The limits of chemical computing

Computation with chemical reaction networks

David Doty

23rd International Meeting on DNA and Molecular Programming

September 2017

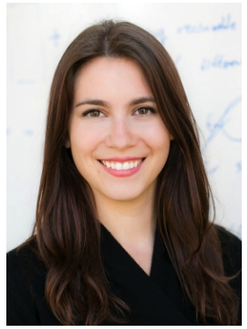
Acknowledgments

co-authors

Amanda Belleville
UC-Davis

Ho-Lin Chen
NTU

Rachel Cummings
Georgia Tech



Monir Hajiaghayi
UBC

David Soloveichik
UT-Austin

Banff International Research Station
for Mathematical Innovation and Discovery

Anne Condon

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

Chemical caucusing

$X + Y \rightarrow U + U$

[Angluin, Aspnes, Eisenstat, A simple population protocol for fast robust approximate majority, *DISC* 2007]

Chemical caucusing

$X + Y \rightarrow U + U$

$X + U \rightarrow X + X$

$Y + U \rightarrow Y + Y$

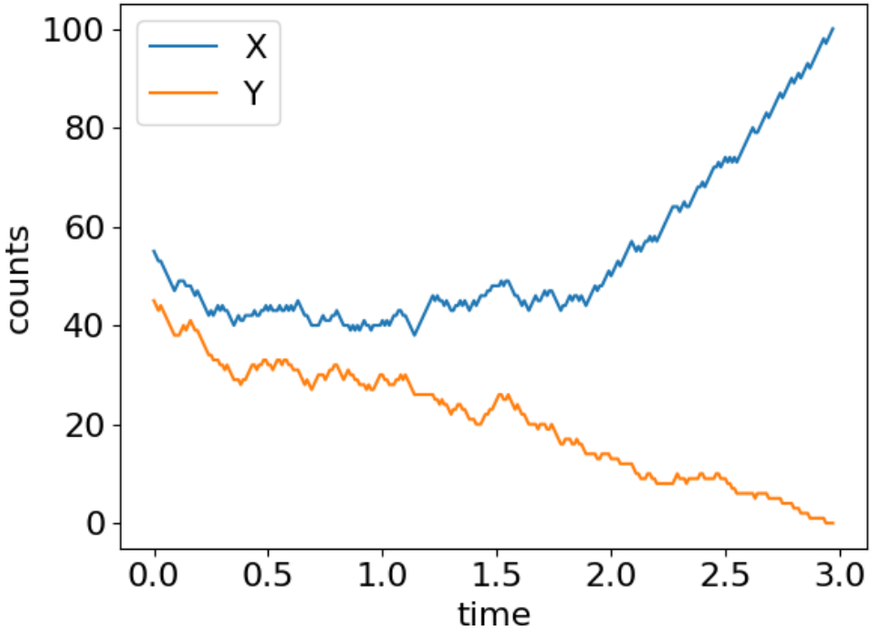
[Angluin, Aspnes, Eisenstat, A simple population protocol for fast robust approximate majority, *DISC* 2007]

Chemical caucusing

$X+Y \rightarrow U+U$

$X+U \rightarrow X+X$

$Y+U \rightarrow Y+Y$



distributed algorithm for *"approximate majority"*:
initial majority (X or Y) quickly overtakes whole population

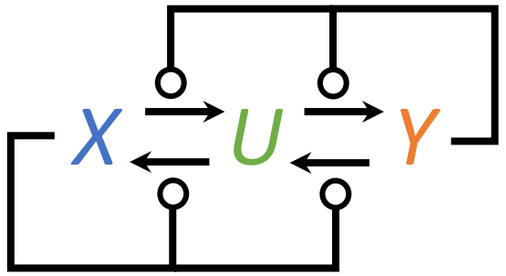
[Angluin, Aspnes, Eisenstat, A simple population protocol for fast robust approximate majority, *DISC* 2007]

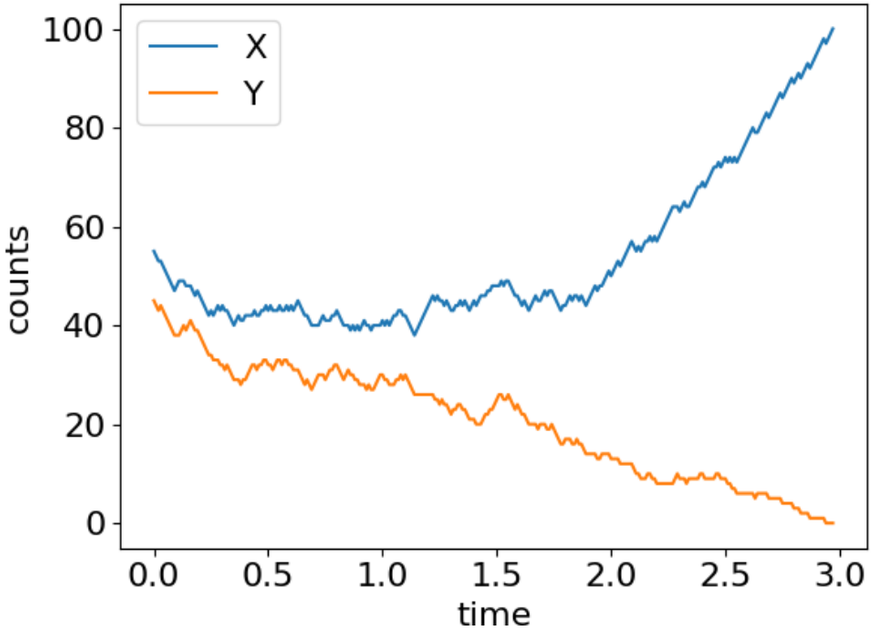
Chemical caucusing

$X+Y \rightarrow U+U$

$X+U \rightarrow X+X$

$Y+U \rightarrow Y+Y$

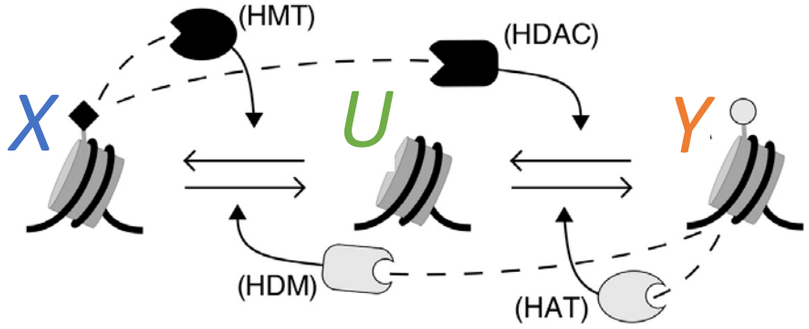




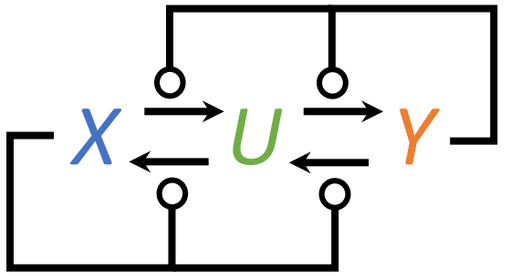
distributed algorithm for *"approximate majority"*:
initial majority (X or Y) quickly overtakes whole population

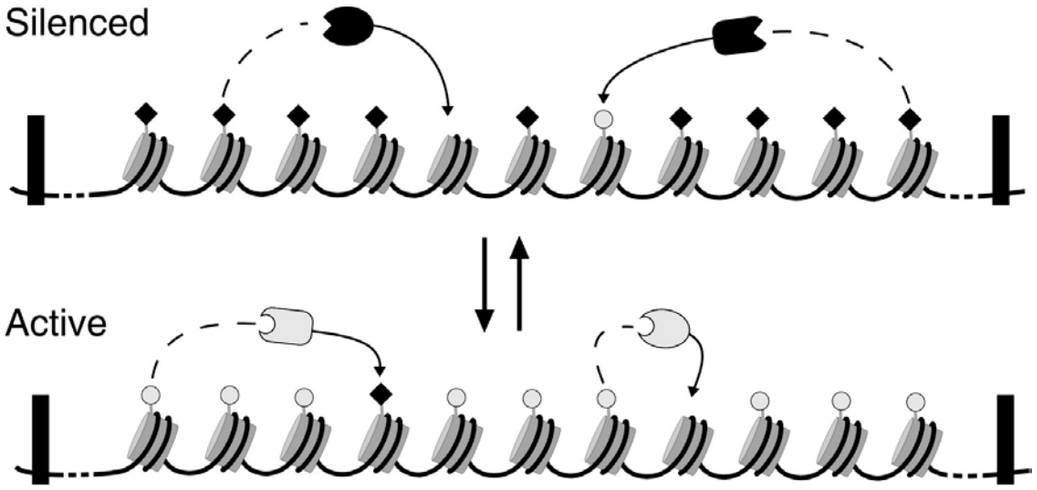
[Angluin, Aspnes, Eisenstat, A simple population protocol for fast robust approximate majority, *DISC* 2007]

Does chemistry compute?



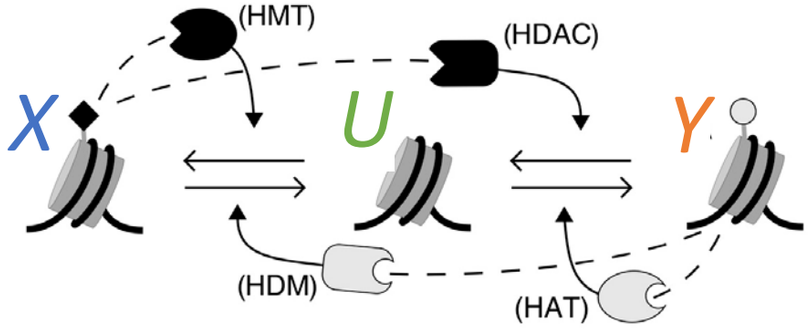
=



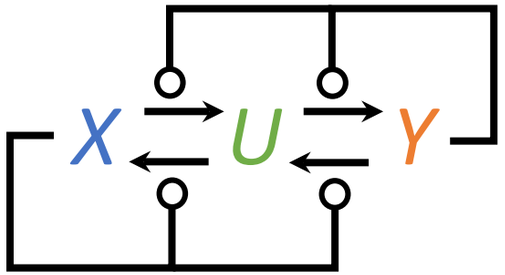


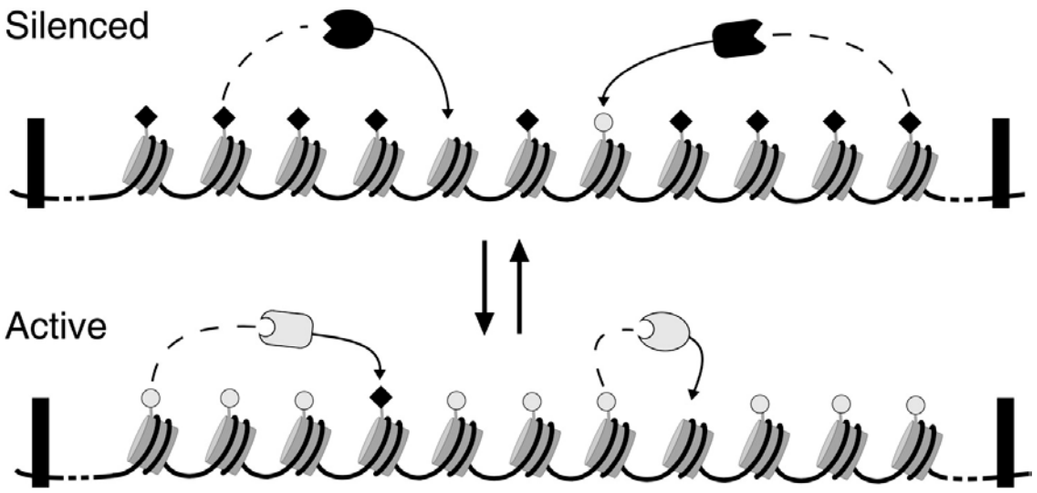
[Dodd, Micheelsen, Sneppen, Thon. Theoretical analysis of epigenetic cell memory by nucleosome modification, *Cell* 2007]

Does chemistry compute?

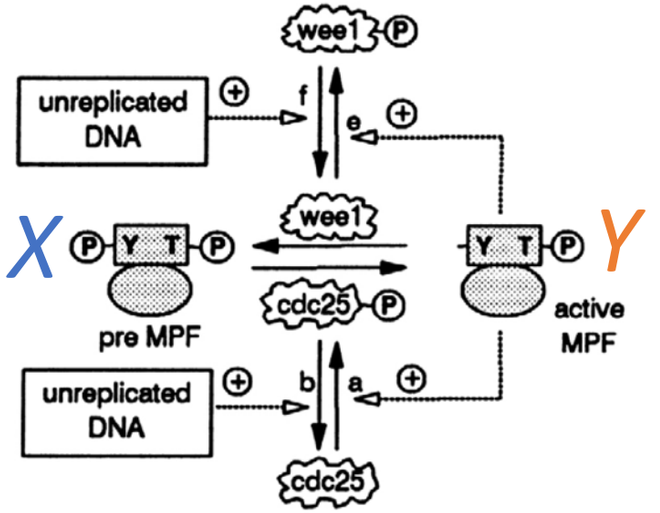


=





≈



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

[Cardelli, Csikász-Nagy. The cell cycle switch computes approximate majority. *Nature Scientific Reports* 2012]

[Cardelli, Morphisms of reaction networks that couple structure to function, *BMC Systems Biology* 2014]

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, Seelig, Winfree, *PNAS* 2010] showed how to physically implement <u>any</u> chemical reaction network using *DNA strand displacement*

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*

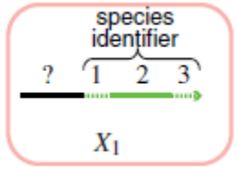
$$X_1 + X_2 \rightarrow X_3$$

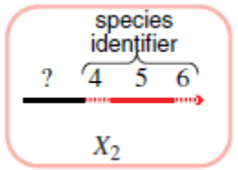
Can we compute with chemistry?

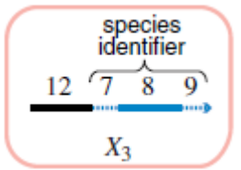
"Not every chemical reaction network describes real chemicals!", i.e. "where's the *compiler*?"

