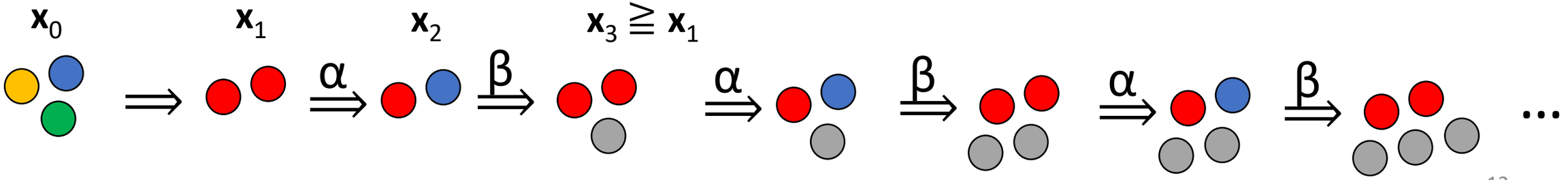
Self-covering executions

Easy Lemma: CRN *C* is not execution bounded from $\mathbf{x}_0$ if and only if there is an execution $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ that is **self-covering**: $\mathbf{x}_i \leqq \mathbf{x}_k$ for some $i < k$.

⇒: *Dickson's Lemma*: If $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$ is any infinite sequence of vectors from $\mathbb{N}^d$, then for some $i < k$, $\mathbf{x}_i \leqq \mathbf{x}_k$. (easy to show by induction on dimension *d*) So if *C* has an infinite execution, it is self-covering.

⇐: If an execution is self-covering, by additivity we can repeat indefinitely the reactions leading from $\mathbf{x}_i$ to $\mathbf{x}_k$, so *C* is not execution bounded from $\mathbf{x}_0$.





Linear potential function

- Definition: Φ: $\mathbb{N}^d \to \mathbb{R}_{\geq 0}$ is a **linear potential function** for a CRN if it is a nonnegative linear function of states that every reaction strictly decreases.

Linear potential function

- Definition: $\Phi: \mathbb{N}^d \to \mathbb{R}_{\geq 0}$ is a **linear potential function** for a CRN if it is a nonnegative linear function of states that every reaction strictly decreases.
- Example:
  - $A+A \to B+C$
  - $B+B \to A$
  - A linear potential function $\Phi(\mathbf{x}) = v_A \cdot \mathbf{x}(A) + v_B \cdot \mathbf{x}(B) + v_C \cdot \mathbf{x}(C)$ must satisfy $2v_A > v_B+v_C$ and $2v_B > v_A$ … $v_A=v_B=1$ and $v_C=0$ works.

Linear potential function

- Definition: $\Phi: \mathbb{N}^d \to \mathbb{R}_{\geq 0}$ is a **linear potential function** for a CRN if it is a nonnegative linear function of states that every reaction strictly decreases.
- Example:
  - $A+A \to B+C$
  - $B+B \to A$
  - A linear potential function $\Phi(\mathbf{x}) = v_A \cdot \mathbf{x}(A) + v_B \cdot \mathbf{x}(B) + v_C \cdot \mathbf{x}(C)$ must satisfy $2v_A > v_B+v_C$ and $2v_B > v_A$ … $v_A=v_B=1$ and $v_C=0$ works.
- A coefficient v_S assigns a nonnegative "mass" to species S, and every reaction removes a positive amount of mass from the system.

Linear potential function

- Definition: $\Phi: \mathbb{N}^d \to \mathbb{R}_{\geq 0}$ is a **linear potential function** for a CRN if it is a nonnegative linear function of states that every reaction strictly decreases.
- Example:
  - $A+A \to B+C$
  - $B+B \to A$
  - A linear potential function $\Phi(\mathbf{x}) = v_A \cdot \mathbf{x}(A) + v_B \cdot \mathbf{x}(B) + v_C \cdot \mathbf{x}(C)$ must satisfy $2v_A > v_B+v_C$ and $2v_B > v_A$ … $v_A=v_B=1$ and $v_C=0$ works.
- A coefficient v_S assigns a nonnegative "mass" to species S, and every reaction removes a positive amount of mass from the system.
- By clearing denominators, we can assume each v_S is an integer, so each reaction decreases Φ by at least 1.

Linear potential functions characterize execution bounded CRNs

Theorem: A CRN has a linear potential function if and only if it is execution bounded from every state.

Linear potential functions characterize execution bounded CRNs

Theorem: A CRN has a linear potential function if and only if it is execution bounded from every state.

Forward direction is easy: Since each reaction reduces Φ by at least 1, from any state $\mathbf{x}$, at most $\Phi(\mathbf{x})$ reactions are possible.

Key technical tool for reverse direction

Theorem: (Gale 1960) "*Theorem of the Alternative*" (similar to Farkas' Lemma): Let **M** be a matrix. Then exactly one of the following statements is true:

1. There is a vector $\mathbf{u} \geq \mathbf{0}$ such that $\mathbf{Mu} \geqq \mathbf{0}$.
2. There is a vector $\mathbf{v} \geq \mathbf{0}$ such that $\mathbf{vM} < \mathbf{0}$.

[David Gale. The Theory of Linear Economic Models. University of Chicago press, 1960.]

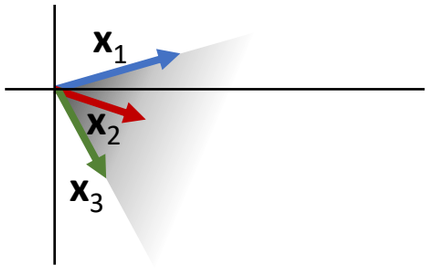
Key technical tool for reverse direction

Theorem: (Gale 1960) "*Theorem of the Alternative*" (similar to Farkas' Lemma): Let **M** be a matrix. Then exactly one of the following statements is true:

1. There is a vector $\mathbf{u} \geq \mathbf{0}$ such that $\mathbf{Mu} \geqq \mathbf{0}$.
2. There is a vector $\mathbf{v} \geq \mathbf{0}$ such that $\mathbf{vM} < \mathbf{0}$.

[David Gale. The Theory of Linear Economic Models. University of Chicago press, 1960.]

1. Either the cone of **M**'s column vectors intersects the nonnegative orthant:



$$\mathbf{M} = [\mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3]$$

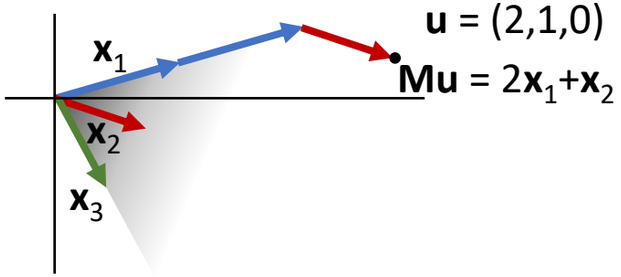
Key technical tool for reverse direction

Theorem: (Gale 1960) "*Theorem of the Alternative*" (similar to Farkas' Lemma): Let **M** be a matrix. Then exactly one of the following statements is true:

1. There is a vector $\mathbf{u} \geq \mathbf{0}$ such that $\mathbf{Mu} \geqq \mathbf{0}$.
2. There is a vector $\mathbf{v} \geq \mathbf{0}$ such that $\mathbf{vM} < \mathbf{0}$.

[David Gale. The Theory of Linear Economic Models. University of Chicago press, 1960.]

1. Either the cone of **M**'s column vectors intersects the nonnegative orthant:



$$\mathbf{M} = [\mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3]$$

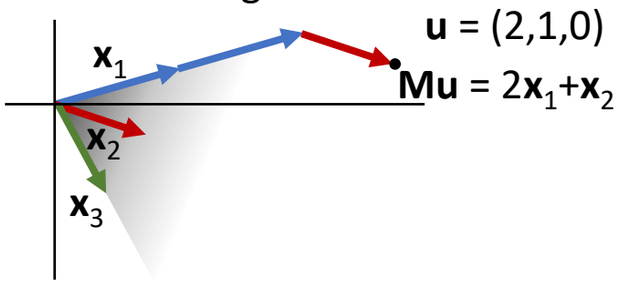
Key technical tool for reverse direction

Theorem: (Gale 1960) "*Theorem of the Alternative*" (similar to Farkas' Lemma): Let **M** be a matrix. Then exactly one of the following statements is true:

1. There is a vector $\mathbf{u} \geq \mathbf{0}$ such that $\mathbf{Mu} \geqq \mathbf{0}$.
2. There is a vector $\mathbf{v} \geq \mathbf{0}$ such that $\mathbf{vM} < \mathbf{0}$.

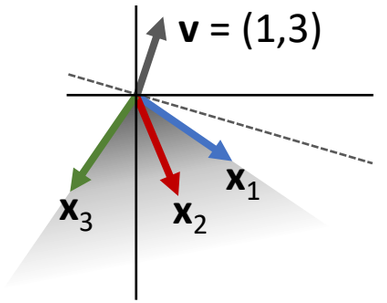
[David Gale. The Theory of Linear Economic Models. University of Chicago press, 1960.]

1. Either the cone of **M**'s column vectors intersects the nonnegative orthant:



$\mathbf{M} = [\mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3]$

2. Or it doesn't, and then some hyperplane (dashed line) separates that cone from the nonnegative orthant:



$\mathbf{vM} < \mathbf{0} \Rightarrow$
$(\forall i)\ \mathbf{v}\cdot\mathbf{x}_i < 0$

CRN is execution bounded from every state ⇒ it has a linear potential function

CRN is execution bounded from every state ⇒ it has a linear potential function

Let **M** be the *stoichiometric matrix*, e.g.

α: $A \rightarrow B + 2C$

β: $3B + C \rightarrow A + B + C$

$$\mathbf{M} = \begin{array}{c} \begin{matrix} \alpha & \beta \end{matrix} \\ \begin{pmatrix} -1 & 1 \\ 1 & -2 \\ 2 & 0 \end{pmatrix} \end{array} \begin{matrix} A \\ B \\ C \end{matrix}$$

CRN is execution bounded from every state ⇒ it has a linear potential function

Let **M** be the *stoichiometric matrix*, e.g.

α: $A \to B + 2C$

β: $3B + C \to A + B + C$

$$\mathbf{M} = \begin{array}{c} \begin{matrix} \alpha & \beta \end{matrix} \\ \begin{pmatrix} -1 & 1 \\ 1 & -2 \\ 2 & 0 \end{pmatrix} \end{array} \begin{matrix} A \\ B \\ C \end{matrix}$$

If **u** = (2,1) is a vector indicating "*do reaction* α *twice and reaction* β *once*", then the vector **Mu** = (–1,0,4) indicates how species counts change.

CRN is execution bounded from every state ⇒ it has a linear potential function

Let **M** be the *stoichiometric matrix*, e.g.

$$\alpha: \quad A \to B + 2C$$

$$\beta: \quad 3B + C \to A + B + C$$

$$\mathbf{M} = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{cc} \alpha & \beta \end{array} \\ \begin{pmatrix} -1 & 1 \\ 1 & -2 \\ 2 & 0 \end{pmatrix} \end{array}$$

If **u** = (2,1) is a vector indicating "*do reaction* α *twice and reaction* β *once*", then the vector **Mu** = (–1,0,4) indicates how species counts change.

Claim: There is no **u** ≥ **0** such that **Mu** ≧ **0**; suppose otherwise. Then from any sufficiently large state **x**, we can execute reactions in **u**, reaching from **x** to **y** = **x** + **Mu**, where **y** ≧ **x**, i.e., a self-covering execution, not possible since the CRN is execution bounded from **x**.

CRN is execution bounded from every state ⇒ it has a linear potential function

Let **M** be the *stoichiometric matrix*, e.g.

α: $A \to B + 2C$

β: $3B + C \to A + B + C$

$$\mathbf{M} = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{cc} \alpha & \beta \end{array} \\ \begin{pmatrix} -1 & 1 \\ 1 & -2 \\ 2 & 0 \end{pmatrix} \end{array}$$

If **u** = (2,1) is a vector indicating "*do reaction* α *twice and reaction* β *once*", then the vector **Mu** = (–1,0,4) indicates how species counts change.

Claim: There is no **u** ≥ **0** such that **Mu** ≧ **0**; suppose otherwise. Then from any sufficiently large state **x**, we can execute reactions in **u**, reaching from **x** to **y** = **x** + **Mu**, where **y** ≧ **x**, i.e., a self-covering execution, not possible since the CRN is execution bounded from **x**.

Then there is a vector **v** ≥ **0** such that **vM** < **0**. Let **v** be the coefficients of a linear function Φ(**x**) = **v**·**x**. Then **vM** < **0** means each reaction decreases Φ: it is a linear potential function. QED

Outline

- Formal definition of chemical reaction networks
- Execution bounded chemical reaction networks and linear potential functions
- **What is “computation” with chemical reactions?**
- Limitations of computation with execution bounded chemical reaction networks
- Possibilities of computation with execution bounded chemical reaction networks

Can we compute with chemistry?

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

Can we compute with chemistry?

"Not every chemical reaction network describes real chemicals!", i.e. "where's the *compiler*?"

Response: Soloveichik et al. showed how to physically implement any chemical reaction network using *DNA strand displacement*

[Soloveichik, Seelig, Winfree, DNA as a Universal Substrate for Chemical Kinetics, *PNAS* 2010]

Can we compute with chemistry?

"Not every chemical reaction network describes real chemicals!", i.e. "where's the *compiler*?"

Response: Soloveichik et al. showed how to physically implement <u>any</u> chemical reaction network using *DNA strand displacement*

$$X_1 + X_2 \rightarrow X_3$$

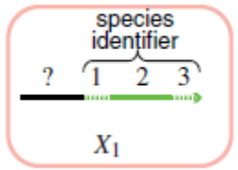
[Soloveichik, Seelig, Winfree, DNA as a Universal Substrate for Chemical Kinetics, *PNAS* 2010]

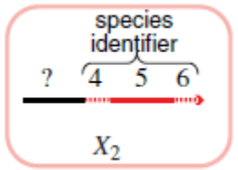
Can we compute with chemistry?

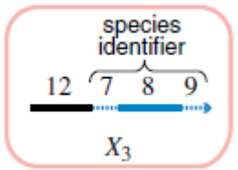
"Not every chemical reaction network describes real chemicals!", i.e. "where's the *compiler*?"

Response: Soloveichik et al. showed how to physically implement <u>any</u> chemical reaction network using *DNA strand displacement*

$$X_1 + X_2 \rightarrow X_3$$







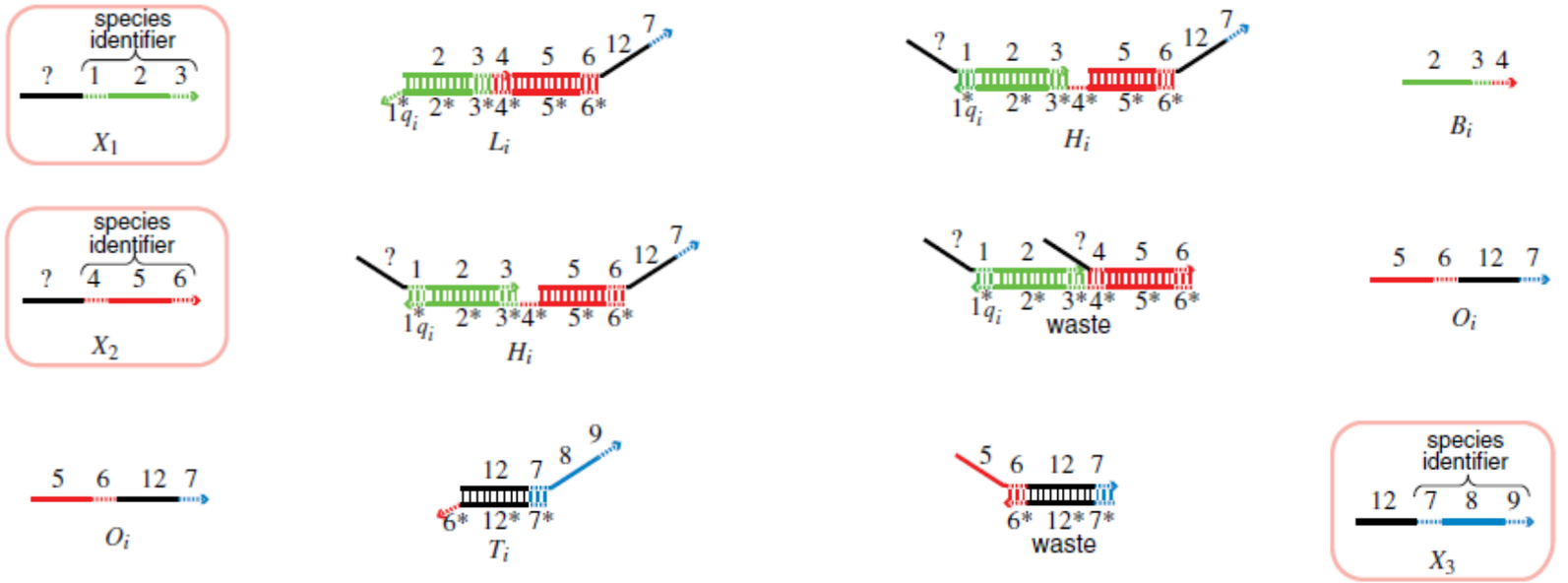
[Soloveichik, Seelig, Winfree, DNA as a Universal Substrate for Chemical Kinetics, *PNAS* 2010]

Can we compute with chemistry?

"Not every chemical reaction network describes real chemicals!", i.e. "where's the *compiler*?"

Response: Soloveichik et al. showed how to physically implement <u>any</u> chemical reaction network using *DNA strand displacement*

$$X_1 + X_2 \rightarrow X_3$$



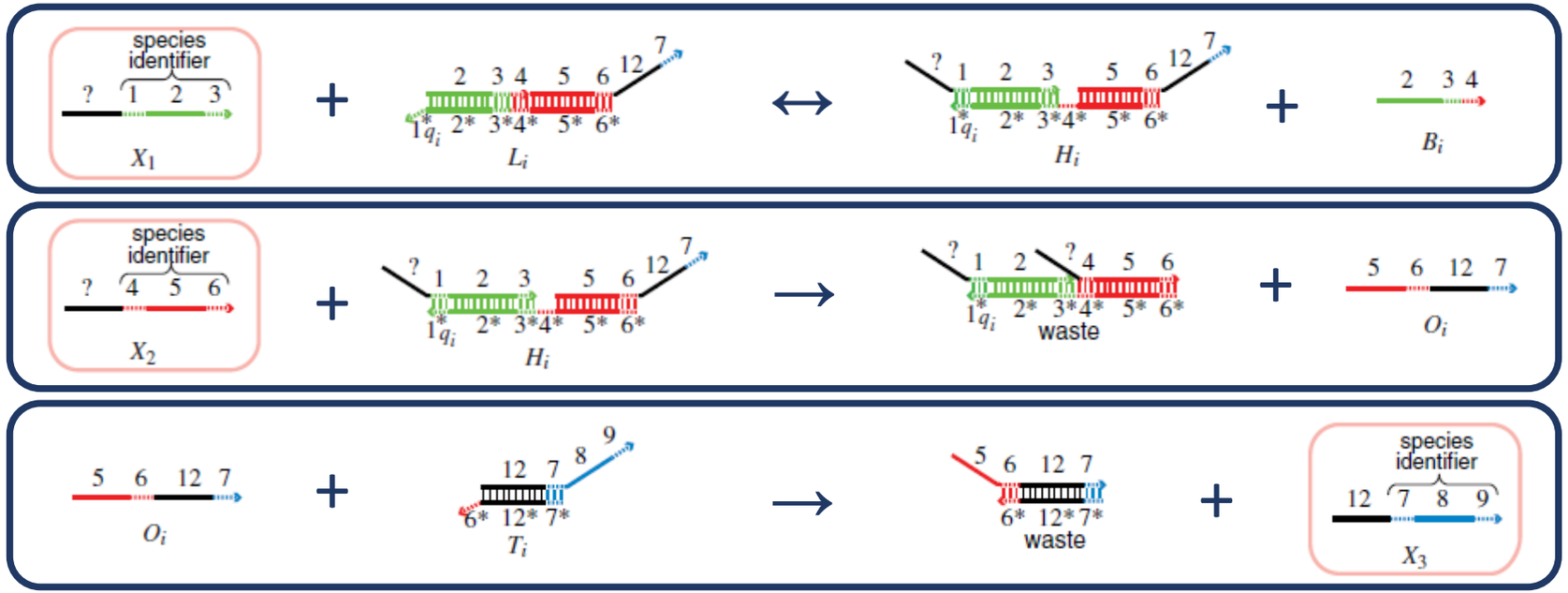
[Soloveichik, Seelig, Winfree, DNA as a Universal Substrate for Chemical Kinetics, *PNAS* 2010]

Can we compute with chemistry?

"Not every chemical reaction network describes real chemicals!", i.e. "where's the *compiler*?"

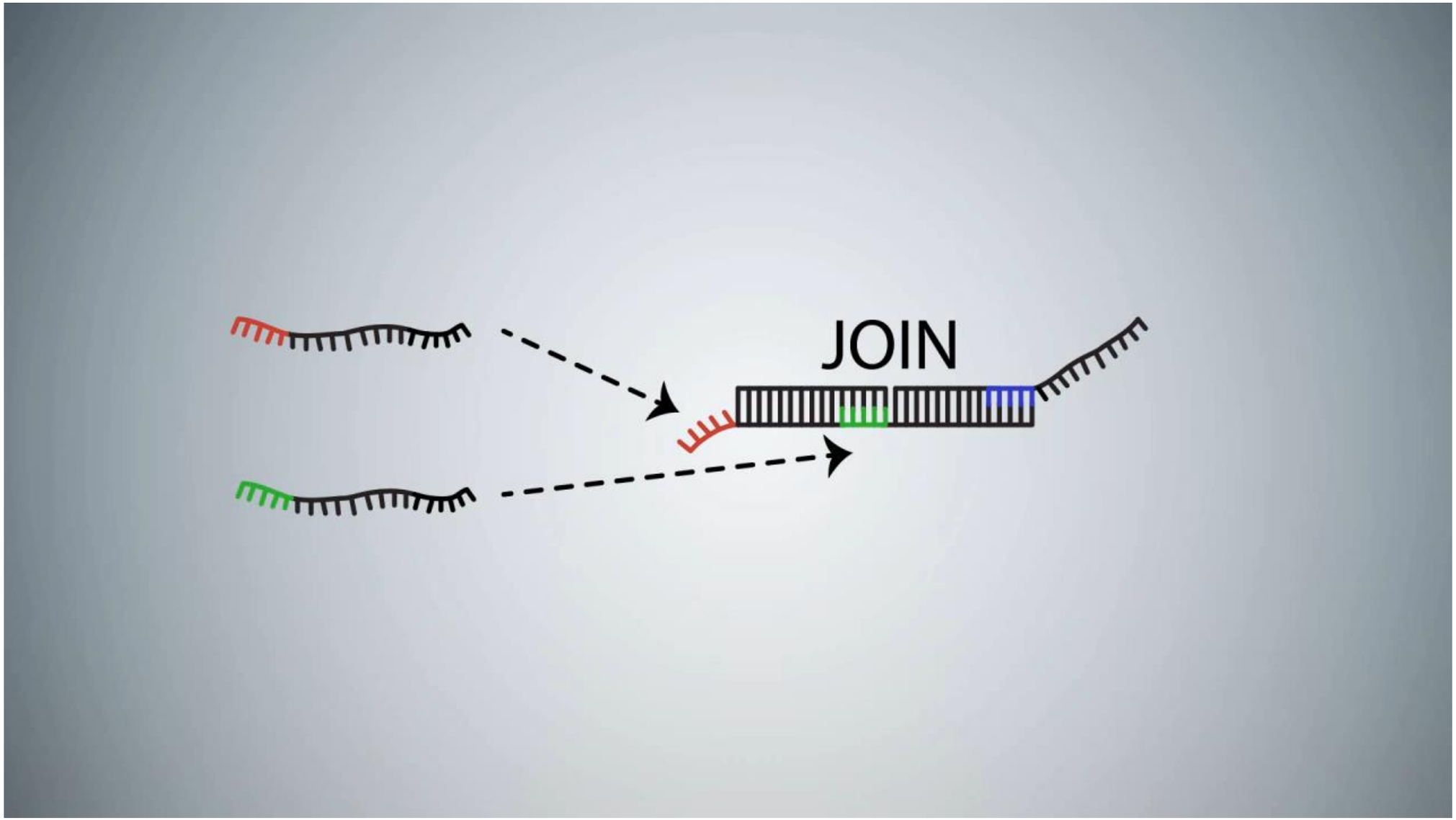
Response: Soloveichik et al. showed how to physically implement <u>any</u> chemical reaction network using *DNA strand displacement*

$$X_1 + X_2 \rightarrow X_3$$



[Soloveichik, Seelig, Winfree, DNA as a Universal Substrate for Chemical Kinetics, *PNAS* 2010]