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

$$X_1 + X_2 \rightarrow X_3$$





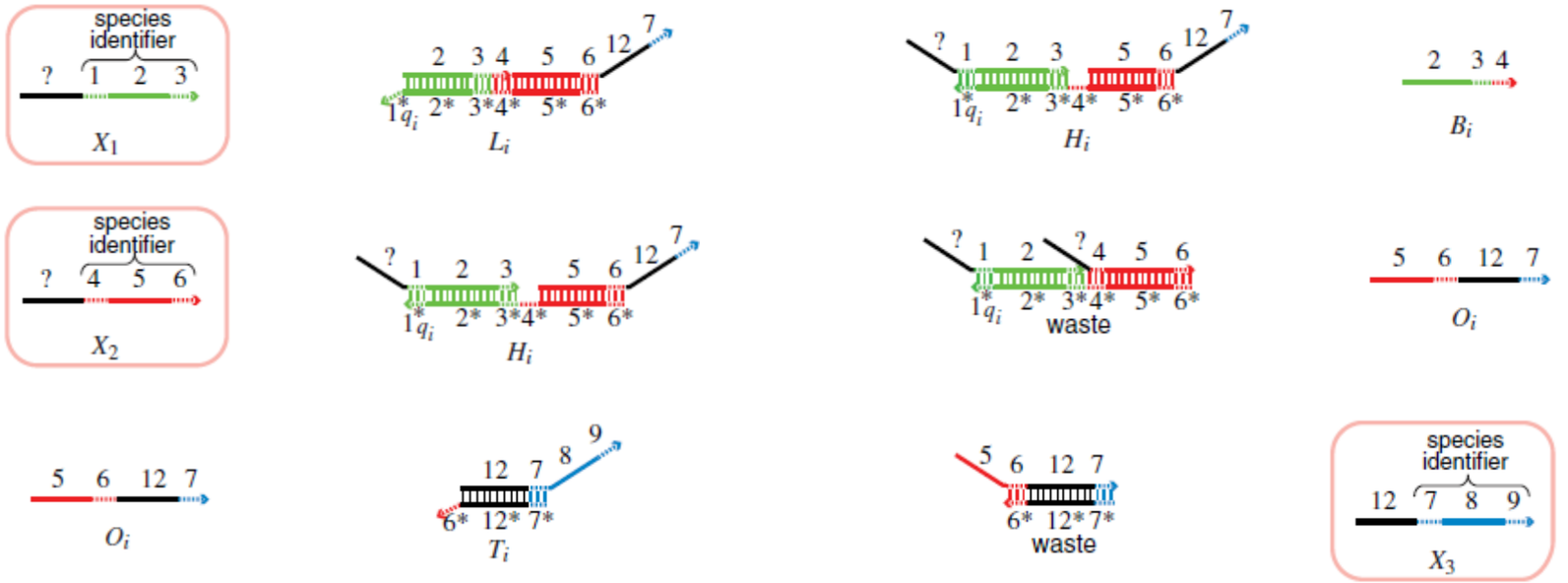


Can we compute with chemistry?

"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 <u>any</u> chemical reaction network using *DNA strand displacement*

$$X_1 + X_2 \rightarrow X_3$$

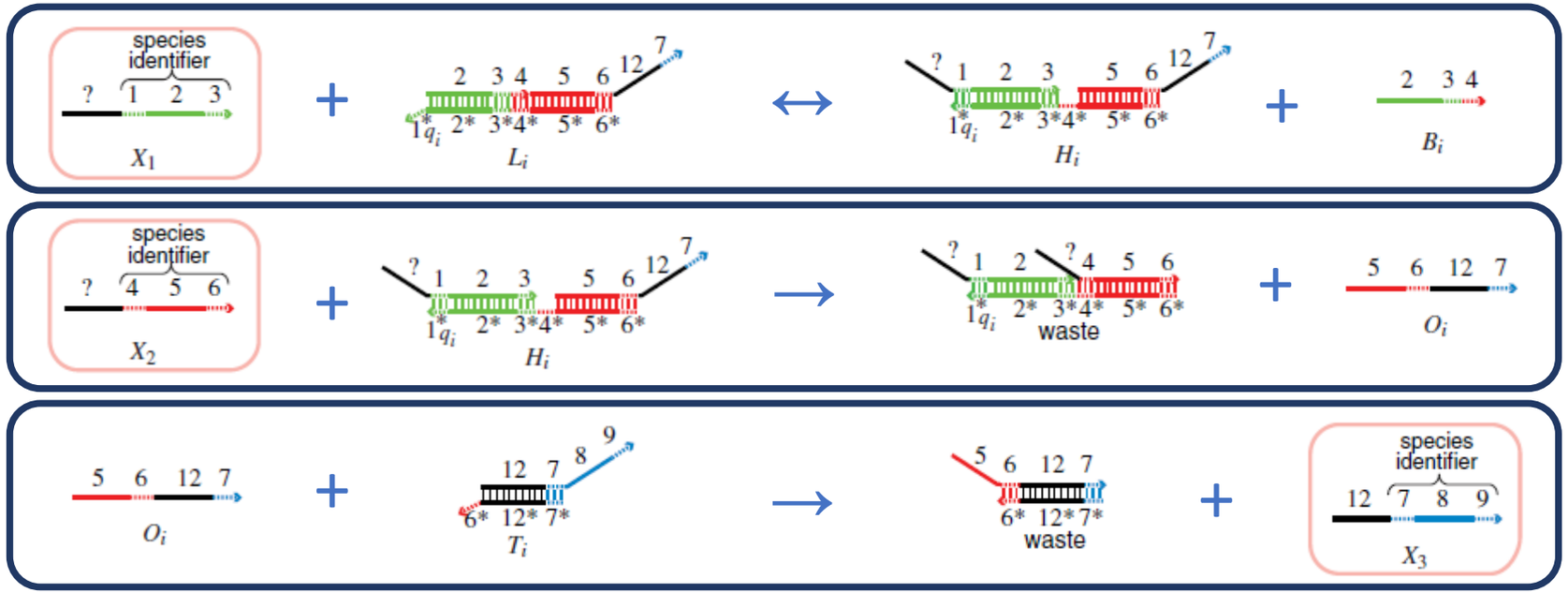


Can we compute with chemistry?

"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 <u>any</u> chemical reaction network using *DNA strand displacement*

$$X_1 + X_2 \rightarrow X_3$$

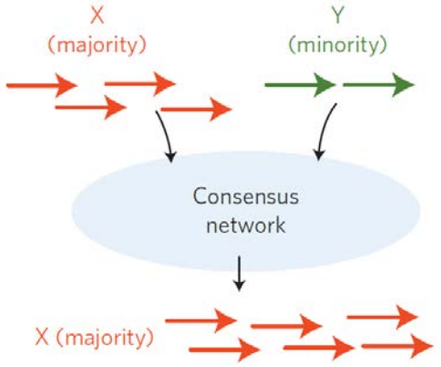


DNA strand displacement implementing A+B → C

video: Microsoft Research Cambridge

Experimental implementations of synthetic chemical reaction networks with DNA

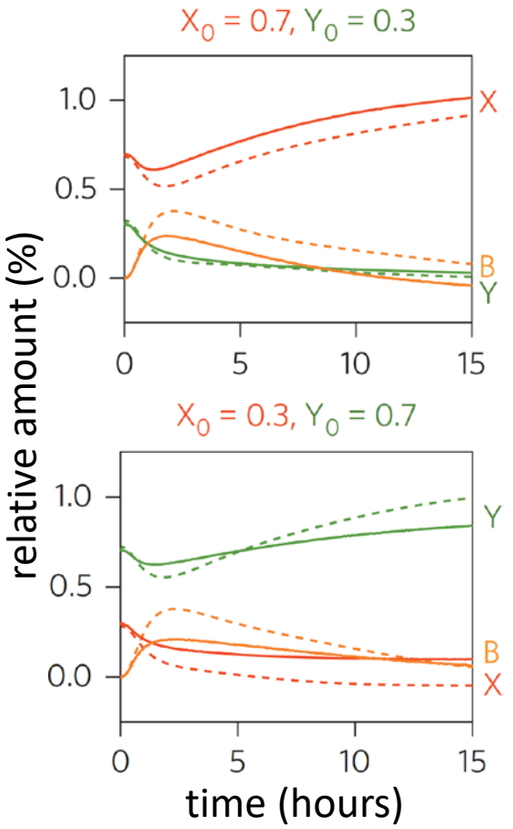
Analog majority computation



$X+Y \rightarrow B+B$

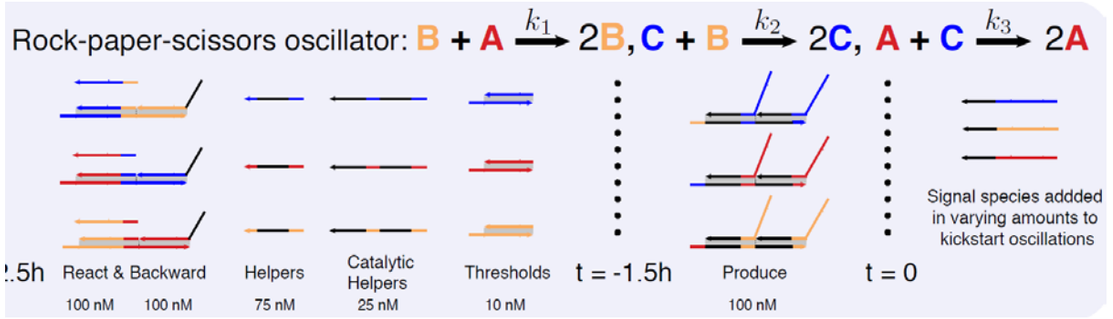
$X+B \rightarrow X+X$

$Y+B \rightarrow Y+Y$



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

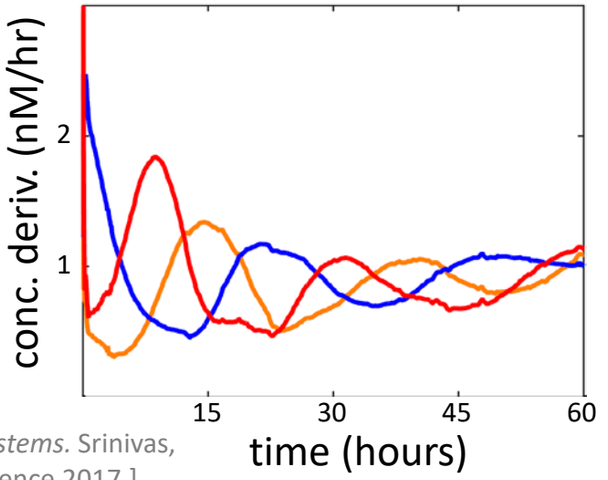
Chemical oscillator



$A+B \rightarrow B+B$

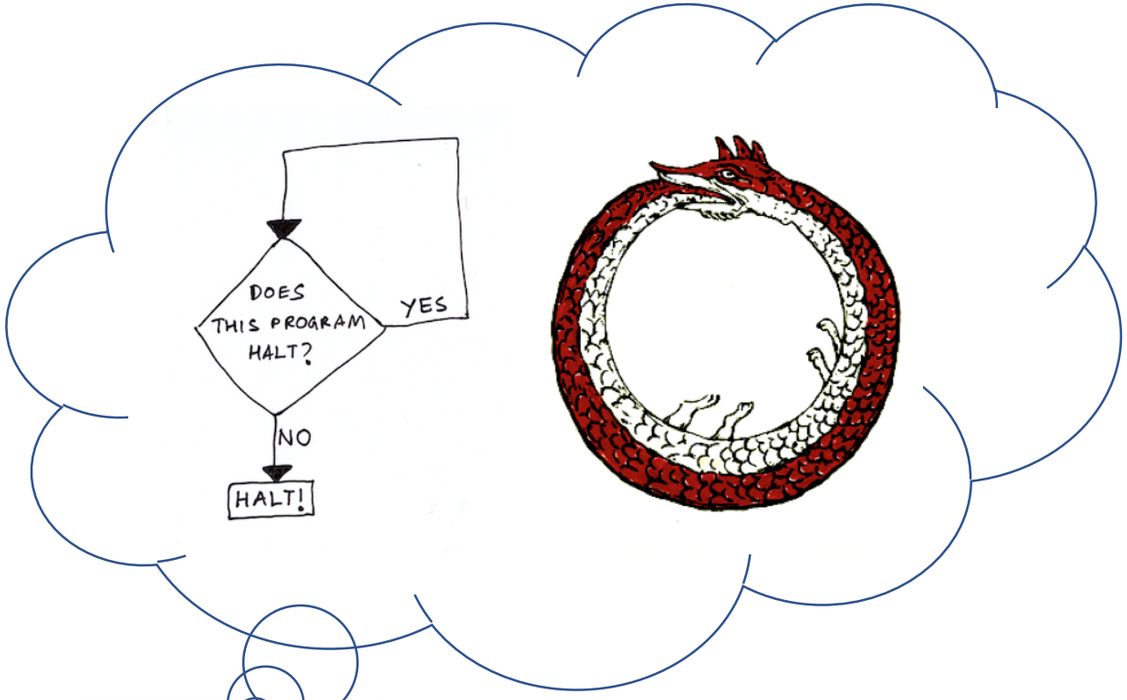
$B+C \rightarrow C+C$

$C+A \rightarrow A+A$



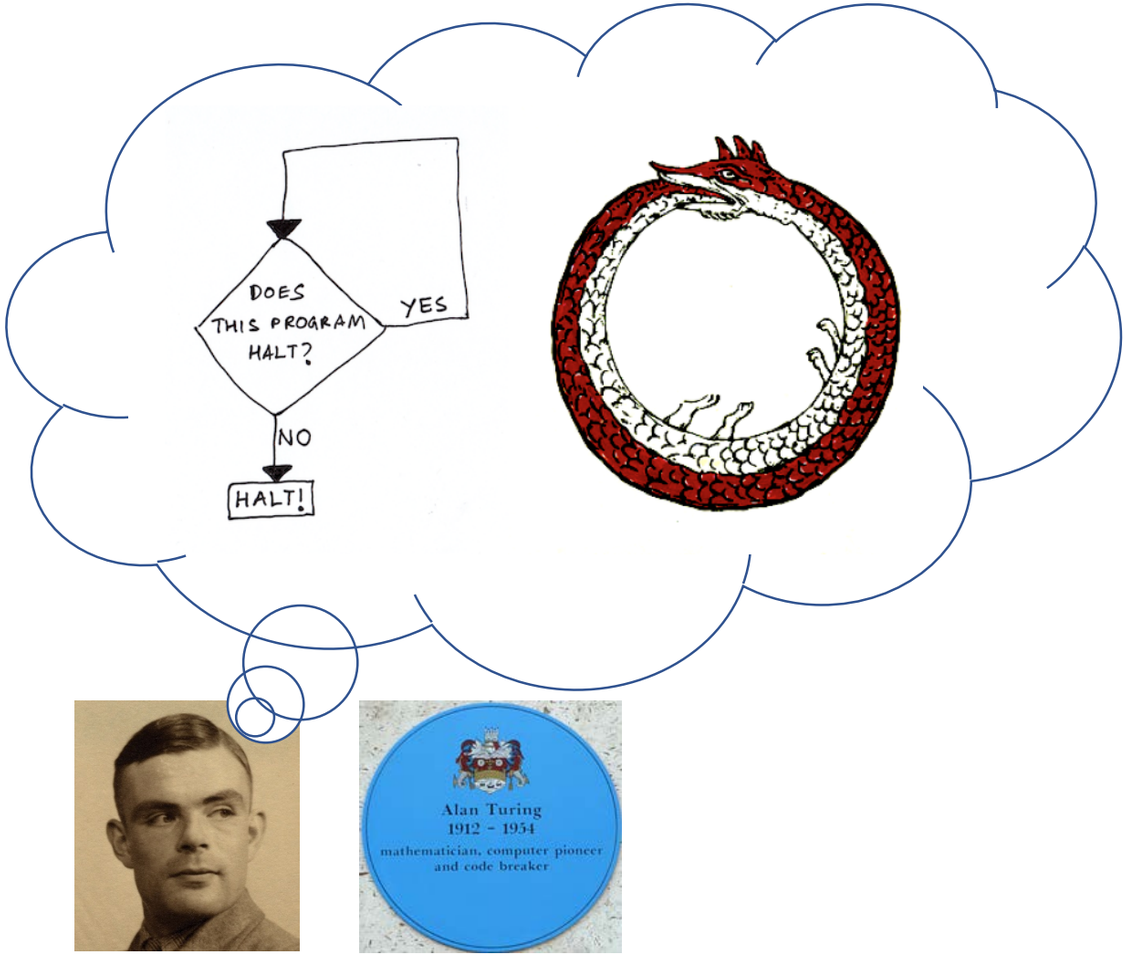
[*Enzyme-free nucleic acid dynamical systems.* Srinivas, Parkin, Seelig, Winfree, Soloveichik, Science 2017.]

Theoretical Computer Science Approach



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

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
- What do we mean by “computation” with chemical reactions?
- What do we mean by “efficient computation” with chemical reactions?
- What is known
- What is not yet known

Chemical Reaction Networks (formal definition)

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

$$A+B \xrightarrow{k_1} A+C$$

$$C \xrightarrow{k_2} A+A$$

$$C+B \xrightarrow{k_3} C$$

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

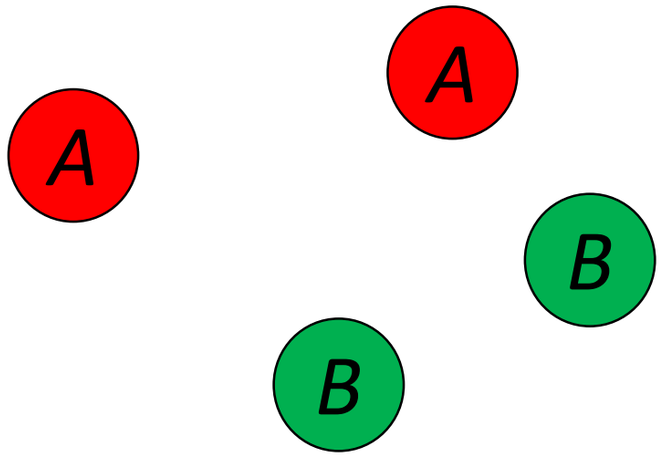
What is **possible**:
Example reaction sequence

α: $A + B \rightarrow A + C$

β: $C \rightarrow A + A$

$A \quad B \quad C$

$\mathbf{x} = (2,\ 2,\ 0)$



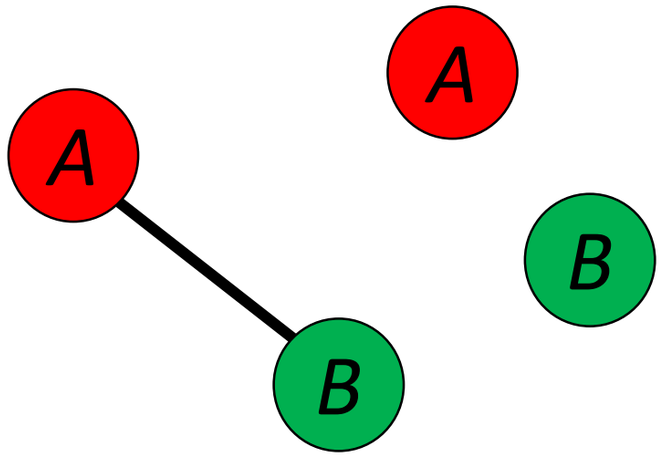
What is **possible**:
Example reaction sequence

α: $A+B \rightarrow A+C$

β: $C \rightarrow A+A$

$A \quad B \quad C$

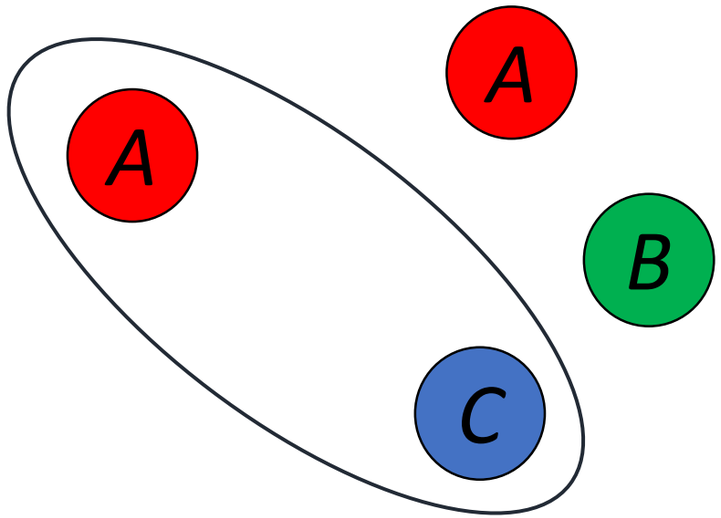
$\mathbf{x} = (2,\ 2,\ 0)$



What is **possible**:
Example reaction sequence

α: $A+B \rightarrow \boxed{A+C}$

β: $C \rightarrow A+A$

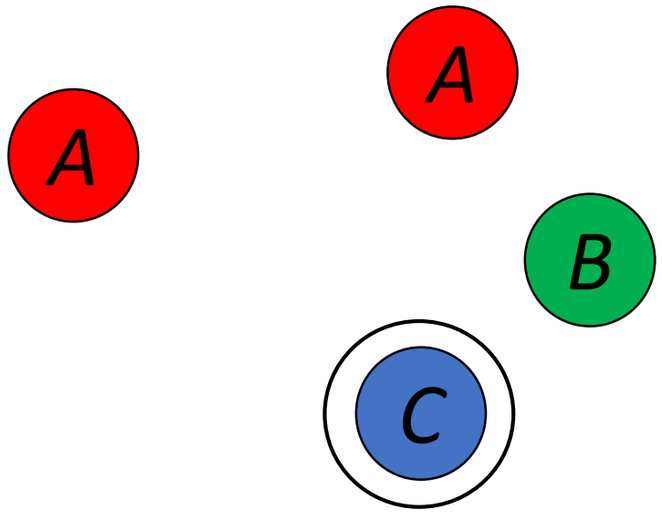


	A	*B*	*C*
x =	**(2,**	**2,**	**0)**
	α ⇓		
	(2,	**1,**	**1)**

What is **possible**:
Example reaction sequence

α: $A + B \rightarrow A + C$

β: $C \rightarrow A + A$

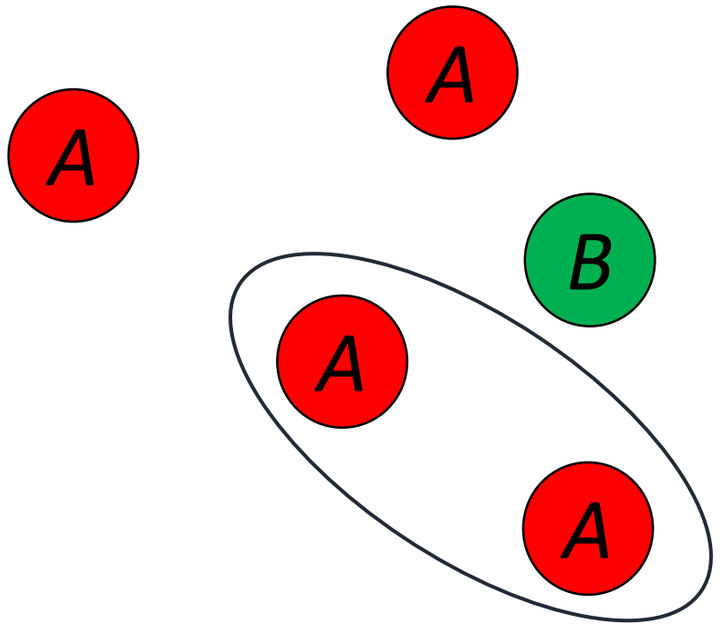


	A	*B*	*C*
x =	**2,**	**2,**	**0)**
α ⇓			
	2,	**1,**	**1)**

What is **possible**:
Example reaction sequence

α: $A+B \rightarrow A+C$

β: $C \rightarrow \boxed{A+A}$

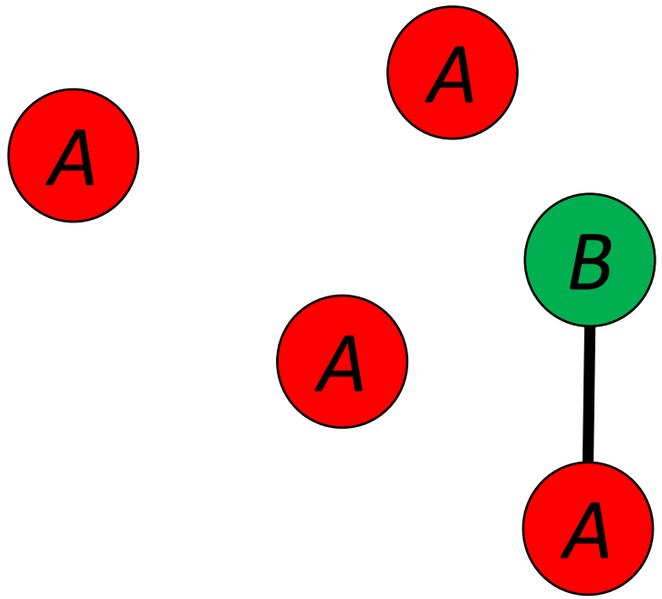


	A	*B*	*C*
x =	**(2,**	**2,**	**0)**
	α ⇓		
	(2,	**1,**	**1)**
	β ⇓		⇘α
	(4,	**1,**	**0)** ⋯

What is **possible**:
Example reaction sequence

α: $A+B \rightarrow A+C$

β: $C \rightarrow A+A$

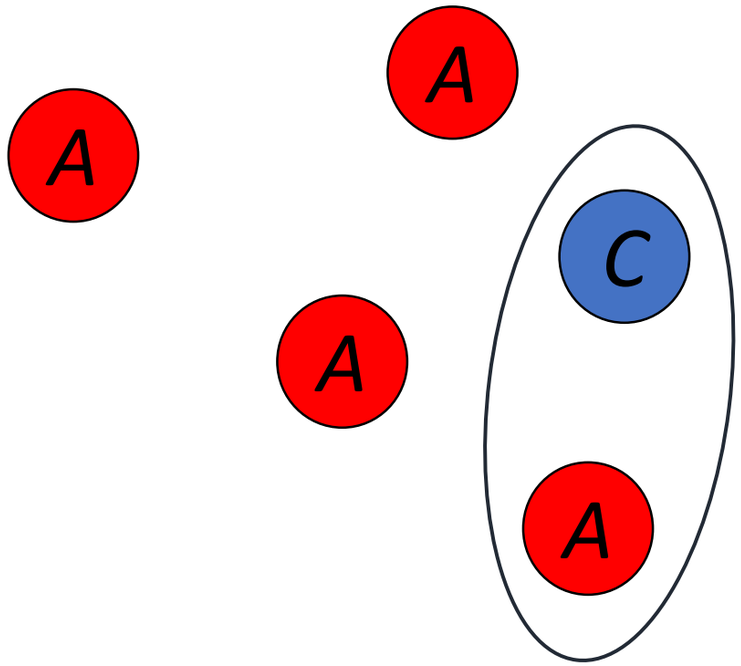


	A	*B*	*C*
x =	**(2,**	**2,**	**0)**
	α ⇓		
	(2,	**1,**	**1)**
	β ⇓		⇘α
	(4,	**1,**	**0)** ⋯

What is **possible**:
Example 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)**
	···		

What is probable:
Stochastic kinetic model of chemical reaction networks

Solution volume v

reaction type	*rate*
$A \xrightarrow{k} \ldots$	$k \cdot \#A$
$A+B \xrightarrow{k} \ldots$	$k \cdot \#A \cdot \#B \,/\, v$

[McQuarrie 1967, van Kampen, Gillespie 1977]

What is **probable**:
Stochastic kinetic model of chemical reaction networks

Solution volume *v*

reaction type	*rate*
$A \xrightarrow{k} \ldots$	$k \cdot \#A$
$A+B \xrightarrow{k} \ldots$	$k \cdot \#A \cdot \#B \,/\, v$

System evolves via a continuous time Markov process:

Pr[next reaction is j^{th} one] = *rate of j^{th} reaction* / (*sum of all reaction rates*)

[McQuarrie 1967, van Kampen, Gillespie 1977]

What is probable:
Stochastic kinetic model of chemical reaction networks

Solution volume *v*

reaction type	*rate*
$A \xrightarrow{k} \ldots$	$k \cdot \#A$
$A+B \xrightarrow{k} \ldots$	$k \cdot \#A \cdot \#B \,/\, v$

System evolves via a continuous time Markov process:

Pr[next reaction is j^{th} one] = *rate of j^{th} reaction* / (*sum of all reaction rates*)

expected time until next reaction is 1 / (*sum of all reaction rates*)

[McQuarrie 1967, van Kampen, Gillespie 1977]

Relationship to distributed computing

population protocol = list of *transitions* such as

$x,y \to x,x$ $\quad$ $a,b \to c,d$ $\quad$ $a,a \to a,a$ (*null* transition)

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

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

Relationship to distributed computing

population protocol = list of *transitions* such as

$x,y \to x,x$ $\quad$ $a,b \to c,d$ $\quad$ $a,a \to a,a$ (*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

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

Relationship to distributed computing

population protocol = list of *transitions* such as

$x,y \to x,x$ $\quad$ $a,b \to c,d$ $\quad$ $a,a \to a,a$ (*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

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

Relationship to distributed computing

population protocol = list of *transitions* such as

$x,y \to x,x$ $\quad$ $a,b \to c,d$ $\quad$ $a,a \to a,a$ (*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)

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

Relationship to distributed computing

population protocol = list of *transitions* such as

$x,y \to x,x$ $\quad$ $a,b \to c,d$ $\quad$ $a,a \to a,a$ (*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 $\subsetneq$ chemical reactions, but "most" ideas that apply to one model also apply to the other

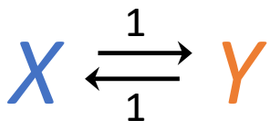
[Angluin, Aspnes, Diamadi, Fischer, Peralta, Computation in networks of passively mobile finite-state sensors, *PODC* 2004]

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”