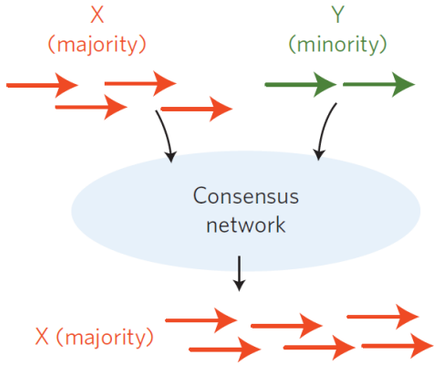
DNA strand displacement implementing A+B → C



video: Microsoft Research Cambridge

Experimental implementations of synthetic chemical reaction networks with DNA

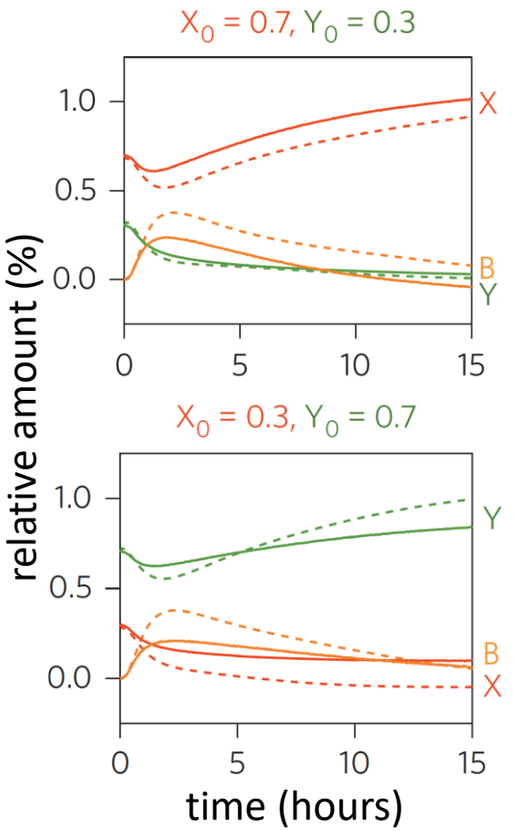
Analog majority computation



$X+Y \rightarrow B+B$

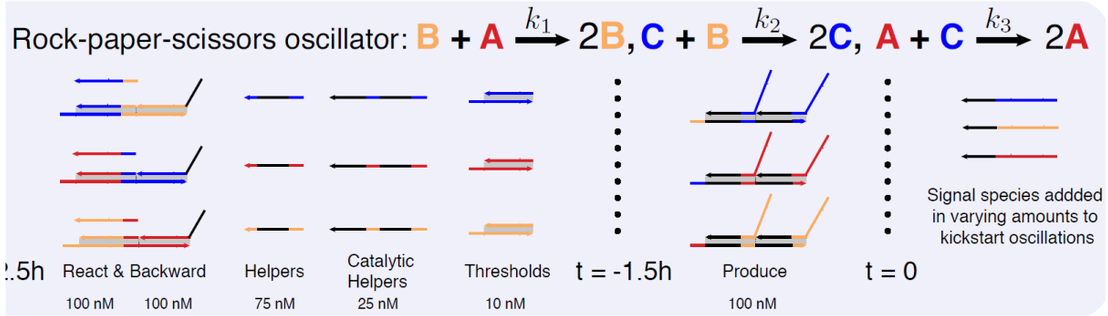
$X+B \rightarrow X+X$

$Y+B \rightarrow Y+Y$



[*Programmable chemical controllers made from DNA.* Chen, Dalchau, Srinivas, Phillips, Cardelli, Soloveichik, Seelig, Nature Nanotechnology 2013.]

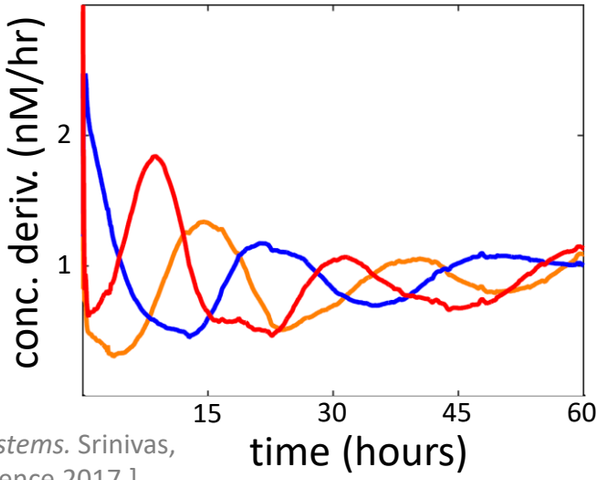
Chemical oscillator



$A+B \rightarrow B+B$

$B+C \rightarrow C+C$

$C+A \rightarrow A+A$

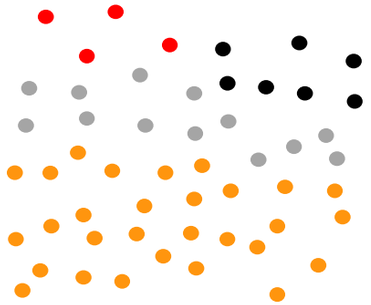


[*Enzyme-free nucleic acid dynamical systems.* Srinivas, Parkin, Seelig, Winfree, Soloveichik, Science 2017.]

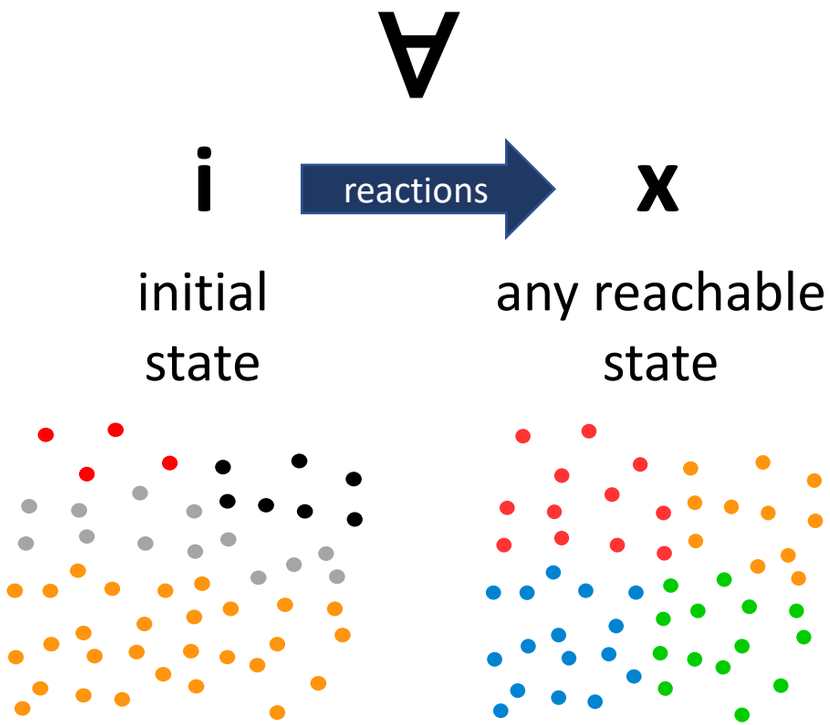
Defining **stable** computation

i

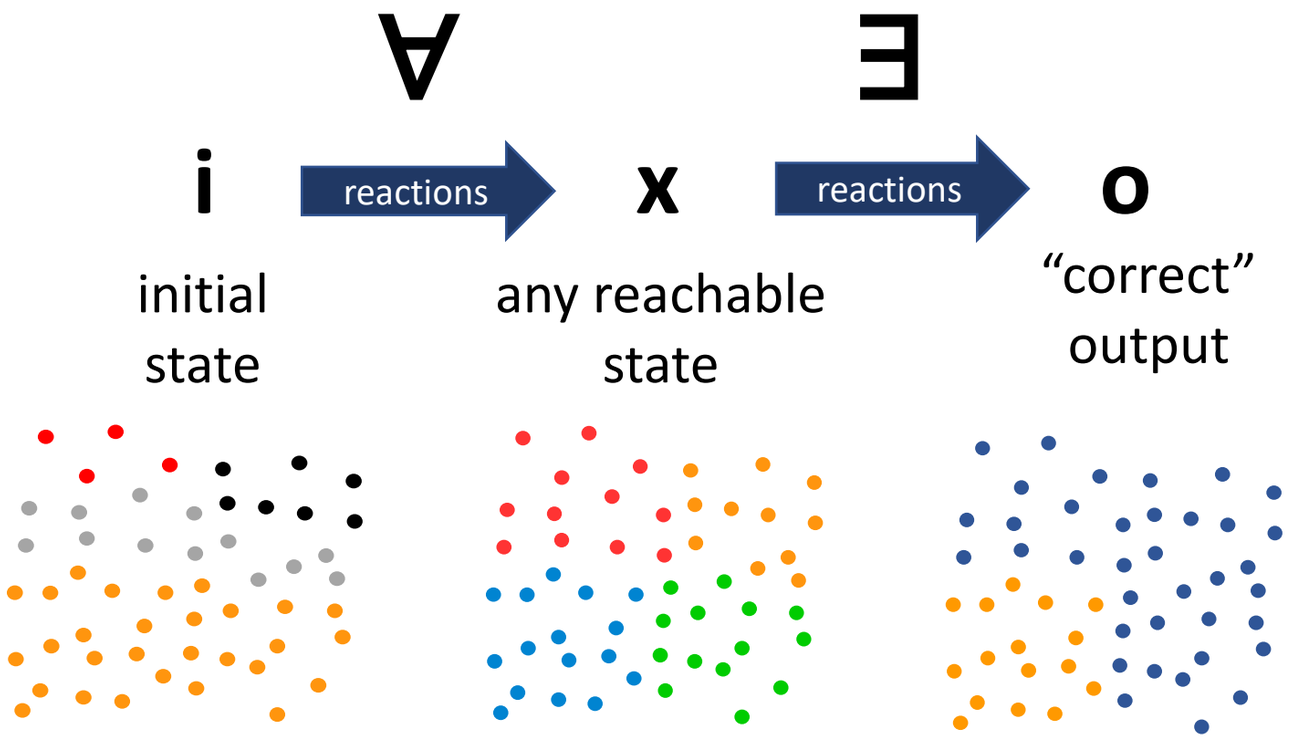
initial
state



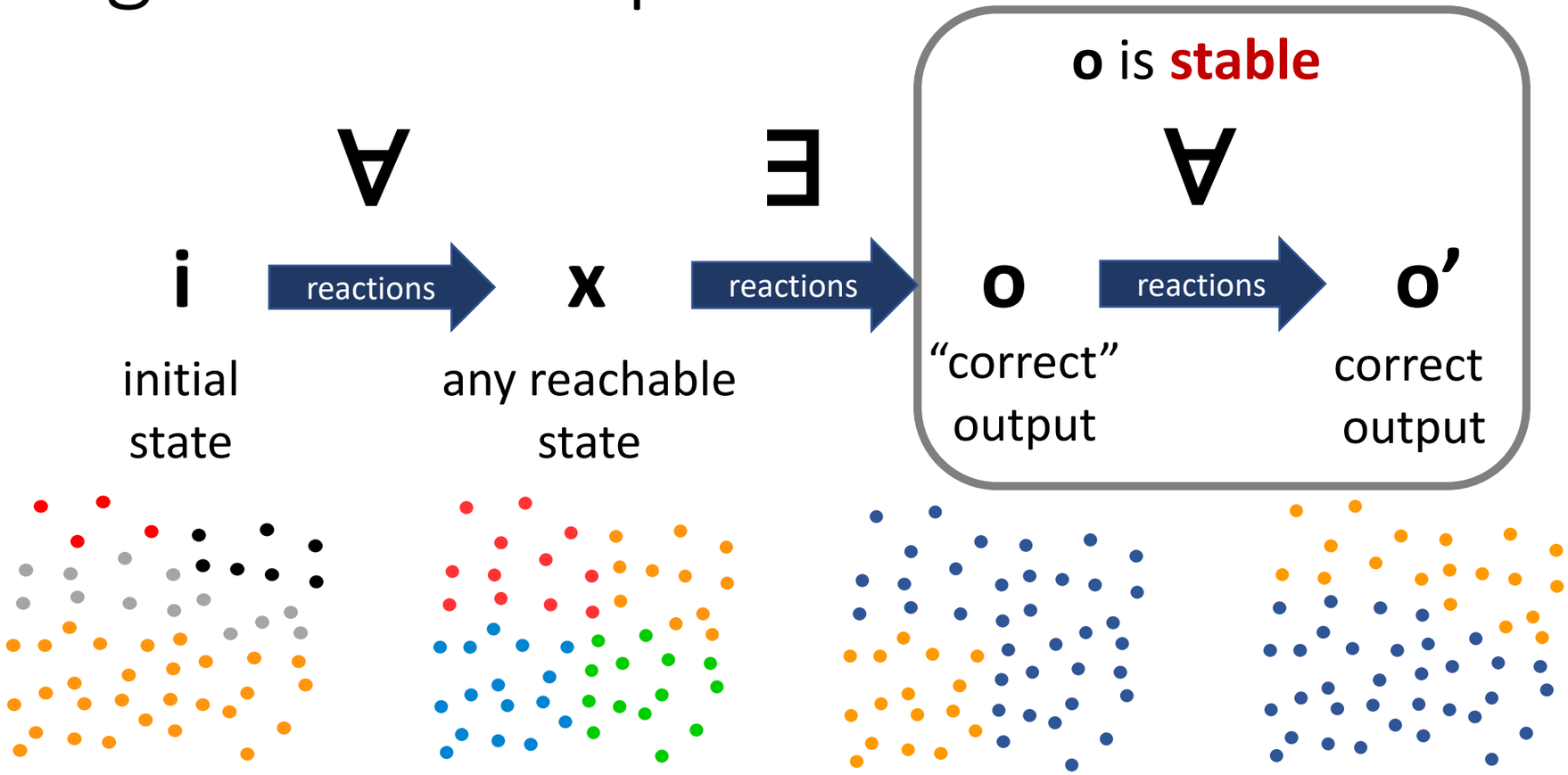
Defining **stable** computation



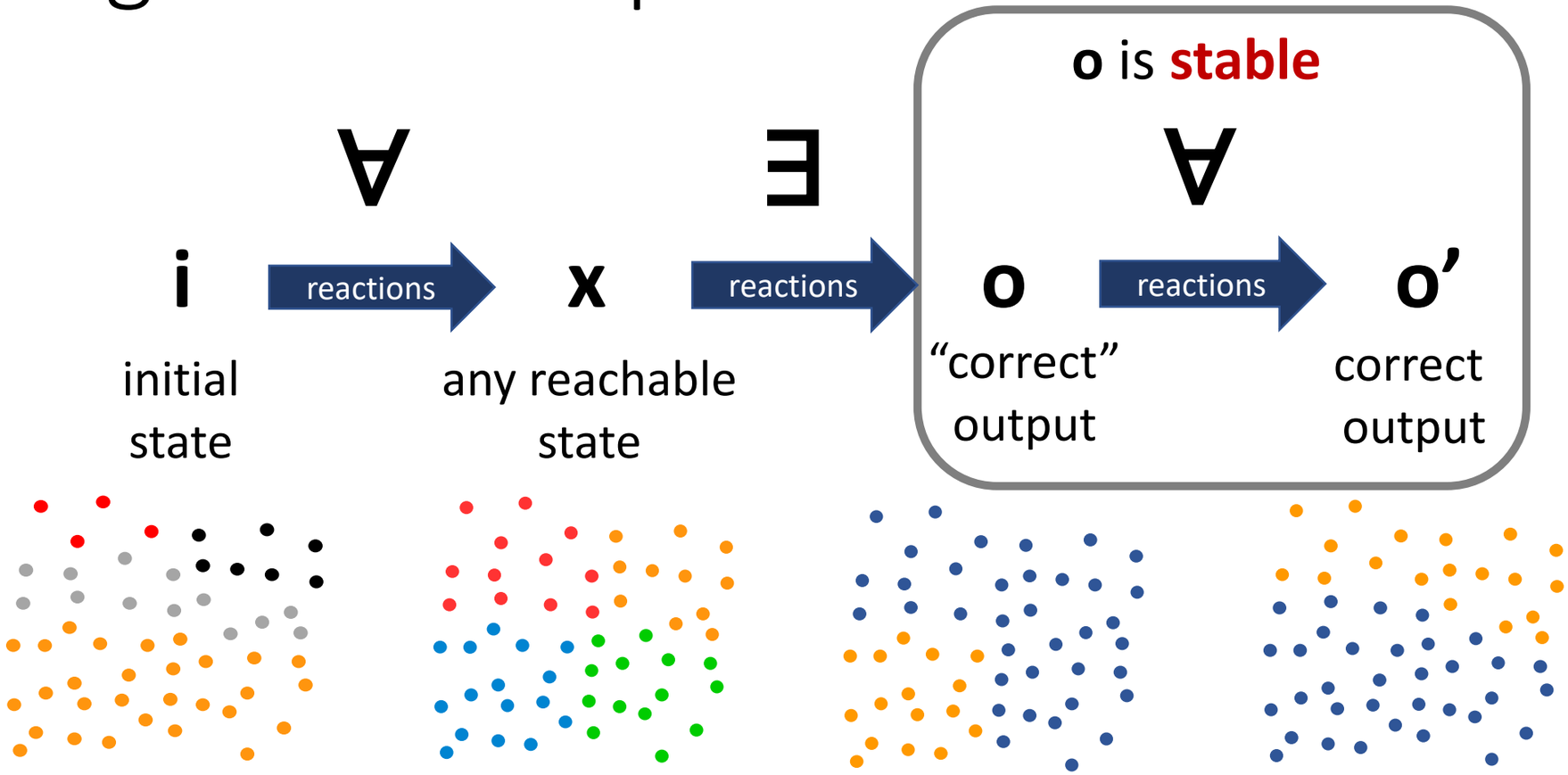
Defining **stable** computation



Defining **stable** computation



Defining **stable** computation



(assuming finite set of reachable states) equivalent to:
The system will reach the correct output with probability 1.

Definition of *predicate* (decision problem) computation

- goal: compute predicate φ: $\mathbb{N}^k \to \{Y,N\}$, e.g., $\varphi(a,b)=Y \iff a \geq b$

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

Definition of *predicate* (decision problem) computation

- goal: compute predicate φ: $\mathbb{N}^k \to$ {Y,N}, e.g., $\varphi(a,b)$=Y $\Leftrightarrow$ $a \geq b$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - in valid initial states all molecules are from Σ, e.g., {100 *A*, 55 *B*}

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

Definition of *predicate* (decision problem) computation

- goal: compute predicate φ: $\mathbb{N}^k \to$ {Y,N}, e.g., $\varphi(a,b)$=Y $\Leftrightarrow$ $a \geq b$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - in valid initial states all molecules are from Σ, e.g., {100 *A*, 55 *B*}
- output specification: partition species Λ into "yes" voters Λ_Y and "no" voters Λ_N

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

Definition of *predicate* (decision problem) computation

- goal: compute predicate φ: $\mathbb{N}^k \to$ {Y,N}, e.g., $\varphi(a,b)$=Y $\Leftrightarrow$ $a \geq b$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - in valid initial states all molecules are from Σ, e.g., {100 *A*, 55 *B*}
- output specification: partition species Λ into "yes" voters Λ_Y and "no" voters Λ_N
  - $\psi(\mathbf{o})$ = Y (state **o** outputs "yes") if vote is unanimously yes: $\mathbf{o}(S)>0 \Leftrightarrow S \in \Lambda_Y$
  - $\psi(\mathbf{o})$ = N (state **o** outputs "no") if vote is unanimously no: $\mathbf{o}(S)>0 \Leftrightarrow S \in \Lambda_N$
  - state **o** has undefined output otherwise: $(\exists\, S \in \Lambda_N, S' \in \Lambda_Y)$ $\mathbf{o}(S)>0$ and $\mathbf{o}(S')>0$

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

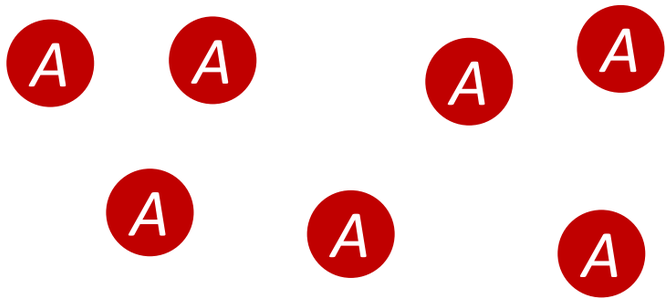
Definition of *predicate* (decision problem) computation

- goal: compute predicate φ: $\mathbb{N}^k \to \{Y,N\}$, e.g., $\varphi(a,b)=Y \iff a \geq b$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - in valid initial states all molecules are from Σ, e.g., {100 *A*, 55 *B*}
- output specification: partition species Λ into "yes" voters Λ_Y and "no" voters Λ_N
  - $\psi(\mathbf{o}) = Y$ (state **o** outputs "yes") if vote is unanimously yes: $\mathbf{o}(S)>0 \iff S \in \Lambda_Y$
  - $\psi(\mathbf{o}) = N$ (state **o** outputs "no") if vote is unanimously no: $\mathbf{o}(S)>0 \iff S \in \Lambda_N$
  - state **o** has undefined output otherwise: $(\exists\, S \in \Lambda_N, S' \in \Lambda_Y)\ \mathbf{o}(S)>0$ and $\mathbf{o}(S')>0$
- **o** is stable if $\psi(\mathbf{o}) = \psi(\mathbf{o'})$ for all **o'** reachable from **o**