Some simple reactions



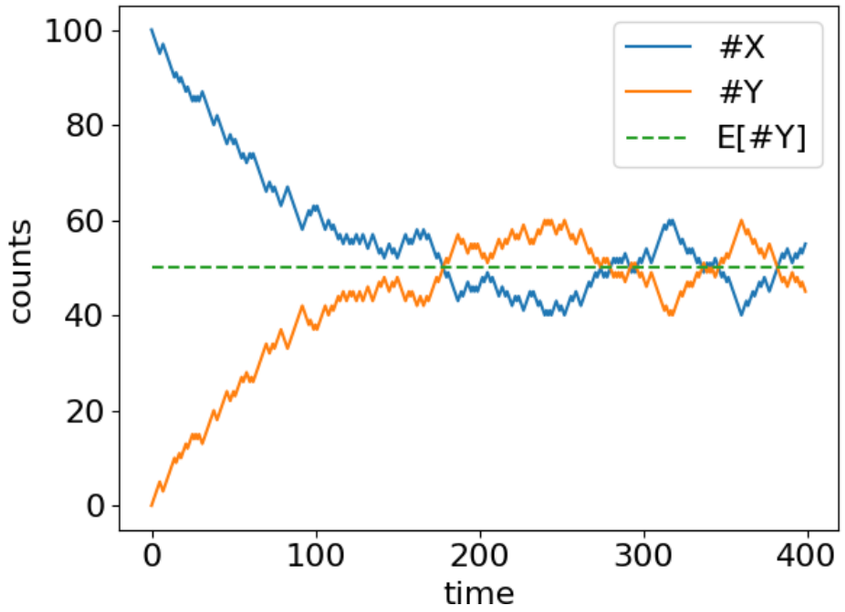
start with *n* copies of molecule *X*

Some simple reactions

$$X \underset{1}{\overset{1}{\rightleftharpoons}} Y$$

start with n copies of molecule X

$\#Y = n/2$ expected at equilibrium



Some simple reactions

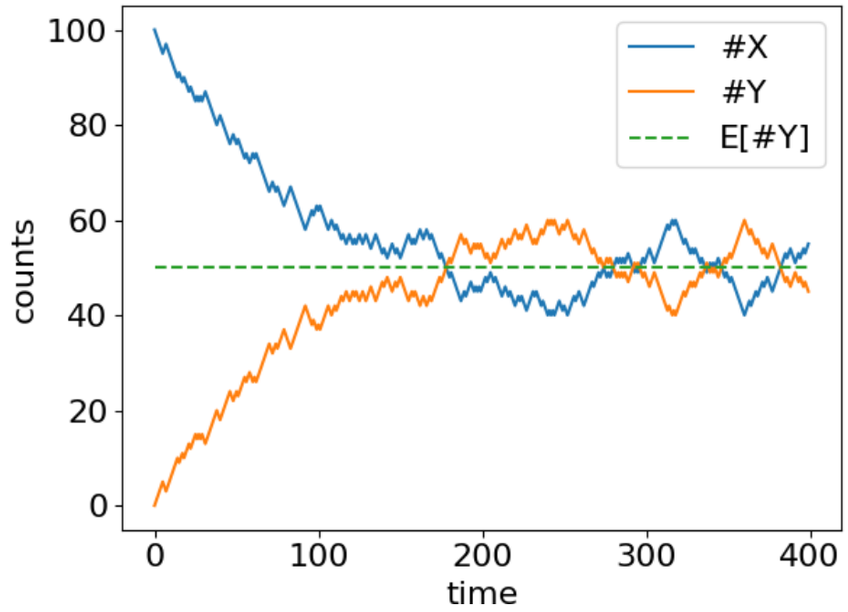
$$X \underset{1}{\overset{1}{\rightleftharpoons}} Y$$

$$X \xrightarrow{1} Y$$

$$X \xrightarrow{1}$$

start with n copies of molecule X

$\#Y = n/2$ expected at equilibrium



Some simple reactions

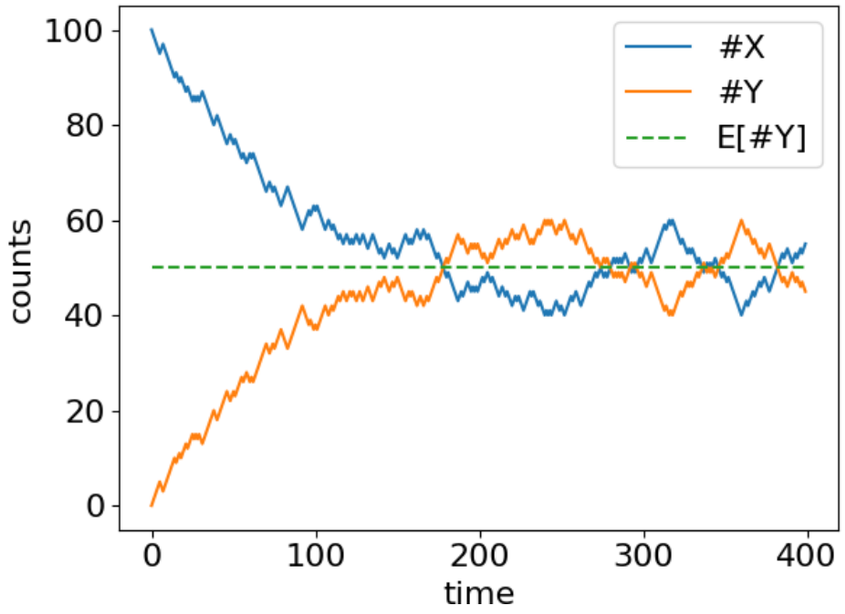
$$X \underset{1}{\overset{1}{\rightleftharpoons}} Y$$

$$X \xrightarrow{1} Y$$

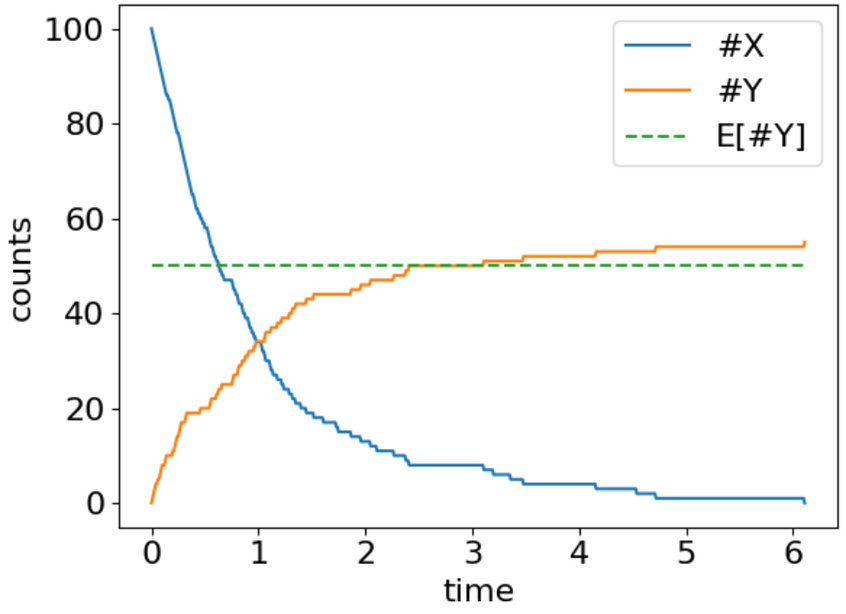
$$X \xrightarrow{1}$$

start with n copies of molecule X

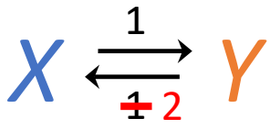
$\#Y = n/2$ expected at equilibrium

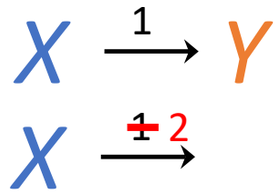


$\#Y$ stabilizes, with expected value $n/2$



Some simple reactions

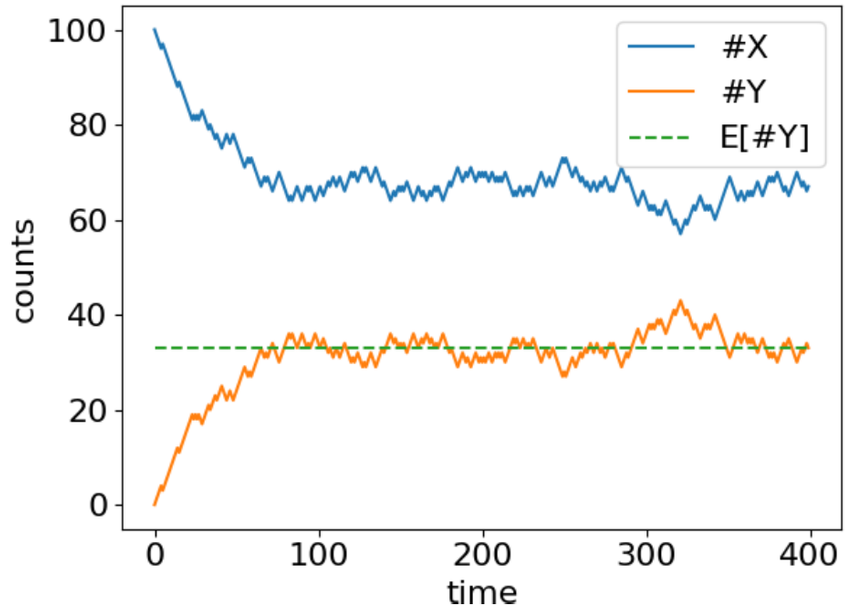


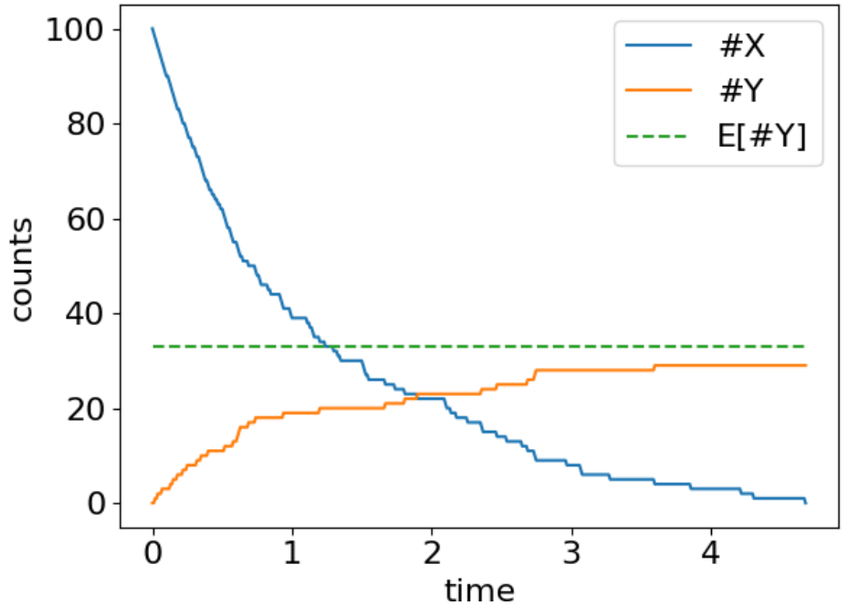


start with *n* copies of molecule *X*

#Y = ~~*n/2*~~ *n/3* expected at equilibrium

#Y <u>stabilizes</u>, with expected value ~~*n/2*~~ *n/3*





Modeling choices in formalizing *“Computing with chemistry”*

Modeling choices in formalizing *“Computing with chemistry”*

- integer counts (“stochastic”) or real concentrations (“mass-action”)?

Modeling choices in formalizing *"Computing with chemistry"*

- integer counts ("stochastic") or real concentrations ("mass-action")?

first part of talk

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 half the amount of X"*

first part of talk

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 half the amount of X"*

first part of talk

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 half 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*

first part of talk

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 half 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*

first part of talk

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 half 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 input $n_1,\ldots,n_k$, what is the initial configuration?
  - only input species present?
  - auxiliary species can be present?

first part of talk

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 half 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 input $n_1,\ldots,n_k$, what is the initial configuration?
  - only input species present?
  - auxiliary species can be present?

first part of talk

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 half 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 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)

first part of talk

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 half 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 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)

first part of talk

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 half 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 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 talk

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 half 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 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?