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

Examples of predicate computation

Detection: $\varphi(a,b) = \text{Y} \Leftrightarrow b > 0$

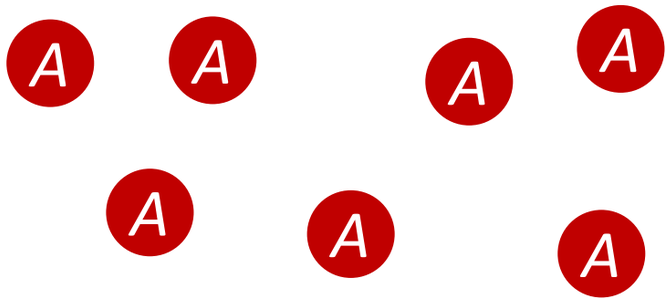


Examples of predicate computation

Detection: $\varphi(a,b) = Y \Leftrightarrow b > 0$

$$B+A \rightarrow B+B$$

A votes no; *B* votes yes

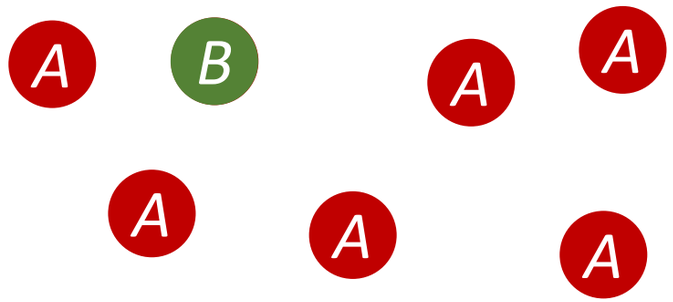


Examples of predicate computation

Detection: $\varphi(a,b) = Y \Leftrightarrow b > 0$

$$B+A \rightarrow B+B$$

A votes no; *B* votes yes

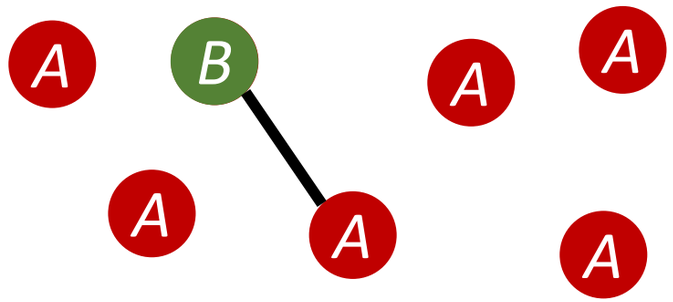


Examples of predicate computation

Detection: $\varphi(a,b) = Y \Leftrightarrow b > 0$

$$B+A \rightarrow B+B$$

A votes no; *B* votes yes

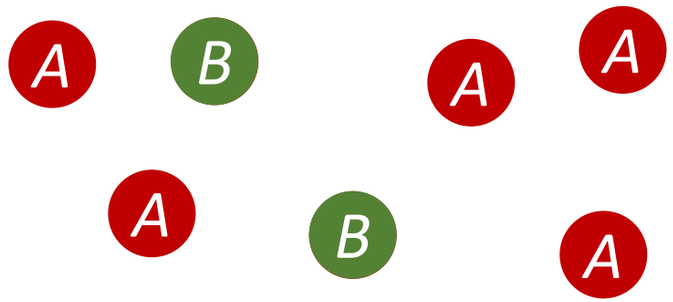


Examples of predicate computation

Detection: $\varphi(a,b) = Y \Leftrightarrow b > 0$

$$B+A \rightarrow B+B$$

A votes no; *B* votes yes

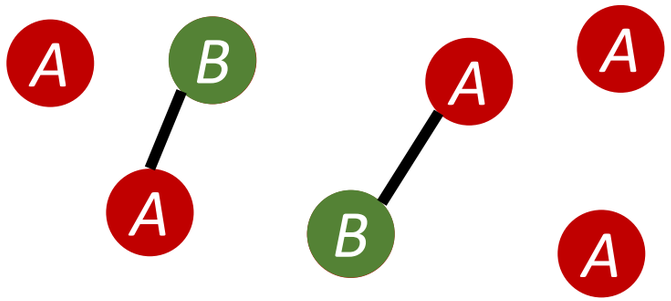


Examples of predicate computation

Detection: $\varphi(a,b) = Y \Leftrightarrow b > 0$

$$B+A \rightarrow B+B$$

A votes no; *B* votes yes

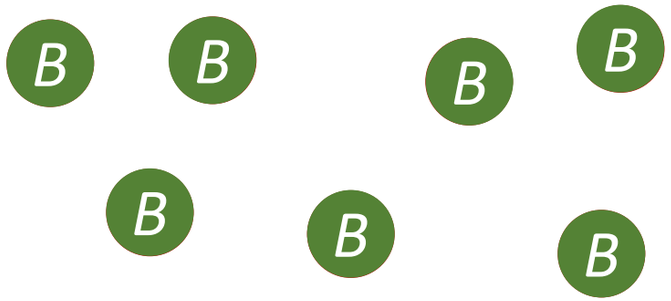


Examples of predicate computation

Detection: $\varphi(a,b) = Y \Leftrightarrow b > 0$

$$B+A \rightarrow B+B$$

A votes no; *B* votes yes



Examples of predicate computation

Parity: $\varphi(a)$=Y $\Leftrightarrow$ a is odd

Examples of predicate computation

Parity: $\varphi(a)$=Y ⇔ a is odd

input species A_o (subscript o/e means ODD/EVEN, and capital *A* means it is leader)

Examples of predicate computation

Parity: $\varphi(a)$=Y $\Leftrightarrow$ a is odd

input species A_o (subscript o/e means ODD/EVEN, and capital *A* means it is leader)

$A_o + A_o \rightarrow A_e + a_e$

$A_e + A_e \rightarrow A_e + a_e$

$A_o + A_e \rightarrow A_o + a_o$

two leaders XOR their parity, and one becomes follower

Examples of predicate computation

Parity: $\varphi(a)$=Y $\Leftrightarrow$ a is odd

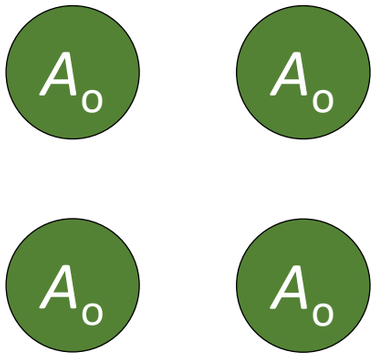
input species A_o (subscript o/e means ODD/EVEN, and capital A means it is leader)

$A_o + A_o \rightarrow A_e + a_e$

$A_e + A_e \rightarrow A_e + a_e$

$A_o + A_e \rightarrow A_o + a_o$

two leaders XOR their parity, and one becomes follower



Examples of predicate computation

Parity: $\varphi(a)$=Y $\Leftrightarrow$ a is odd

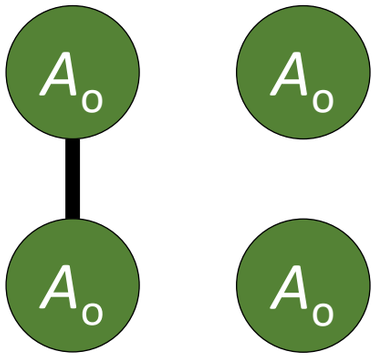
input species A_o (subscript o/e means ODD/EVEN, and capital A means it is leader)

$A_o + A_o \rightarrow A_e + a_e$

$A_e + A_e \rightarrow A_e + a_e$

$A_o + A_e \rightarrow A_o + a_o$

two leaders XOR their parity, and one becomes follower



Examples of predicate computation

Parity: $\varphi(a)$=Y $\Leftrightarrow$ a is odd

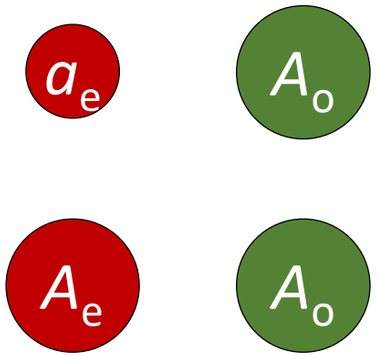
input species A_o (subscript o/e means ODD/EVEN, and capital *A* means it is leader)

$A_o + A_o \rightarrow A_e + a_e$

$A_e + A_e \rightarrow A_e + a_e$

$A_o + A_e \rightarrow A_o + a_o$

two leaders XOR their parity, and one becomes follower



Examples of predicate computation

Parity: $\varphi(a)$=Y $\Leftrightarrow$ a is odd

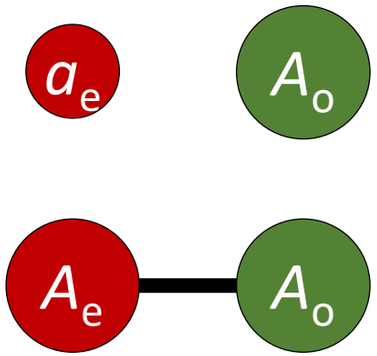
input species A_o (subscript o/e means ODD/EVEN, and capital A means it is leader)

$A_o + A_o \rightarrow A_e + a_e$

$A_e + A_e \rightarrow A_e + a_e$

$A_o + A_e \rightarrow A_o + a_o$

two leaders XOR their parity, and one becomes follower



Examples of predicate computation

Parity: $\varphi(a)$=Y $\Leftrightarrow$ a is odd

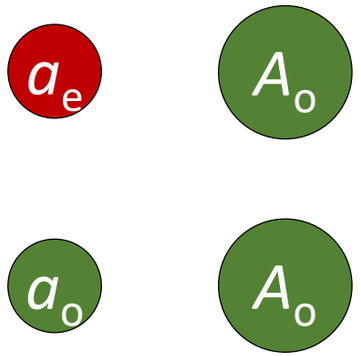
input species A_o (subscript o/e means ODD/EVEN, and capital *A* means it is leader)

$A_o + A_o \rightarrow A_e + a_e$

$A_e + A_e \rightarrow A_e + a_e$

$A_o + A_e \rightarrow A_o + a_o$

two leaders XOR their parity, and one becomes follower



Examples of predicate computation

Parity: $\varphi(a)$=Y $\Leftrightarrow$ a is odd

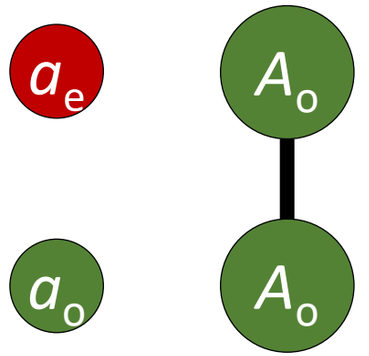
input species A_o (subscript o/e means ODD/EVEN, and capital *A* means it is leader)

$A_o + A_o \rightarrow A_e + a_e$

$A_e + A_e \rightarrow A_e + a_e$

$A_o + A_e \rightarrow A_o + a_o$

two leaders XOR their parity, and one becomes follower



Examples of predicate computation

Parity: $\varphi(a)$=Y $\Leftrightarrow$ a is odd

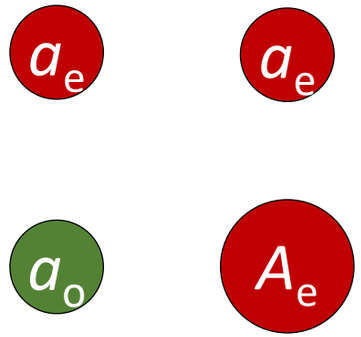
input species A_o (subscript o/e means ODD/EVEN, and capital A means it is leader)

$A_o + A_o \rightarrow A_e + a_e$

$A_e + A_e \rightarrow A_e + a_e$

$A_o + A_e \rightarrow A_o + a_o$

two leaders XOR their parity, and one becomes follower



Examples of predicate computation

Parity: $\varphi(a)$=Y $\Leftrightarrow$ a is odd

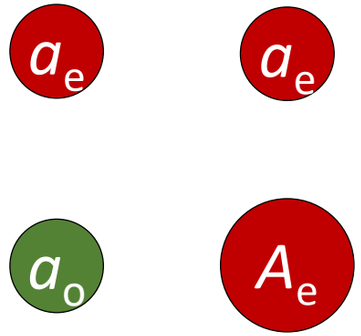
input species A_o (subscript o/e means ODD/EVEN, and capital *A* means it is leader)

$A_o + A_o \rightarrow A_e + a_e$
$A_e + A_e \rightarrow A_e + a_e$
$A_o + A_e \rightarrow A_o + a_o$

two leaders XOR their parity, and one becomes follower

$A_o + a_e \rightarrow A_o + a_o$
$A_e + a_o \rightarrow A_e + a_e$

leader overwrites bit of follower



Examples of predicate computation

Parity: $\varphi(a)$=Y $\Leftrightarrow$ a is odd

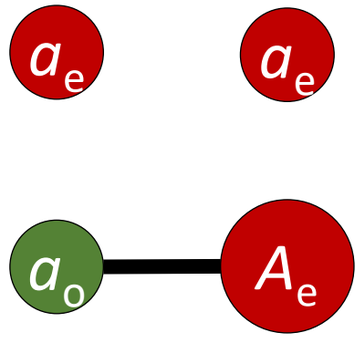
input species A_o (subscript o/e means ODD/EVEN, and capital *A* means it is leader)

$A_o + A_o \rightarrow A_e + a_e$
$A_e + A_e \rightarrow A_e + a_e$
$A_o + A_e \rightarrow A_o + a_o$

two leaders XOR their parity, and one becomes follower

$A_o + a_e \rightarrow A_o + a_o$
$A_e + a_o \rightarrow A_e + a_e$

leader overwrites bit of follower



Examples of predicate computation

Parity: $\varphi(a)$=Y $\Leftrightarrow$ a is odd

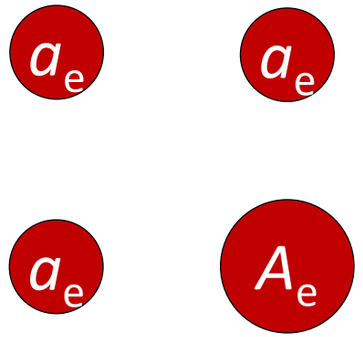
input species A_o (subscript o/e means ODD/EVEN, and capital *A* means it is leader)

$A_o + A_o \rightarrow A_e + a_e$
$A_e + A_e \rightarrow A_e + a_e$
$A_o + A_e \rightarrow A_o + a_o$

two leaders XOR their parity, and one becomes follower

$A_o + a_e \rightarrow A_o + a_o$
$A_e + a_o \rightarrow A_e + a_e$

leader overwrites bit of follower



Examples of predicate computation

Parity: $\varphi(a)$=Y $\Leftrightarrow$ a is odd

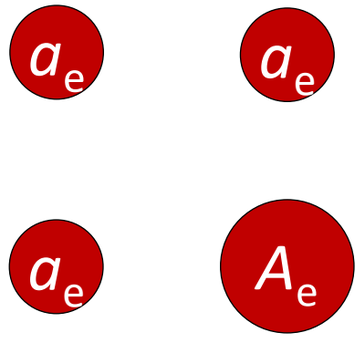
input species A_o (subscript o/e means ODD/EVEN, and capital A means it is leader)

$A_o + A_o \rightarrow A_e + a_e$
$A_e + A_e \rightarrow A_e + a_e$
$A_o + A_e \rightarrow A_o + a_o$

two leaders XOR their parity, and one becomes follower

$A_o + a_e \rightarrow A_o + a_o$
$A_e + a_o \rightarrow A_e + a_e$

leader overwrites bit of follower



Not execution bounded!

Examples of predicate computation

Majority: $\varphi(a,b) = \mathrm{Y} \Leftrightarrow a \geq b$

Examples of predicate computation

Majority: $\varphi(a,b) = Y \Leftrightarrow a \geq b$

$A+B \rightarrow a+b$ (both become "followers" but preserve difference between *A*'s and *B*'s)

[Draief, Vojnovic. Convergence speed of binary interval consensus. *SIAM Journal on Control and Optimization*, 50(3):1087–1109, 2012]
[Mertzios, Nikoletseas, Raptopoulos, Spirakis, Determining Majority in Networks with Local Interactions and very Small Local Memory, *Distributed Computing* 2015]

Examples of predicate computation

Majority: $\varphi(a,b) = Y \Leftrightarrow a \geq b$

$A+B \rightarrow a+b$ (both become "followers" but preserve difference between *A*'s and *B*'s)

$A+b \rightarrow A+a$ (leader changes vote of follower)

$B+a \rightarrow B+b$ (leader changes vote of follower)

[Draief, Vojnovic. Convergence speed of binary interval consensus. *SIAM Journal on Control and Optimization*, 50(3):1087–1109, 2012]
[Mertzios, Nikoletseas, Raptopoulos, Spirakis, Determining Majority in Networks with Local Interactions and very Small Local Memory, *Distributed Computing* 2015]

Examples of predicate computation

Majority: $\varphi(a,b) = Y \Leftrightarrow a \geq b$

$A+B \rightarrow a+b$ (both become "followers" but preserve difference between *A*'s and *B*'s)

$A+b \rightarrow A+a$ (leader changes vote of follower)

$B+a \rightarrow B+b$ (leader changes vote of follower)

$a+b \rightarrow a+a$ (tiebreaker if $a=b$)

[Draief, Vojnovic. Convergence speed of binary interval consensus. *SIAM Journal on Control and Optimization*, 50(3):1087–1109, 2012]
[Mertzios, Nikoletseas, Raptopoulos, Spirakis, Determining Majority in Networks with Local Interactions and very Small Local Memory, *Distributed Computing* 2015]

Examples of predicate computation

Majority: $\varphi(a,b) = Y \Leftrightarrow a \geq b$

$A+B \rightarrow a+b$ (both become "followers" but preserve difference between A's and B's)

$A+b \rightarrow A+a$ (leader changes vote of follower)

$B+a \rightarrow B+b$ (leader changes vote of follower)

$a+b \rightarrow a+a$ (tiebreaker if $a=b$)

Not execution bounded!

[Draief, Vojnovic. Convergence speed of binary interval consensus. *SIAM Journal on Control and Optimization*, 50(3):1087–1109, 2012]
[Mertzios, Nikoletseas, Raptopoulos, Spirakis, Determining Majority in Networks with Local Interactions and very Small Local Memory, *Distributed Computing* 2015]

Limits of stable computation

Theorem: φ: $\mathbb{N}^k \to$ {Y,N} is stably computable by a CRN if and only if φ is *semilinear*.

semilinear = Boolean combination of threshold and mod predicates:

take weighted sum $s = w_1 \cdot x_1 + \ldots w_k \cdot x_k$ of inputs $x_1 \ldots x_k$ and ask if

$s \leq$ constant c?

$s \equiv c \bmod m$ for constants c,m?

$a>b$? $a=b$? a is odd? $a>1$? $a>1$ and b is odd?

NOT $a=b^2$? a is a power of 2? a is prime?

[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]

Outline

- Formal definition of chemical reaction networks
- Execution bounded chemical reaction networks and linear potential functions
- What is “computation” with chemical reactions?
- **Limitations of computation with execution bounded chemical reaction networks**
- Possibilities of computation with execution bounded chemical reaction networks

Noncollapsing CRNs

Definition: A CRN is **noncollapsing** if $\lim_{n\to\infty} s(n) = \infty$, where $s(n)$ = size of smallest stable state reachable from any initial state of size n.

Noncollapsing CRNs

Definition: A CRN is **noncollapsing** if $\lim_{n\to\infty} s(n) = \infty$, where $s(n)$ = size of smallest stable state reachable from any initial state of size n.

Rules out CRNs such as

$A_o + A_o \to A_e$

$A_e + A_e \to A_e$

$A_o + A_e \to A_o$

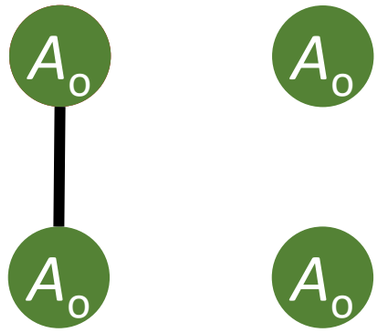
which computes parity but always ends up with a single voter.

Noncollapsing CRNs

Definition: A CRN is **noncollapsing** if $\lim_{n\to\infty} s(n) = \infty$, where $s(n)$ = size of smallest stable state reachable from any initial state of size n.

Rules out CRNs such as

$A_o + A_o \to A_e$
$A_e + A_e \to A_e$
$A_o + A_e \to A_o$



which computes parity but always ends up with a single voter.

Noncollapsing CRNs

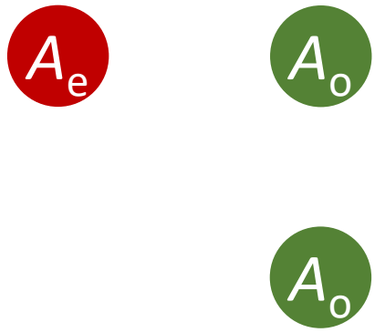
Definition: A CRN is **noncollapsing** if $\lim_{n\to\infty} s(n) = \infty$, where $s(n)$ = size of smallest stable state reachable from any initial state of size n.

Rules out CRNs such as

$A_o + A_o \to A_e$

$A_e + A_e \to A_e$

$A_o + A_e \to A_o$



which computes parity but always ends up with a single voter.

Noncollapsing CRNs

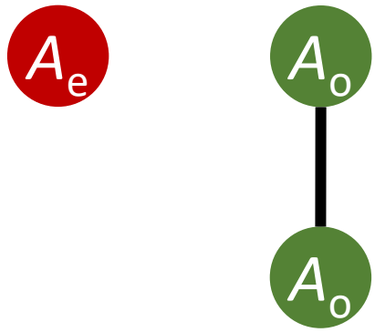
Definition: A CRN is **noncollapsing** if $\lim_{n\to\infty} s(n) = \infty$, where $s(n)$ = size of smallest stable state reachable from any initial state of size n.

Rules out CRNs such as

$A_o + A_o \to A_e$

$A_e + A_e \to A_e$

$A_o + A_e \to A_o$



which computes parity but always ends up with a single voter.

Noncollapsing CRNs

Definition: A CRN is **noncollapsing** if $\lim_{n\to\infty} s(n) = \infty$, where $s(n)$ = size of smallest stable state reachable from any initial state of size n.

Rules out CRNs such as

$A_o + A_o \to A_e$

$A_e + A_e \to A_e$

$A_o + A_e \to A_o$

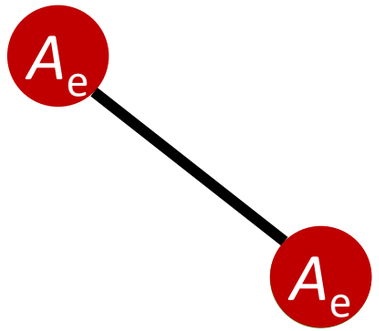
which computes parity but always ends up with a single voter.

Noncollapsing CRNs

Definition: A CRN is **noncollapsing** if $\lim_{n\to\infty} s(n) = \infty$, where $s(n)$ = size of smallest stable state reachable from any initial state of size n.

Rules out CRNs such as

$A_o + A_o \to A_e$
$A_e + A_e \to A_e$
$A_o + A_e \to A_o$



which computes parity but always ends up with a single voter.

Noncollapsing CRNs

Definition: A CRN is **noncollapsing** if $\lim_{n\to\infty} s(n) = \infty$, where $s(n)$ = size of smallest stable state reachable from any initial state of size n.

Rules out CRNs such as

$A_o + A_o \to A_e$
$A_e + A_e \to A_e$
$A_o + A_e \to A_o$

which computes parity but always ends up with a single voter.

Eventually constant predicates

Definition: A predicate $\varphi: \mathbb{N}^k \to \{Y,N\}$ is **eventually constant** if, for some $c \in \mathbb{N}$, $\varphi(\mathbf{x})$ is constant on all inputs $\mathbf{x} \geqq (c,c,\ldots,c)$.

Eventually constant predicates

Definition: A predicate $\varphi: \mathbb{N}^k \to \{Y,N\}$ is **eventually constant** if, for some $c \in \mathbb{N}$, $\varphi(\mathbf{x})$ is constant on all inputs $\mathbf{x} \geqq (c,c,\ldots,c)$.