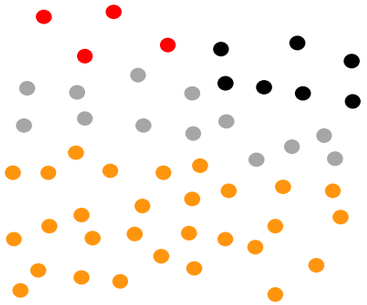
first part of talk

Defining **stable** computation

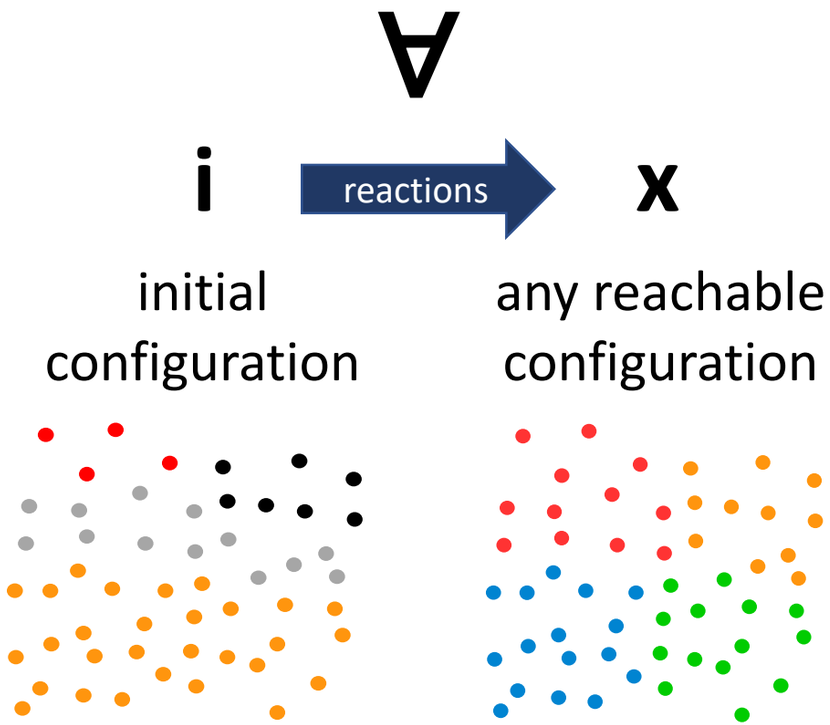
Defining **stable** computation

i

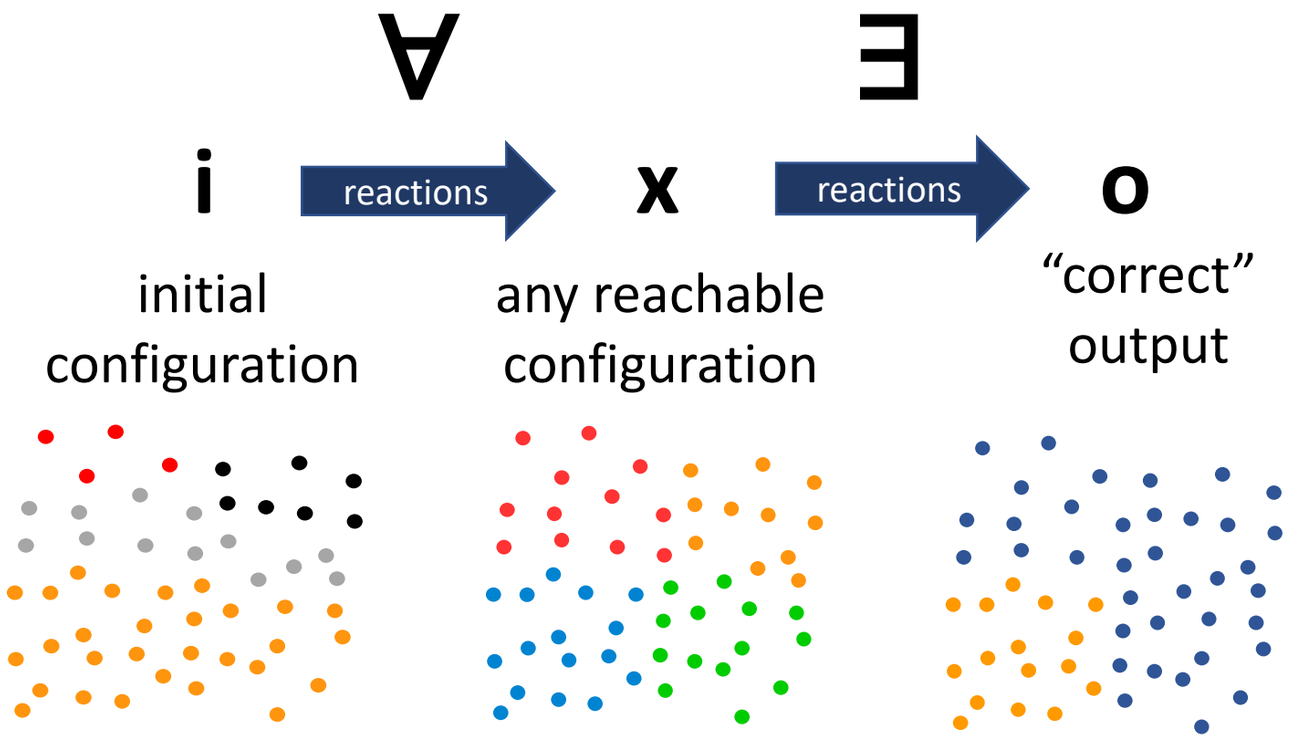
initial
configuration



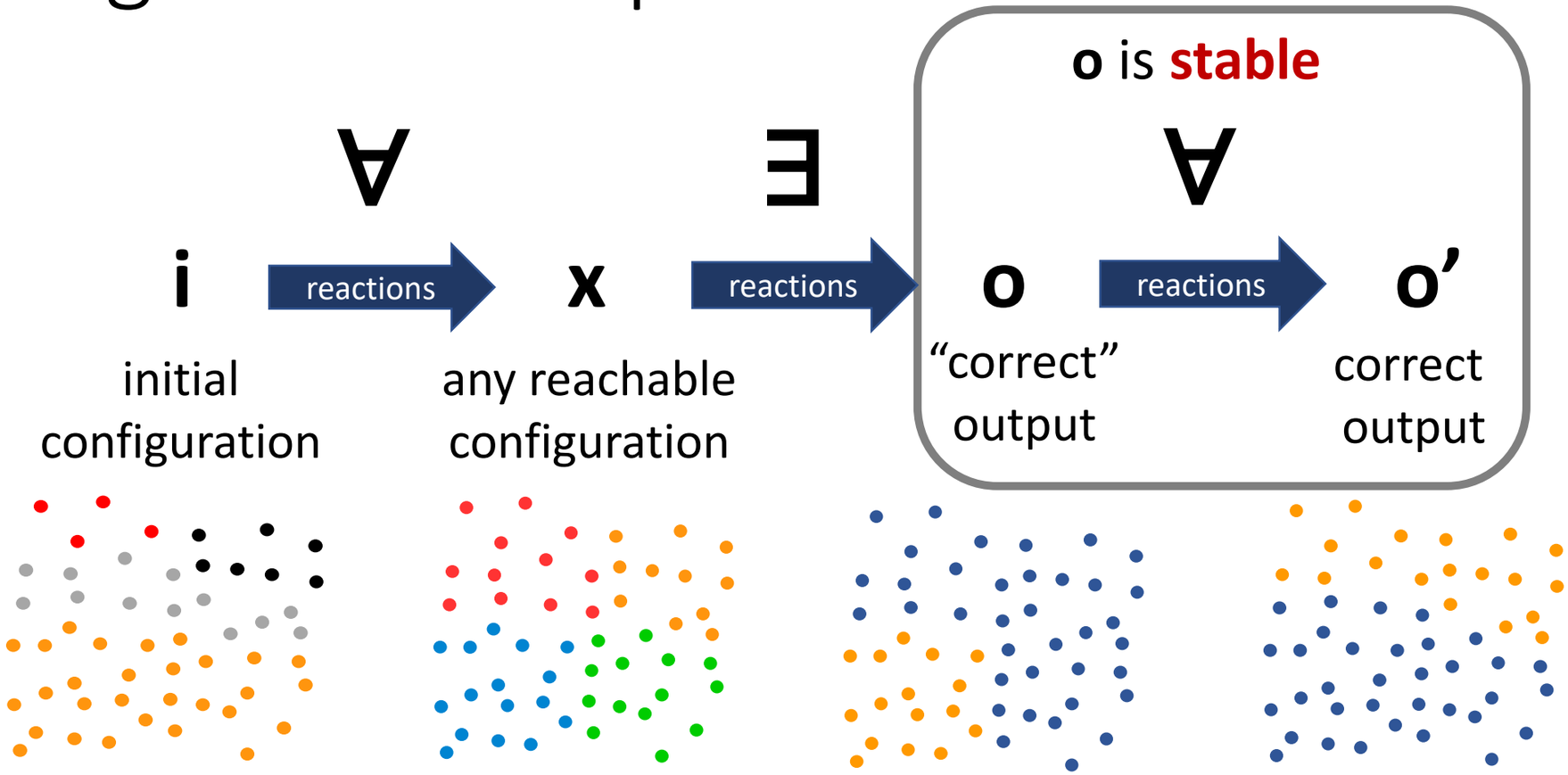
Defining **stable** computation



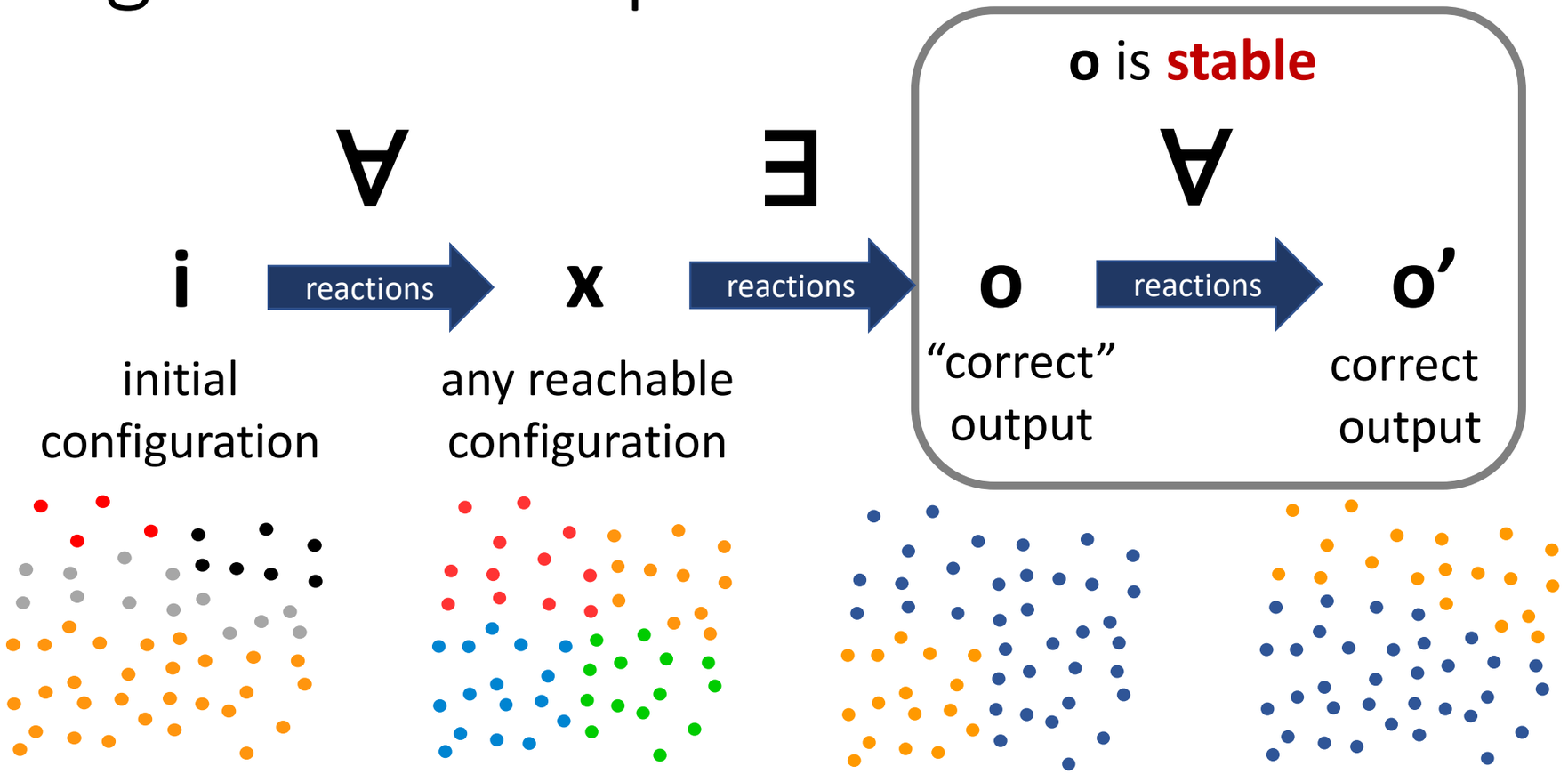
Defining **stable** computation



Defining **stable** computation



Defining **stable** computation



(assuming finite set of reachable configurations) equivalent to:
The system will reach a correct stable configuration with probability 1.

Speed of computation

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

Speed of computation

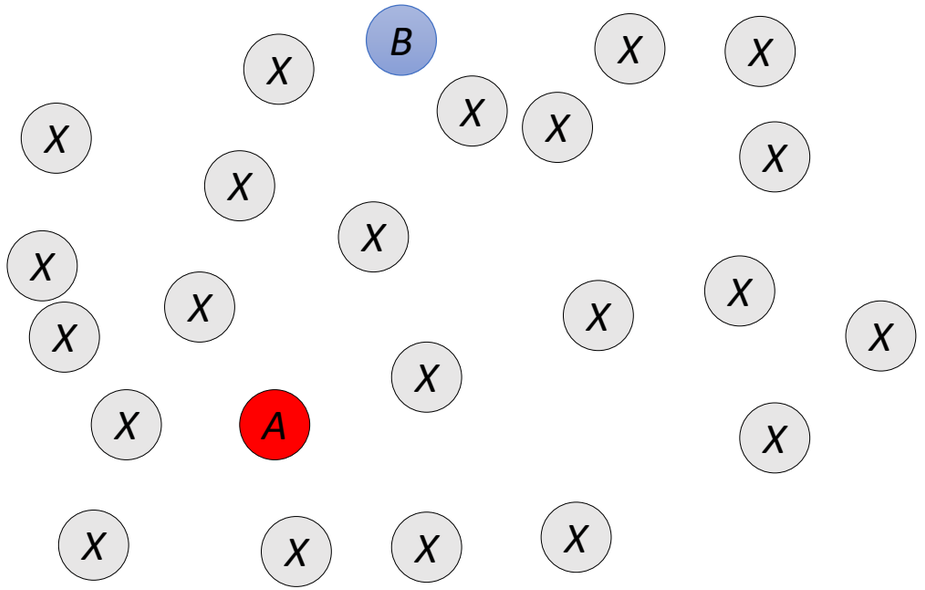
How to fairly assess speed?

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

n = total molecular count
volume $v = O(n)$
i.e., require bounded concentration (finite density constraint)

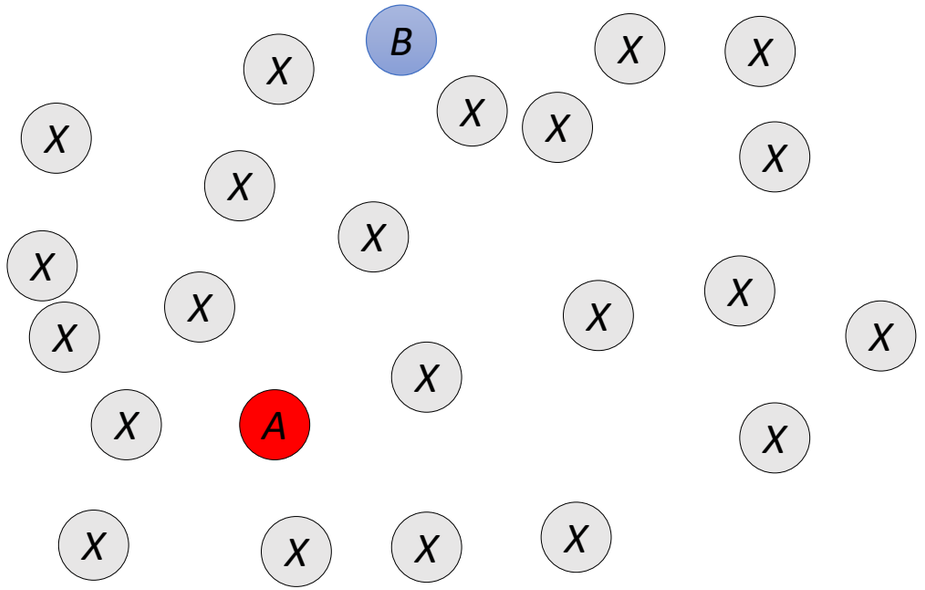
An exponential time difference

n molecules
volume $v = O(n)$



An exponential time difference

n molecules
volume $v = O(n)$



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

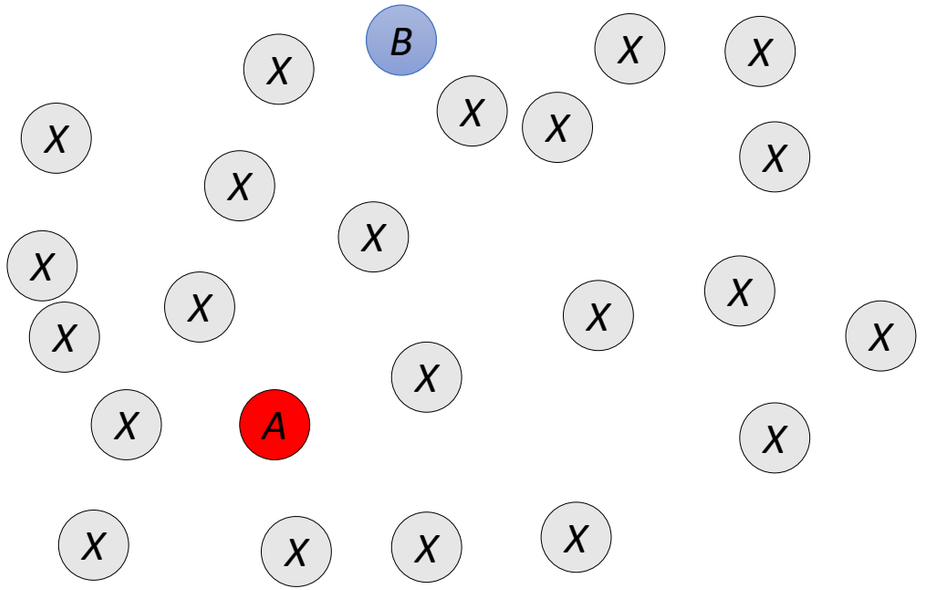
propensity: $\#A \cdot \#B \,/\, v = O(1/n)$

expected time to produce *Y*:

$O(n)$

An exponential time difference

n molecules
volume $v = O(n)$



$A+B \rightarrow Y+B$

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

$B+X \rightarrow B+B$

$A+B \rightarrow Y+B$

distributed computing terms:
- *epidemic*
- *rumor/gossip spreading*

chemical term:
- *autocatalysis*

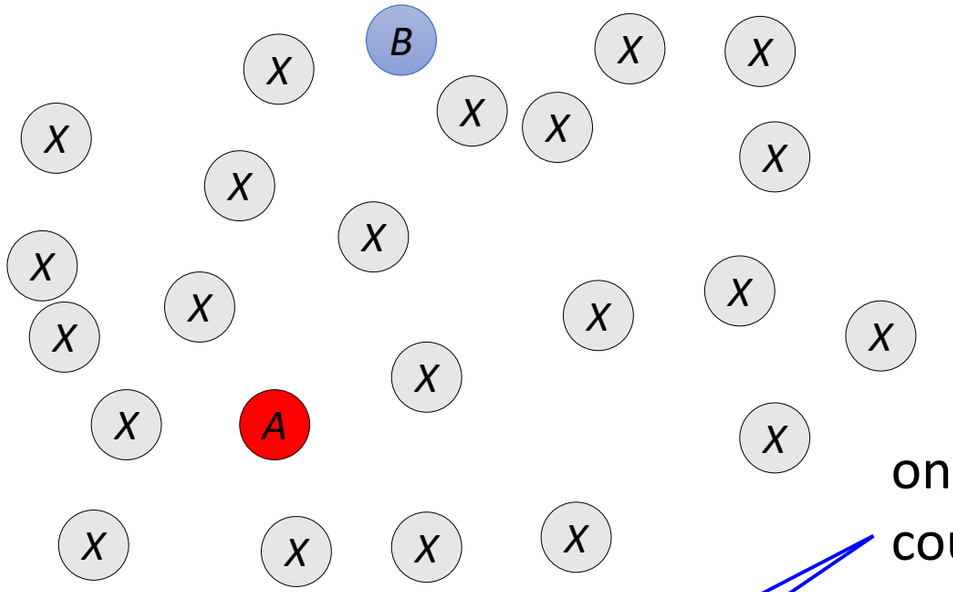
expected time to produce Y:

$O(n)$

$O(\log n)$

An exponential time difference

n molecules
volume $v = O(n)$



one of these is always
count ≥ $n/2$

$A+B \rightarrow Y+B$

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

$B+X \rightarrow B+B$

$A+B \rightarrow Y+B$

distributed computing terms:
- *epidemic*
- *rumor/gossip spreading*

chemical term:
- *autocatalysis*

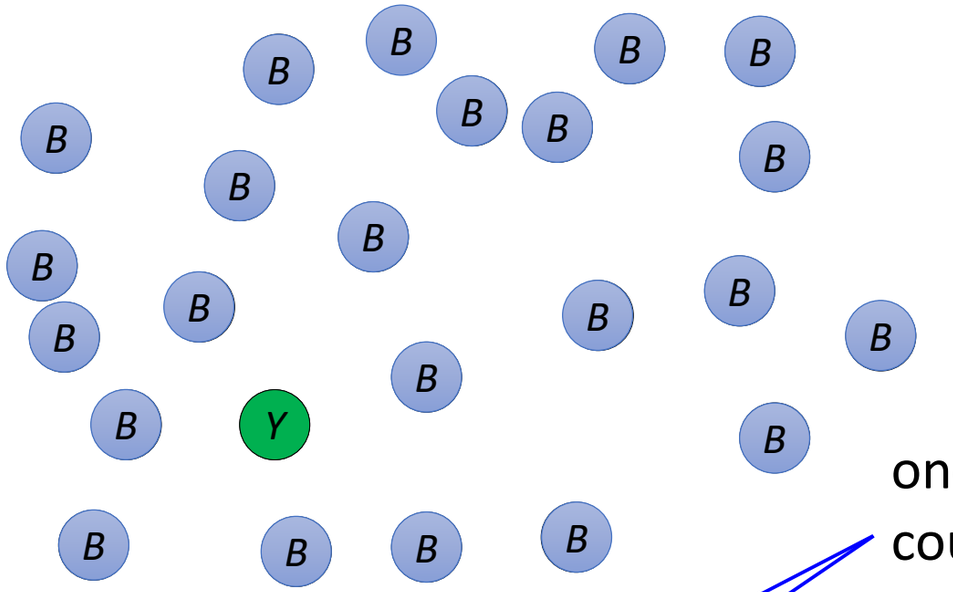
expected time to produce *Y*:

$O(n)$

$O(\log n)$

An exponential time difference

n molecules
volume $v = O(n)$



one of these is always
count ≥ $n/2$

$A+B \rightarrow Y+B$

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

$B+X \rightarrow B+B$
$A+B \rightarrow Y+B$

distributed computing terms:
- *epidemic*
- *rumor/gossip spreading*

chemical term:
- *autocatalysis*

expected time to produce Y:

$O(n)$

$O(\log n)$

Definition of *predicate* (decision problem) computation

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

[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 $\Leftrightarrow$ $a>b$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - in valid initial configurations 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., $\varphi(a,b)=Y \iff a>b$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - in valid initial configurations 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., $\varphi(a,b)=Y \iff a>b$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - in valid initial configurations 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
  - $\psi(\mathbf{o}) = Y$ (configuration **o** outputs "yes") if vote is unanimously yes: $\mathbf{o}(s)>0 \Leftrightarrow s \in \Lambda_Y$
  - $\psi(\mathbf{o}) = N$ (configuration **o** outputs "no") if vote is unanimously no: $\mathbf{o}(s)>0 \Leftrightarrow s \in \Lambda_N$
  - configuration **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]

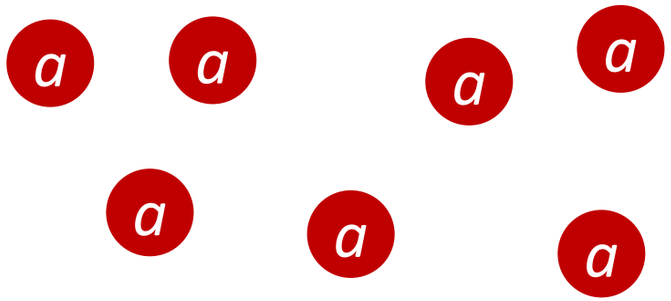
Definition of *predicate* (decision problem) computation

- goal: compute predicate φ: $\mathbb{N}^k \to \{Y,N\}$, e.g., $\varphi(a,b)=Y \iff a>b$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - in valid initial configurations 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
  - $\psi(\mathbf{o}) = Y$ (configuration **o** outputs "yes") if vote is unanimously yes: $\mathbf{o}(s)>0 \iff s \in \Lambda_Y$
  - $\psi(\mathbf{o}) = N$ (configuration **o** outputs "no") if vote is unanimously no: $\mathbf{o}(s)>0 \iff s \in \Lambda_N$
  - configuration **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: $\varphi(a,b) = \text{Y} \Leftrightarrow b > 0$

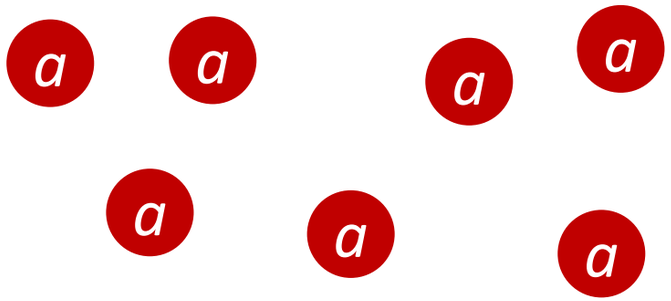


Examples of predicate computation

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

$$b+a \rightarrow b+b$$

a votes no; b votes yes
E[time] = O(log n)

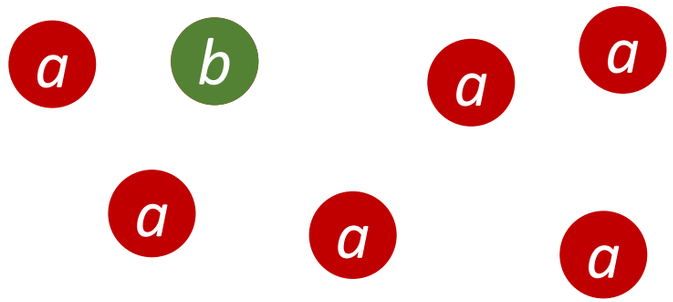


Examples of predicate computation

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

$$b+a \rightarrow b+b$$

a votes no; *b* votes yes
E[time] = O(log *n*)

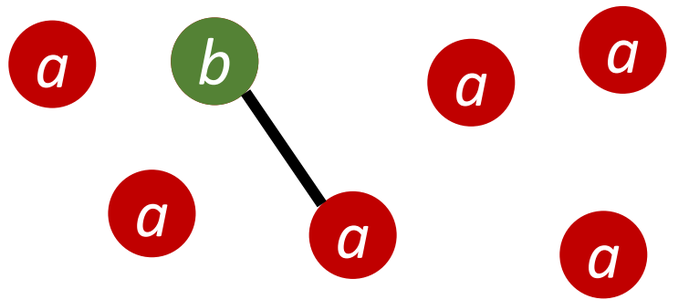


Examples of predicate computation

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

$$b+a \rightarrow b+b$$

a votes no; *b* votes yes
E[time] = O(log *n*)

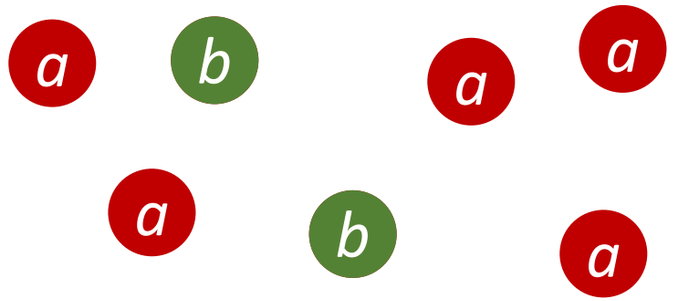


Examples of predicate computation

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

$$b+a \rightarrow b+b$$

a votes no; *b* votes yes
E[time] = O(log *n*)

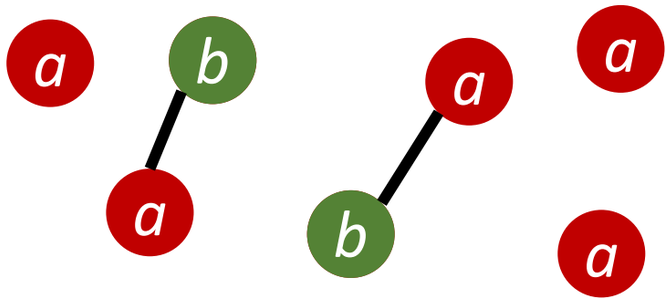


Examples of predicate computation

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

$$b+a \rightarrow b+b$$

a votes no; *b* votes yes
E[time] = O(log *n*)

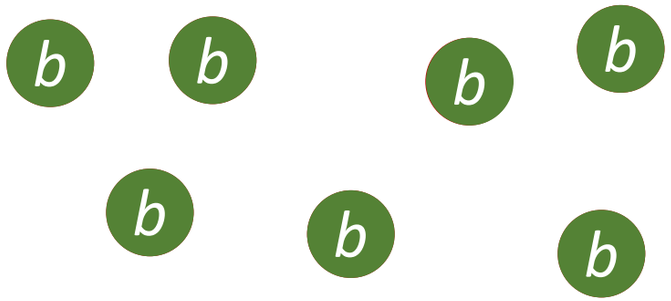


Examples of predicate computation

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

$$b+a \rightarrow b+b$$

a votes no; *b* votes yes
E[time] = O(log *n*)



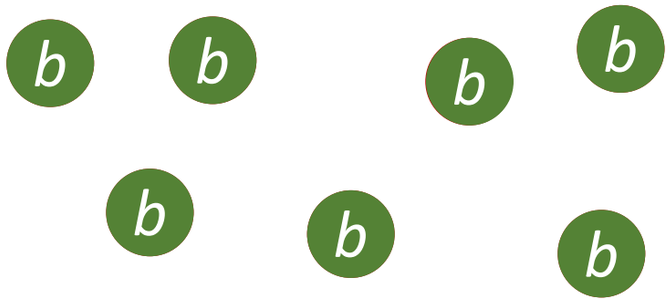
Examples of predicate computation

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

Counting: $\varphi(a,b) = Y \Leftrightarrow b > 1$

$$b+a \rightarrow b+b$$

a votes no; *b* votes yes
E[time] = O(log *n*)

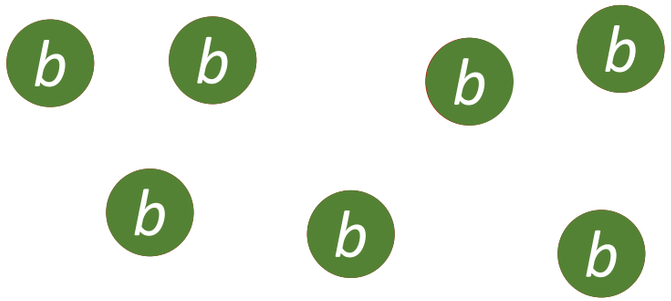


Examples of predicate computation

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

$$b+a \rightarrow b+b$$

a votes no; b votes yes
E[time] = O(log n)



Counting: $\varphi(a,b) = Y \Leftrightarrow b > 1$

$b+b \rightarrow y+y$ $O(n)$ time

$y+b \rightarrow y+y$ $O(\log n)$ time

$y+a \rightarrow y+y$ $O(\log n)$ time

a,b vote no; y votes yes
E[time] = O(n)

Examples of predicate computation

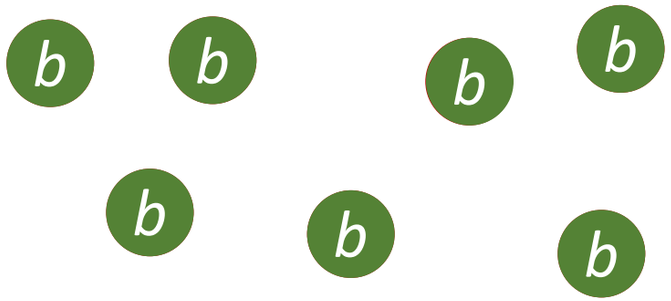
Detection: $\varphi(a,b) = Y \Leftrightarrow b > 0$

$b+a \rightarrow b+b$

a votes no; *b* votes yes

E[time] = O(log *n*)

exponential time difference!