Non-eventually constant predicates:

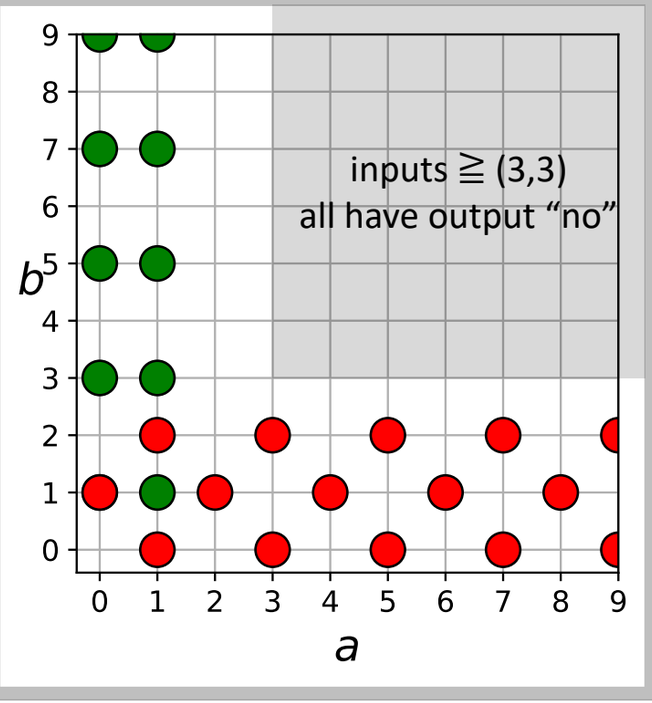
majority ($a \geq b$?)

parity (a is odd?)

equality ($a=b$?)

and most anything interesting.

Example of eventually constant predicate:
$a < 2$ and b is odd, or $b < 3$ and $a+b$ is odd



Limitations of execution bounded CRNs

Theorem: If a CRN stably computing φ is noncollapsing and execution bounded from every input state, then φ is eventually constant.

Limitations of execution bounded CRNs

Theorem: If a CRN stably computing φ is noncollapsing and execution bounded from every input state, then φ is eventually constant.

Proof: complex.

Proof that such CRNs cannot compute parity (a is odd?):

Limitations of execution bounded CRNs

Theorem: If a CRN stably computing φ is noncollapsing and execution bounded from every input state, then φ is eventually constant.

Proof: complex.

Proof that such CRNs cannot compute parity (a is odd?):

1. Start with $\{A\}$, CRN can reach to stable **YES** state $\mathbf{s}_1$.

{A}

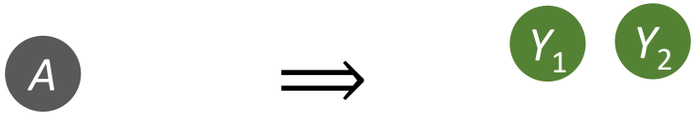
Limitations of execution bounded CRNs

Theorem: If a CRN stably computing φ is noncollapsing and execution bounded from every input state, then φ is eventually constant.

Proof: complex.

Proof that such CRNs cannot compute parity (a is odd?):

1. Start with $\{A\}$, CRN can reach to stable **YES** state $\mathbf{s}_1$.



{A}

$\mathbf{s}_1$

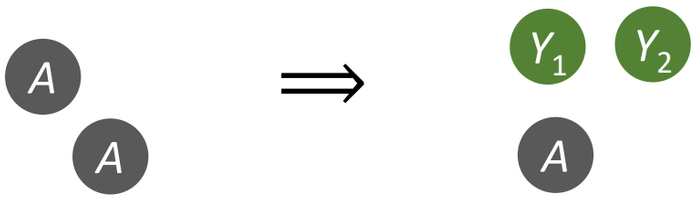
Limitations of execution bounded CRNs

Theorem: If a CRN stably computing φ is noncollapsing and execution bounded from every input state, then φ is eventually constant.

Proof: complex.

Proof that such CRNs cannot compute parity (a is odd?):

1. Start with $\{A\}$, CRN can reach to stable **YES** state $\mathbf{s}_1$.
2. Add 1 A. The state $\mathbf{s}_1+\{A\}$ is reachable from $\{2A\}$, so the CRN can reach from there to a stable **NO** state $\mathbf{s}_2$.



$\Longrightarrow$

$\{A\} + \{A\}$ $\quad\quad$ $\mathbf{s}_1 + \{A\}$

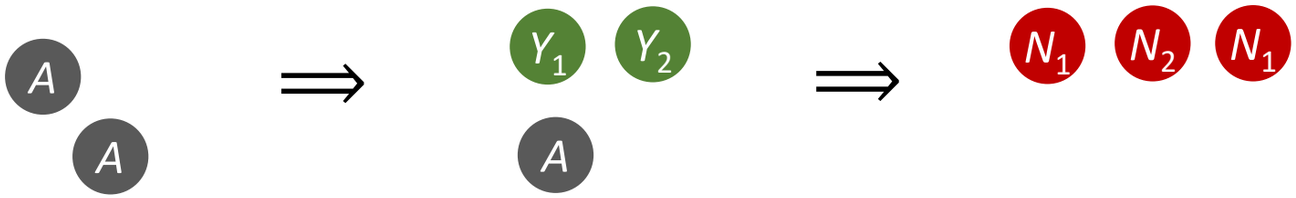
Limitations of execution bounded CRNs

Theorem: If a CRN stably computing φ is noncollapsing and execution bounded from every input state, then φ is eventually constant.

Proof: complex.

Proof that such CRNs cannot compute parity (a is odd?):

1. Start with $\{A\}$, CRN can reach to stable **YES** state $\mathbf{s}_1$.
2. Add 1 A. The state $\mathbf{s}_1+\{A\}$ is reachable from $\{2A\}$, so the CRN can reach from there to a stable **NO** state $\mathbf{s}_2$.



{A} + {A} $\mathbf{s}_1$ + {A} $\mathbf{s}_2$

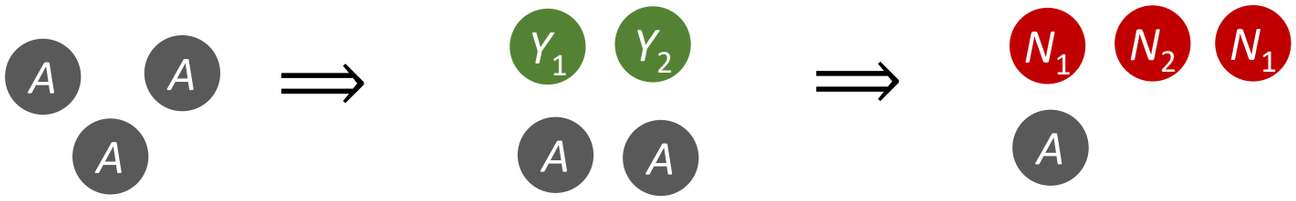
Limitations of execution bounded CRNs

Theorem: If a CRN stably computing φ is noncollapsing and execution bounded from every input state, then φ is eventually constant.

Proof: complex.

Proof that such CRNs cannot compute parity (a is odd?):

1. Start with $\{A\}$, CRN can reach to stable **YES** state $\mathbf{s}_1$.
2. Add 1 A. The state $\mathbf{s}_1+\{A\}$ is reachable from $\{2A\}$, so the CRN can reach from there to a stable **NO** state $\mathbf{s}_2$.
3. Add 1 A. The state $\mathbf{s}_2+\{A\}$ is reachable from $\{3A\}$, so the CRN can reach from there to a stable **YES** state $\mathbf{s}_3$.
4. …



{A} + {A}
+ {A}

$\mathbf{s}_1$ + {A}
+ {A}

$\mathbf{s}_2$ + {A}

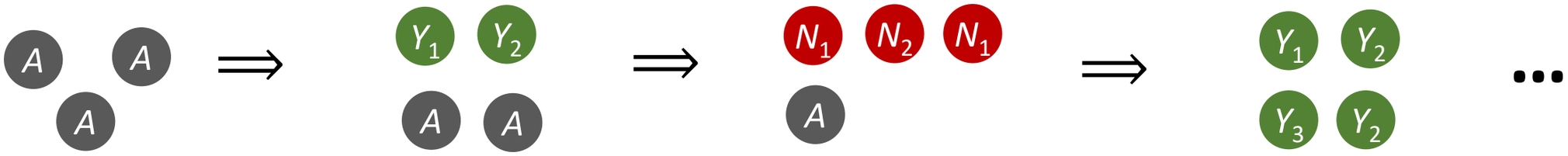
Limitations of execution bounded CRNs

Theorem: If a CRN stably computing φ is noncollapsing and execution bounded from every input state, then φ is eventually constant.

Proof: complex.

Proof that such CRNs cannot compute parity (a is odd?):

1. Start with $\{A\}$, CRN can reach to stable **YES** state $\mathbf{s}_1$.
2. Add 1 A. The state $\mathbf{s}_1+\{A\}$ is reachable from $\{2A\}$, so the CRN can reach from there to a stable **NO** state $\mathbf{s}_2$.
3. Add 1 A. The state $\mathbf{s}_2+\{A\}$ is reachable from $\{3A\}$, so the CRN can reach from there to a stable **YES** state $\mathbf{s}_3$.
4. …



{A} + {A} + {A}

$\mathbf{s}_1$ + {A} + {A}

$\mathbf{s}_2$ + {A}

$\mathbf{s}_3$

Limitations of execution bounded CRNs

- Since CRN is execution bounded from all states, it has a linear potential function Φ.

Limitations of execution bounded CRNs

- Since CRN is execution bounded from all states, it has a linear potential function Φ.
- Adding $\{A\}$ to $\mathbf{s}_i$ increases Φ by the constant $\Phi(\{A\})$.

Limitations of execution bounded CRNs

- Since CRN is execution bounded from all states, it has a linear potential function Φ.
- Adding $\{A\}$ to $\mathbf{s}_i$ increases Φ by the constant $\Phi(\{A\})$.
- To get from $\mathbf{s}_i+\{A\}$ to $\mathbf{s}_{i+1}$, since $\lim_{i\to\infty} |\mathbf{s}_i| = \infty$ (noncollapsing), we must execute increasingly more reactions as $i \to \infty$, which all decrease Φ.
  - Key reason: all species vote, so all molecules in $\mathbf{s}_i$ must be removed to switch the output.

Limitations of execution bounded CRNs

- Since CRN is execution bounded from all states, it has a linear potential function Φ.
- Adding $\{A\}$ to $\mathbf{s}_i$ increases Φ by the constant $\Phi(\{A\})$.
- To get from $\mathbf{s}_i+\{A\}$ to $\mathbf{s}_{i+1}$, since $\lim_{i\to\infty} |\mathbf{s}_i| = \infty$ (noncollapsing), we must execute increasingly more reactions as $i \to \infty$, which all decrease Φ.
  - Key reason: all species vote, so all molecules in $\mathbf{s}_i$ must be removed to switch the output.
- After some i, the net change in Φ, in going from $\mathbf{s}_i$ to $\mathbf{s}_i+\{A\}$ to $\mathbf{s}_{i+1}$, is negative.

Limitations of execution bounded CRNs

- Since CRN is execution bounded from all states, it has a linear potential function Φ.
- Adding $\{A\}$ to $\mathbf{s}_i$ increases Φ by the constant $\Phi(\{A\})$.
- To get from $\mathbf{s}_i+\{A\}$ to $\mathbf{s}_{i+1}$, since $\lim_{i\to\infty} |\mathbf{s}_i| = \infty$ (noncollapsing), we must execute increasingly more reactions as $i \to \infty$, which all decrease Φ.
  - Key reason: all species vote, so all molecules in $\mathbf{s}_i$ must be removed to switch the output.
- After some i, the net change in Φ, in going from $\mathbf{s}_i$ to $\mathbf{s}_i+\{A\}$ to $\mathbf{s}_{i+1}$, is negative.
- Since Φ is nonnegative, at some point we cannot continue. QED

Outline

- Formal definition of chemical reaction networks
- Execution bounded chemical reaction networks and linear potential functions
- What is “computation” with chemical reactions?
- Limitations of computation with execution bounded chemical reaction networks
- **Possibilities of computation with execution bounded chemical reaction networks**

Are execution bounded CRNs good for any computation?

Yes! Execution bounded CRNs *can* stably compute all semilinear predicates if the CRN is *leader-driven*: it starts with an "initial leader", e.g., to compute majority ($a \geq b$?), start in initial state {1 *L*, *a A*, *b B*}… these are execution bounded from such states, but not from states with multiple leaders.

We also relax the voting requirement and allow only the leader to vote. (though this requirement can be relaxed; not shown in slides)

Single-voting CRNs

Definition: A CRN computing a predicate $\varphi: \mathbb{N}^k \to \{Y,N\}$ is **single-voting** if all states reachable from the input have a single voter.

Such CRNs are leader-driven: valid initial configurations have a single leader/voter molecule, and only the leader votes.

Semilinear predicates are Boolean combinations of threshold and mod predicates

Recall: Theorem: φ: $\mathbb{N}^k \to$ {Y,N} is stably computable by a CRN if and only if φ is *semilinear*. (= Boolean combination of threshold and mod predicates)

To show execution bounded CRNs can compute all semilinear predicates, it suffices to show:

- They can compute all threshold predicates.
- They can compute all mod predicates.
- They can be composed to compute AND, OR, and NOT of other CRNs.

Execution bounded CRNs can compute threshold predicates

Theorem: Every threshold predicate [$w_1x_1 + \ldots + w_kx_k \leq c$?] can be stably computed by a single-voting execution bounded CRN.

Execution bounded CRNs can compute threshold predicates

Theorem: Every threshold predicate [$w_1x_1 + \ldots + w_kx_k \leq c$?] can be stably computed by a single-voting execution bounded CRN.

Proof by example: To compute [$2x_1 + 3x_2 - 5x_3 \leq 4$?], in addition to inputs X_1, X_2, X_3, start with 1 L_Y (yes voter/leader) and 4 N, and have reactions:

Execution bounded CRNs can compute threshold predicates

Theorem: Every threshold predicate [$w_1x_1 + \ldots + w_kx_k \leq c$?] can be stably computed by a single-voting execution bounded CRN.

Proof by example: To compute [$2x_1 + 3x_2 - 5x_3 \leq 4$?], in addition to inputs X_1, X_2, X_3, start with 1 L_Y (yes voter/leader) and 4 N, and have reactions:

$$X_1 \rightarrow 2P$$
$$X_2 \rightarrow 3P$$
$$X_3 \rightarrow 5N$$

Execution bounded CRNs can compute threshold predicates

Theorem: Every threshold predicate $[w_1x_1 + \ldots + w_kx_k \leq c?]$ can be stably computed by a single-voting execution bounded CRN.

Proof by example: To compute $[2x_1 + 3x_2 - 5x_3 \leq 4?]$, in addition to inputs X_1, X_2, X_3, start with 1 L_Y (yes voter/leader) and 4 N, and have reactions:

$$X_1 \to 2P$$
$$X_2 \to 3P$$

P will have count = weighted sum of inputs with positive weights

$$X_3 \to 5N$$

N will have count = weighted sum of inputs with negative weights (including constant –4)

Execution bounded CRNs can compute threshold predicates

Theorem: Every threshold predicate [$w_1x_1 + \ldots + w_kx_k \leq c$?] can be stably computed by a single-voting execution bounded CRN.

Proof by example: To compute [$2x_1 + 3x_2 - 5x_3 \leq 4$?], in addition to inputs X_1, X_2, X_3, start with 1 L_Y (yes voter/leader) and 4 N, and have reactions:

$$X_1 \to 2P$$
$$X_2 \to 3P$$

P will have count = weighted sum of inputs with positive weights

$$X_3 \to 5N$$

N will have count = weighted sum of inputs with negative weights (including constant –4)

Now we compute majority [$P \leq N$?]

$$L_Y + P \to L_N$$
$$L_N + N \to L_Y$$

Execution bounded CRNs can compute mod predicates

Theorem: Every mod predicate [$w_1x_1 + \ldots + w_kx_k \equiv c \mod m$?] can be stably computed by a single-voting execution bounded CRN.

Execution bounded CRNs can compute mod predicates

Theorem: Every mod predicate [$w_1x_1 + \ldots + w_kx_k \equiv c \mod m$?] can be stably computed by a single-voting execution bounded CRN.

Proof by example: To compute [$2x_1 + 3x_2 \equiv 4 \mod 5$?], in addition to inputs X_1, X_2, start with 1 L_0, and have reactions:

$$L_i + X_1 \rightarrow L_{i+2 \bmod 5} \qquad \text{for } i = 0,1,2,3,4$$
$$L_i + X_2 \rightarrow L_{i+3 \bmod 5}$$

L_4 votes **yes**, L_0, L_1, L_2, L_3 vote **no**

Composing CRNs to compute Boolean combinations of predicates

Theorem: If single-voting, execution bounded CRNs C_1 and C_2 stably compute predicates $\varphi_1: \mathbb{N}^k \to \{Y,N\}$ and $\varphi_2: \mathbb{N}^k \to \{Y,N\}$, then there are single-voting, execution bounded CRNs stably computing [φ_1 and φ_2], [φ_1 or φ_2], and [not φ_1].

Composing CRNs to compute Boolean combinations of predicates

Theorem: If single-voting, execution bounded CRNs C_1 and C_2 stably compute predicates $\varphi_1: \mathbb{N}^k \to \{Y,N\}$ and $\varphi_2: \mathbb{N}^k \to \{Y,N\}$, then there are single-voting, execution bounded CRNs stably computing [φ_1 and φ_2], [φ_1 or φ_2], and [not φ_1].

Proof : To compute [not φ_1], swap votes of voting species.

Composing CRNs to compute Boolean combinations of predicates

Theorem: If single-voting, execution bounded CRNs C_1 and C_2 stably compute predicates $\varphi_1: \mathbb{N}^k \to \{Y,N\}$ and $\varphi_2: \mathbb{N}^k \to \{Y,N\}$, then there are single-voting, execution bounded CRNs stably computing [φ_1 and φ_2], [φ_1 or φ_2], and [not φ_1].

Proof : To compute [not φ_1], swap votes of voting species.

To compute [φ_1 and φ_2] and [φ_1 or φ_2], "split" each input X via reaction $X \to X_1 + X_2$, so C_1 operates on X_1 and C_2 operates on X_2.

Composing CRNs to compute Boolean combinations of predicates

Theorem: If single-voting, execution bounded CRNs C_1 and C_2 stably compute predicates $\varphi_1: \mathbb{N}^k \to \{Y,N\}$ and $\varphi_2: \mathbb{N}^k \to \{Y,N\}$, then there are single-voting, execution bounded CRNs stably computing [φ_1 and φ_2], [φ_1 or φ_2], and [not φ_1].

Proof : To compute [not φ_1], swap votes of voting species.

To compute [φ_1 and φ_2] and [φ_1 or φ_2], "split" each input X via reaction $X \to X_1 + X_2$, so C_1 operates on X_1 and C_2 operates on X_2.

"global" voters of composed CRN:
V_{NN}, V_{NY}, V_{YN}, V_{YY}; start with 1 V_{NN}

Let S_Y, S_N be yes and no voters of C_1
Let T_Y, T_N be yes and no voters of C_2

Composing CRNs to compute Boolean combinations of predicates

Theorem: If single-voting, execution bounded CRNs C_1 and C_2 stably compute predicates $\varphi_1: \mathbb{N}^k \to \{Y,N\}$ and $\varphi_2: \mathbb{N}^k \to \{Y,N\}$, then there are single-voting, execution bounded CRNs stably computing $[\varphi_1 \text{ and } \varphi_2]$, $[\varphi_1 \text{ or } \varphi_2]$, and $[\text{not } \varphi_1]$.

Proof : To compute $[\text{not } \varphi_1]$, swap votes of voting species.

To compute $[\varphi_1 \text{ and } \varphi_2]$ and $[\varphi_1 \text{ or } \varphi_2]$, "split" each input X via reaction $X \to X_1 + X_2$, so C_1 operates on X_1 and C_2 operates on X_2.

"global" voters of composed CRN:
V_{NN}, V_{NY}, V_{YN}, V_{YY}; start with 1 V_{NN}

Let S_Y, S_N be yes and no voters of C_1
Let T_Y, T_N be yes and no voters of C_2

Voters of C_1 and C_2 influence global voters:

$S_Y + V_{NN} \to S_Y + V_{YN}$
$S_Y + V_{NY} \to S_Y + V_{YY}$

$T_Y + V_{NN} \to T_Y + V_{NY}$
$T_Y + V_{YN} \to T_Y + V_{YY}$

$S_N + V_{YN} \to S_N + V_{NN}$
$S_N + V_{YY} \to S_N + V_{NY}$

$T_N + V_{NY} \to T_N + V_{NN}$
$T_N + V_{YY} \to T_N + V_{YN}$

Composing CRNs to compute Boolean combinations of predicates

Theorem: If single-voting, execution bounded CRNs C_1 and C_2 stably compute predicates $\varphi_1: \mathbb{N}^k \to \{Y,N\}$ and $\varphi_2: \mathbb{N}^k \to \{Y,N\}$, then there are single-voting, execution bounded CRNs stably computing $[\varphi_1 \text{ and } \varphi_2]$, $[\varphi_1 \text{ or } \varphi_2]$, and $[\text{not } \varphi_1]$.

Proof : To compute $[\text{not } \varphi_1]$, swap votes of voting species.

To compute $[\varphi_1 \text{ and } \varphi_2]$ and $[\varphi_1 \text{ or } \varphi_2]$, "split" each input X via reaction $X \to X_1 + X_2$, so C_1 operates on X_1 and C_2 operates on X_2.

"global" voters of composed CRN:
V_{NN}, V_{NY}, V_{YN}, V_{YY}; start with 1 V_{NN}

Let S_Y, S_N be yes and no voters of C_1
Let T_Y, T_N be yes and no voters of C_2

C_1 is execution bounded, so can only switch between S_Y and S_N a finite number of times, limiting how many times we can flip between $V_{N?}$ and $V_{Y?}$, so full CRN is execution bounded.

Voters of C_1 and C_2 influence global voters:

$$S_Y + V_{NN} \to S_Y + V_{YN}$$
$$S_Y + V_{NY} \to S_Y + V_{YY}$$
$$S_N + V_{YN} \to S_N + V_{NN}$$
$$S_N + V_{YY} \to S_N + V_{NY}$$

$$T_Y + V_{NN} \to T_Y + V_{NY}$$
$$T_Y + V_{YN} \to T_Y + V_{YY}$$
$$T_N + V_{NY} \to T_N + V_{NN}$$
$$T_N + V_{YY} \to T_N + V_{YN}$$

Open question

- Using standard stochastic model of chemical kinetics (not shown), the execution bounded CRNs stably computing semilinear predicates can be shown to take expected time $O(n \log n)$ to converge.

Open question

- Using standard stochastic model of chemical kinetics (not shown), the execution bounded CRNs stably computing semilinear predicates can be shown to take expected time $O(n \log n)$ to converge.
- Without the execution bounded constraint, it is known they can be computed exponentially faster: polylog(n). [Angluin, Aspnes, Eisenstat, Fast computation by population protocols with a leader, *DISC* 2006]

Open question

- Using standard stochastic model of chemical kinetics (not shown), the execution bounded CRNs stably computing semilinear predicates can be shown to take expected time $O(n \log n)$ to converge.
- Without the execution bounded constraint, it is known they can be computed exponentially faster: polylog(n). [Angluin, Aspnes, Eisenstat, Fast computation by population protocols with a leader, *DISC* 2006]
- Conjecture: Execution bounded CRNs require $\Omega(n)$ time to stably compute any non-eventually constant predicate (e.g., majority or parity).

Thank you!

Questions?