Counting: $\varphi(a,b) = Y \Leftrightarrow b > 1$

$b+b \rightarrow y+y$ $O(n)$ time

$y+b \rightarrow y+y$ $O(\log n)$ time

$y+a \rightarrow y+y$ $O(\log n)$ time

a,b vote no; *y* votes yes

E[time] = O(*n*)

Examples of predicate computation

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

Examples of predicate computation

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

$A+B \rightarrow a+b$ (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: $\varphi(a,b) = Y \Leftrightarrow a > b$

$A+B \rightarrow a+b$ (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: $\varphi(a,b) = Y \Leftrightarrow a > b$

$A+B \rightarrow a+b$ (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)

$O(n)$ time if $a - b = O(1)$

$O(\log n)$ time if $a - b > 0.01n$

[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

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

Examples of predicate computation

Parity: $\varphi(a)$=Y ⇔ a is odd

$a = A_o$ (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

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

$A_o + A_o \rightarrow A_e + a_e$

$A_e + A_e \rightarrow A_e + a_e$

$A_o + A_e \rightarrow A_o + a_o$

two leaders XOR their parity, and one becomes follower

Examples of predicate computation

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

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

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

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

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

$A_o + A_o \to A_e + a_e$
$A_e + A_e \to A_e + a_e$
$A_o + A_e \to A_o + a_o$

two leaders XOR their parity, and one becomes follower

$A_o + a_e \to A_o + a_o$
$A_e + a_o \to A_e + a_e$

leader overwrites bit of follower

$O(n \log n)$ time

$O(n)$ bottleneck transition before leader reaches count 1, then $O(n \log n)$ coupon collector for leader to encounter each follower

possible to optimize to $O(n)$
[Angluin, Aspnes, Eisenstat, DISC 2006]

Definition of function computation

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

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: all molecules are from Σ, e.g., {100 a, 100 b}

Definition of function computation

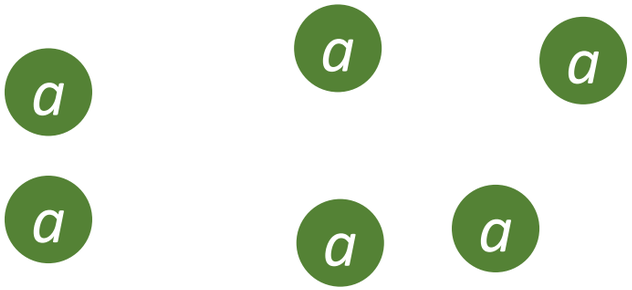
- goal: compute function f: $\mathbb{N}^k \to \mathbb{N}$, e.g., $f(a,b) = 2a + b/2$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - valid initial configuration: all molecules are from Σ, e.g., {100 a, 100 b}
- output specification: designate one species $y \in \Lambda$ whose count is the *output*

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: all molecules are from Σ, e.g., {100 *a*, 100 *b*}
- output specification: designate one species $y \in \Lambda$ whose count is the *output*
- recall: **o** is stable if **o**(y) = **o'**(y) for all **o'** reachable from **o**

Examples of function computation

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

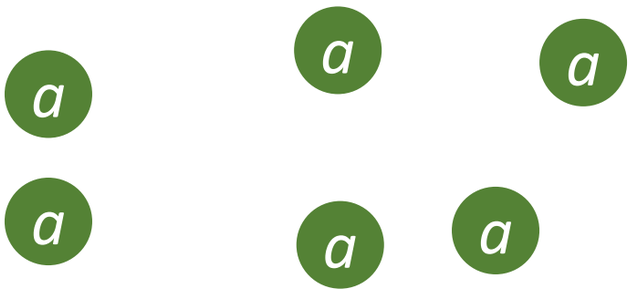


Examples of function computation

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

$a + a \rightarrow y$

E[time] = $O(n)$
(last transition is bottleneck: has count ≤ 3 of a)

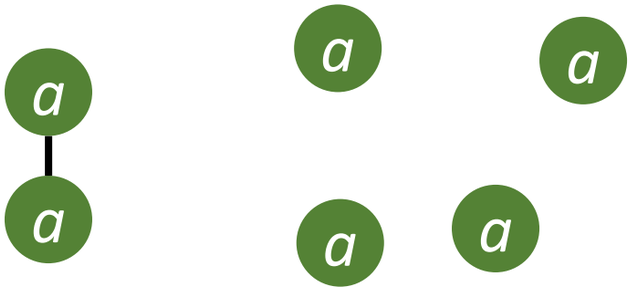


Examples of function computation

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

$a + a \rightarrow y$

E[time] = $O(n)$
(last transition is bottleneck: has count ≤ 3 of a)

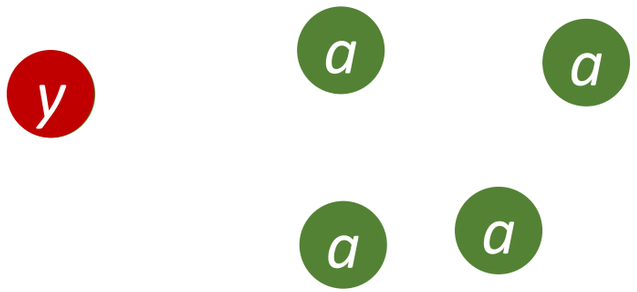


Examples of function computation

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

$a+a \rightarrow y$

E[time] = $O(n)$
(last transition is bottleneck: has count ≤ 3 of a)

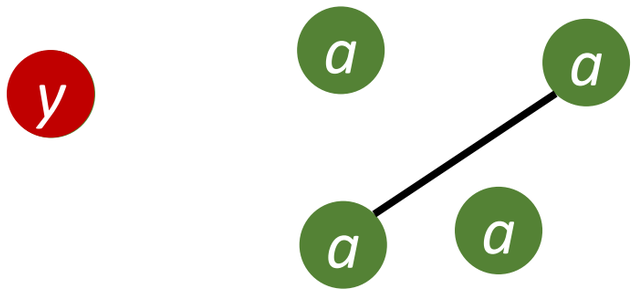


Examples of function computation

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

$a+a \rightarrow y$

E[time] = $O(n)$
(last transition is bottleneck: has count ≤ 3 of a)

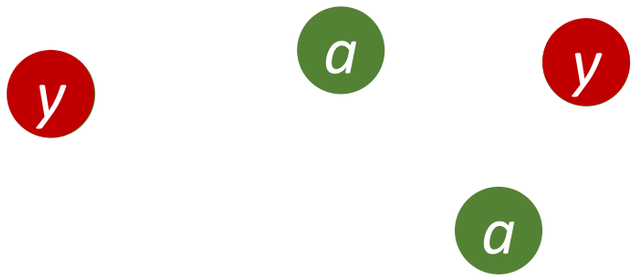


Examples of function computation

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

$a+a \rightarrow y$

E[time] = $O(n)$
(last transition is bottleneck: has count ≤ 3 of a)

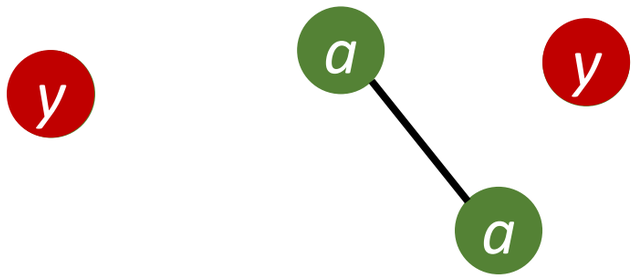


Examples of function computation

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

$a+a \rightarrow y$

E[time] = $O(n)$
(last transition is bottleneck: has count ≤ 3 of a)



Examples of function computation

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

$a+a \rightarrow y$

E[time] = $O(n)$
(last transition is bottleneck: has count ≤ 3 of a)

Examples of function computation

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

$a+a \rightarrow y$

E[time] = $O(n)$
(last transition is bottleneck: has count ≤ 3 of a)

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

Examples of function computation

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

$a+a \rightarrow y$

E[time] = $O(n)$
(last transition is bottleneck: has count ≤ 3 of a)

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

$a \rightarrow y+y$

E[time] = $O(\log n)$

Examples of function computation

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

$a+a \to y$

E[time] = $O(n)$
(last transition is bottleneck: has count ≤ 3 of a)

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

$a \to y+y$

E[time] = $O(\log n)$

Examples of function computation

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

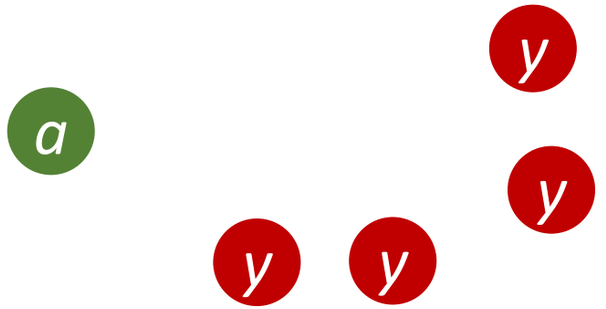
$a+a \to y$

E[time] = $O(n)$
(last transition is bottleneck: has count ≤ 3 of a)

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

$a \to y+y$

E[time] = $O(\log n)$



Examples of function computation

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

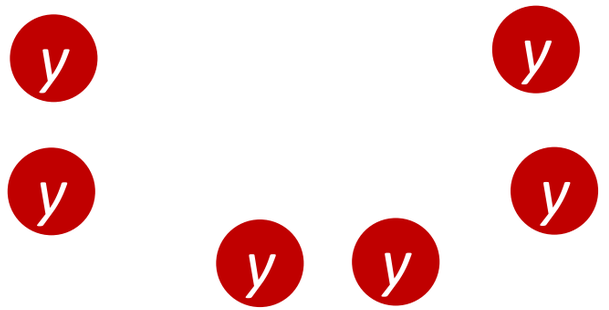
$a+a \to y$

E[time] = $O(n)$
(last transition is bottleneck: has count ≤ 3 of a)

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

$a \to y+y$

E[time] = $O(\log n)$



Examples of function computation

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

$a+a \to y$

E[time] = $O(n)$

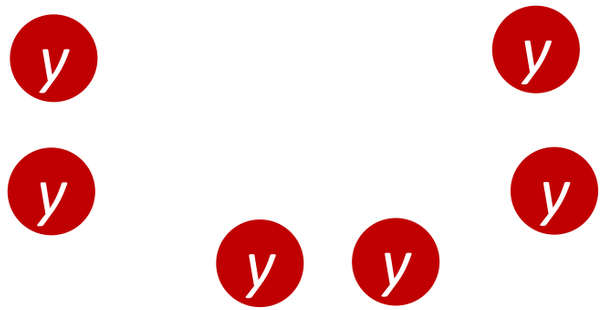
(last transition is bottleneck: has count ≤ 3 of a)

exponential time difference!

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

$a \to y+y$

E[time] = $O(\log n)$



Examples of function computation

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

Examples of function computation

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

$a \rightarrow y$

$b \rightarrow y$

E[time] = $O(\log n)$

Examples of function computation

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

$a \rightarrow y$

$b \rightarrow y$

E[time] = $O(\log n)$

subtraction: $f(a,b) = a-b$

Examples of function computation

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

$a \rightarrow y$

$b \rightarrow y$

E[time] = $O(\log n)$

subtraction: $f(a,b) = a-b$

$a \rightarrow y$

$b+y \rightarrow$

E[time] = $O(n)$ if $a - b = O(1)$ initially
(last transition is bottleneck)

Examples of function computation

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

Examples of function computation

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

$$a + a \rightarrow a + y$$

E[time] = $O(n)$

Examples of function computation

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

$$a+a \rightarrow a+y$$

E[time] = $O(n)$

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

Examples of function computation

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

$$a+a \rightarrow a+y$$

E[time] = $O(n)$

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

$$a+b \rightarrow y$$

E[time] = $O(n)$ if $a - b = O(1)$ initially

Examples of function computation

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

$a+a \rightarrow a+y$

E[time] = $O(n)$

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

$a+b \rightarrow y$

E[time] = $O(n)$ if $a - b = O(1)$ initially

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

Examples of function computation

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

$a+a \rightarrow a+y$

E[time] = $O(n)$

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

$a+b \rightarrow y$

E[time] = $O(n)$ if $a - b = O(1)$ initially

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

$a \rightarrow y+a_2$

$b \rightarrow y+b_2$

addition

Examples of function computation

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

$a+a \rightarrow a+y$

E[time] = $O(n)$

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

$a+b \rightarrow y$

E[time] = $O(n)$ if $a - b = O(1)$ initially

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

$a \rightarrow y+a_2$
$b \rightarrow y+b_2$
addition

$a_2+b_2 \rightarrow k$ minimum

Examples of function computation

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

$a+a \rightarrow a+y$

E[time] = $O(n)$

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

$a+b \rightarrow y$

E[time] = $O(n)$ if $a - b = O(1)$ initially

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

$a \rightarrow y+a_2$
$b \rightarrow y+b_2$ addition

$a_2+b_2 \rightarrow k$ minimum

$k+y \rightarrow$ subtraction

E[time] = $O(n)$ (uses minimum and subtraction as "subroutines")

Examples of function computation

constant: $f(a) = 1$

Examples of function computation

constant: $f(a) = 1$

$$a \rightarrow y$$
$$y+y \rightarrow y$$

E[time] = O(n)

Examples of function computation

constant: $f(a) = 1$

$a \to y$

$y+y \to y$

E[time] = O(n)

a.k.a. “leader election”

Limits of stable computation (unbounded time)

Limits of stable computation (unbounded time)

Predicates

- φ is stably computable if and only if φ is *semilinear*.

[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]

Limits of stable computation (unbounded time)

Predicates

- φ is stably computable if and only if φ is *semilinear*.
- semilinear = Boolean combination of threshold and mod predicates: take weighted sum $s = w_1 \cdot a_1 + \ldots w_k \cdot a_k$ of inputs and ask if

  $s >$ constant c?

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

$a>b$?	$a=b$?	a is odd?	$a>0$?	$a>1$?

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]

Limits of stable computation (unbounded time)

Predicates

- φ is stably computable if and only if φ is *semilinear*.
- semilinear = Boolean combination of threshold and mod predicates: take weighted sum $s = w_1 \cdot a_1 + \ldots w_k \cdot a_k$ of inputs and ask if

  $s >$ constant c?

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

$a>b$?	$a=b$?	a is odd?	$a>0$?	$a>1$?

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]

Functions

- f is stably computable if and only if graph(f) = { ($\boldsymbol{a}$,y) | $f(\boldsymbol{a})=y$ } is semilinear.

[Chen, Doty, Soloveichik, Deterministic function computation with chemical reaction networks, *DNA* 2012]
[Doty, Hajiaghayi, Leaderless deterministic chemical reaction networks, *DNA* 2013]

Limits of stable computation (unbounded time)

Predicates

- φ is stably computable if and only if φ is *semilinear*.
- semilinear = Boolean combination of threshold and mod predicates: take weighted sum $s = w_1 \cdot a_1 + \ldots w_k \cdot a_k$ of inputs and ask if

  $s >$ constant c?

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

$a>b$?	$a=b$?	a is odd?	$a>0$?	$a>1$?

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]

Functions

- f is stably computable if and only if graph(f) = { ($\boldsymbol{a}$,y) | $f(\boldsymbol{a})=y$ } is semilinear.
- piecewise linear, with semilinear predicate to determine which piece.

$a+b$	$a-b$	$2a$	$a/2$	$\min(a,b)$	$a+1$	$a-1$

$f(a) = 2a-b/3$ if $a+b$ is odd, else $f(a) = a/4+5b$

NOT	a^2	2^a	$2a$ if a is prime, else $3a$

[Chen, Doty, Soloveichik, Deterministic function computation with chemical reaction networks, *DNA* 2012]
[Doty, Hajiaghayi, Leaderless deterministic chemical reaction networks, *DNA* 2013]

Limits of stable computation (unbounded time)

Predicates

- φ is stably computable if and only if φ is *semilinear*.
- semilinear = Boolean combination of threshold and mod predicates: take weighted sum $s = w_1 \cdot a_1 + \ldots w_k \cdot a_k$ of inputs and ask if

  $s >$ constant c?

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

$a>b$?	$a=b$?	a is odd?	$a>0$?	$a>1$?

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]

Functions

- f is stably computable if and only if graph(f) = { ($\boldsymbol{a}$,y) | $f(\boldsymbol{a})=y$ } is semilinear.
- piecewise linear, with semilinear predicate to determine which piece.

$a+b$	$a-b$	$2a$	$a/2$	$\min(a,b)$	$a+1$	$a-1$

$f(a) = 2a-b/3$ if $a+b$ is odd, else $f(a) = a/4+5b$

NOT	a^2	2^a	$2a$ if a is prime, else $3a$

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

[Chen, Doty, Soloveichik, Deterministic function computation with chemical reaction networks, *DNA* 2012]
[Doty, Hajiaghayi, Leaderless deterministic chemical reaction networks, *DNA* 2013]

What is known to be computable in less than time $O(n)$?

What is known to be computable in less than time $O(n)$?

Predicates

Boolean combination of detection predicates

"detection" means $\varphi(a)$ = "$a > 0$?"

What is known to be computable in less than time $O(n)$?

Predicates

Boolean combination of detection predicates

"detection" means $\varphi(a)$ = "$a > 0$?"

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

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

What is known to be computable in less than time $O(n)$?

Predicates

Boolean combination of detection predicates

"detection" means $\varphi(a)$ = "$a > 0$?"

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

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

Functions

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

What is known to be computable in less than time $O(n)$?

Predicates

Boolean combination of detection predicates

"detection" means $\varphi(a)$ = "$a > 0$?"

$\varphi(a,b,c) = a>0$ OR ($b>0$ 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., $f(a,b) = 2a + 3b$

$a \to y+y$

$b \to y+y+y$

[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: “most” predicates/functions

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

[Doty, Soloveichik, Stable leader election in population protocols requires linear time, *DISC* 2015]

[Belleville, Doty, Soloveichik, Hardness of computing and approximating predicates and functions with leaderless population protocols, *ICALP* 2017]

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 $O(n)$ time to compute
- Definition: $\varphi: \mathbb{N}^k \to \{Y,N\}$ is eventually constant if there is $m \in \mathbb{N}$ so that $\varphi(\boldsymbol{a}) = \varphi(\boldsymbol{b})$ for all $\boldsymbol{a},\boldsymbol{b}$ with all components $\geq m$

[Doty, Soloveichik, Stable leader election in population protocols requires linear time, *DISC* 2015]

[Belleville, Doty, Soloveichik, Hardness of computing and approximating predicates and functions with leaderless population protocols, *ICALP* 2017]

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 $O(n)$ time to compute
- Definition: φ: $\mathbb{N}^k \to \{Y,N\}$ is eventually constant if there is $m \in \mathbb{N}$ so that $\varphi(\boldsymbol{a}) = \varphi(\boldsymbol{b})$ for all $\boldsymbol{a},\boldsymbol{b}$ with all components $\geq m$
- Definition: f: $\mathbb{N}^k \to \mathbb{N}$ is eventually $\mathbb{N}$-linear if there is $m \in \mathbb{N}$ so that $f(\boldsymbol{a})$ is $\mathbb{N}$-linear for all $\boldsymbol{a}$ with all components $\geq m$
  - Both definitions allow exceptions “near a face of $\mathbb{N}^k$”

[Doty, Soloveichik, Stable leader election in population protocols requires linear time, *DISC* 2015]

[Belleville, Doty, Soloveichik, Hardness of computing and approximating predicates and functions with leaderless population protocols, *ICALP* 2017]

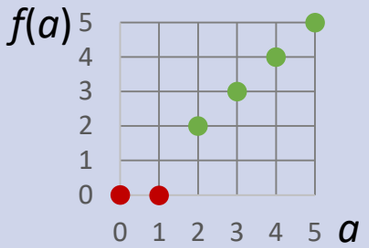
Known time lower bounds: "most" predicates/functions

- Informal: "most" semilinear predicates and functions not known to be computable in $o(n)$ time, actually require at least $O(n)$ time to compute
- Definition: $\varphi: \mathbb{N}^k \to \{Y,N\}$ is eventually constant if there is $m \in \mathbb{N}$ so that $\varphi(\boldsymbol{a}) = \varphi(\boldsymbol{b})$ for all $\boldsymbol{a},\boldsymbol{b}$ with all components $\geq m$
- Definition: $f: \mathbb{N}^k \to \mathbb{N}$ is eventually $\mathbb{N}$-linear if there is $m \in \mathbb{N}$ so that $f(\boldsymbol{a})$ is $\mathbb{N}$-linear for all $\boldsymbol{a}$ 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 $O(n)$ to compute.
  - They're all computable in at most $O(n)$ time, so this settles their time complexity.

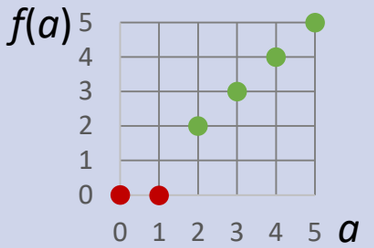
[Doty, Soloveichik, Stable leader election in population protocols requires linear time, *DISC* 2015]

[Belleville, Doty, Soloveichik, Hardness of computing and approximating predicates and functions with leaderless population protocols, *ICALP* 2017]

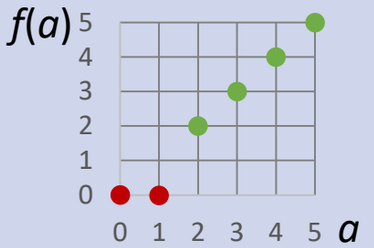
What is currently known/unknown

	Predicates	Functions
computable in $O(\log n)$ time	detection $a>0$ **AND** ($b>0$ **OR** $c=0$)	$\mathbb{N}$-linear $3a + b + 2c$
not computable in less than $O(n)$ time	non-eventually constant $a>b$? $a=b$? a is odd?	non-eventually $\mathbb{N}$-linear $a/2$ $a-b$ $a+1$ $a-1$ 1 $\min(a,b)$ $\max(a,b)$ $\max(a, \min(b + 3, 2c)) - c - 1$
unknown (best known protocol is $O(n)$ time)	eventually constant but not constant on positive values $a>1$?	eventually $\mathbb{N}$-linear but not $\mathbb{N}$-linear $f(a) = \begin{cases} a \text{ if } a>1, \\ 0 \text{ otherwise} \end{cases}$ 

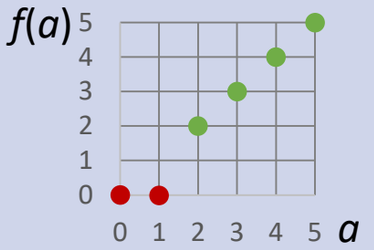
What is currently known/unknown

	Predicates	Functions
computable in $O(\log n)$ time	detection $a>0$ **AND** ($b>0$ **OR** $c=0$)	$\mathbb{N}$-linear $3a + b + 2c$
not computable in less than $O(n)$ time	non-eventually constant $a>b$? $a=b$? a is odd?	non-eventually $\mathbb{N}$-linear $a/2$ $a-b$ $a+1$ $a-1$ 1 $\min(a,b)$ $\max(a,b)$ $\max(a, \min(b + 3, 2c)) - c - 1$
unknown (best known protocol is $O(n)$ time)	eventually constant but not constant on positive values $a>1$?	eventually $\mathbb{N}$-linear but not $\mathbb{N}$-linear $f(a) = \begin{cases} a \text{ if } a>1, \\ 0 \text{ otherwise} \end{cases}$ 

What is currently known/unknown

	Predicates	Functions
computable in $O(\log n)$ time	detection $a>0$ **AND** ($b>0$ **OR** $c=0$)	$\mathbb{N}$-linear $3a + b + 2c$
not computable in less than $O(n)$ time	non-eventually constant $a>b$? $a=b$? a is odd?	non-eventually $\mathbb{N}$-linear $a/2$ $a-b$ $a+1$ $a-1$ 1 $\min(a,b)$ $\max(a,b)$ $\max(a, \min(b + 3, 2c)) - c - 1$
unknown (best known protocol is $O(n)$ time)	eventually constant but not constant on positive values $a>1$?	eventually $\mathbb{N}$-linear but not $\mathbb{N}$-linear $f(a) = \begin{cases} a \text{ if } a>1, \\ 0 \text{ otherwise} \end{cases}$ 

What is currently known/unknown

	Predicates	Functions
computable in $O(\log n)$ time	detection $a>0$ **AND** ($b>0$ **OR** $c=0$)	$\mathbb{N}$-linear $3a + b + 2c$
not computable in less than $O(n)$ time	non-eventually constant $a>b$? $a=b$? a is odd?	non-eventually $\mathbb{N}$-linear $a/2$ $a-b$ $a+1$ $a-1$ 1 $\min(a,b)$ $\max(a,b)$ $\max(a, \min(b + 3, 2c)) - c - 1$
unknown (best known protocol is $O(n)$ time)	eventually constant but not constant on positive values $a>1$?	eventually $\mathbb{N}$-linear but not $\mathbb{N}$-linear $f(a) = \begin{cases} a \text{ if } a>1, \\ 0 \text{ otherwise} \end{cases}$ 

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 talk

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?

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* }

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*)

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: $\varphi(a)$ = “a is odd”

without a leader

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

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: $\varphi(a)$ = “a is odd”

without a leader

$A_o + A_o \rightarrow A_e + a_e$

$A_e + A_e \rightarrow A_e + a_e$

$A_o + A_e \rightarrow A_o + a_o$

$A_o + a_e \rightarrow A_o + a_o$

$A_e + a_o \rightarrow A_e + a_e$

with a leader L_e

$L_e + A \rightarrow L_o$

$L_o + A \rightarrow L_e$

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: $\varphi(a)$ = "a is odd"

without a leader

$A_o + A_o \to A_e + a_e$
$A_e + A_e \to A_e + a_e$
$A_o + A_e \to A_o + a_o$
$A_o + a_e \to A_o + a_o$
$A_e + a_o \to A_e + a_e$

with a leader L_e

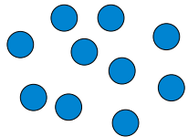
$L_e + A \to L_o$
$L_o + A \to L_e$

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, Hajiaghayi, *DNA* 2013]

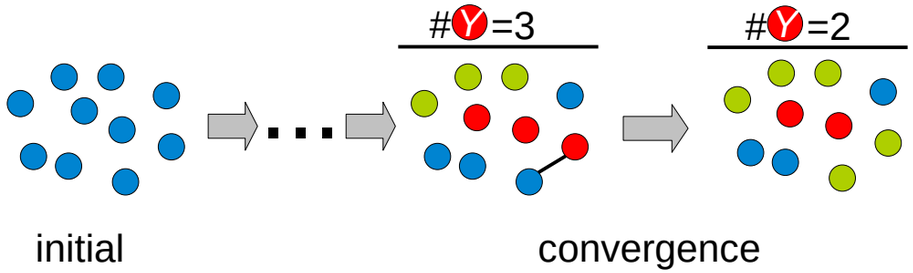
Convergence vs stabilization and leader vs anarchy

Convergence vs stabilization and leader vs anarchy

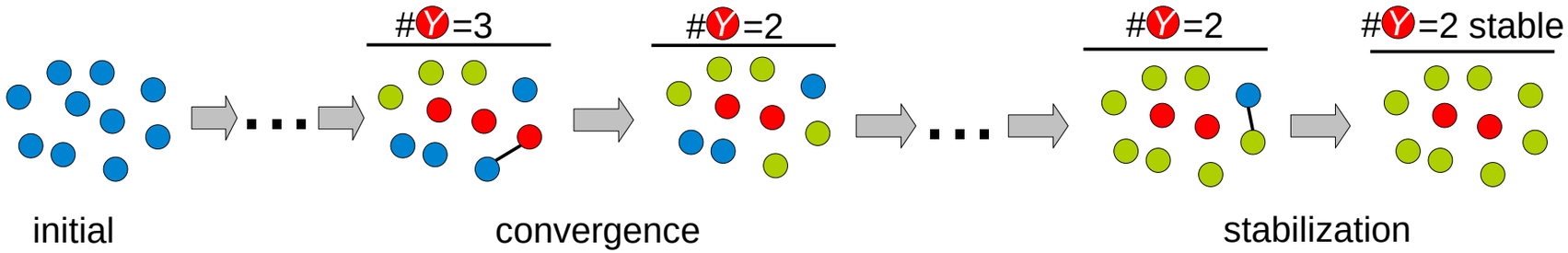


initial

Convergence vs stabilization and leader vs anarchy

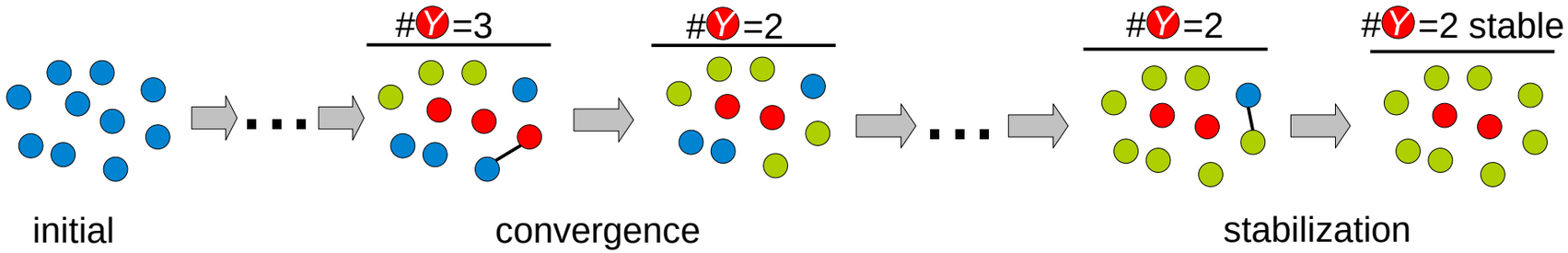


Convergence vs stabilization and leader vs anarchy



Theorem: Without a leader, all non-eventually constant predicates and non-eventually-$\mathbb{N}$-linear functions require at least $O(n)$ stabilization time. [Belleville, Doty, Soloveichik, *ICALP* 2017]

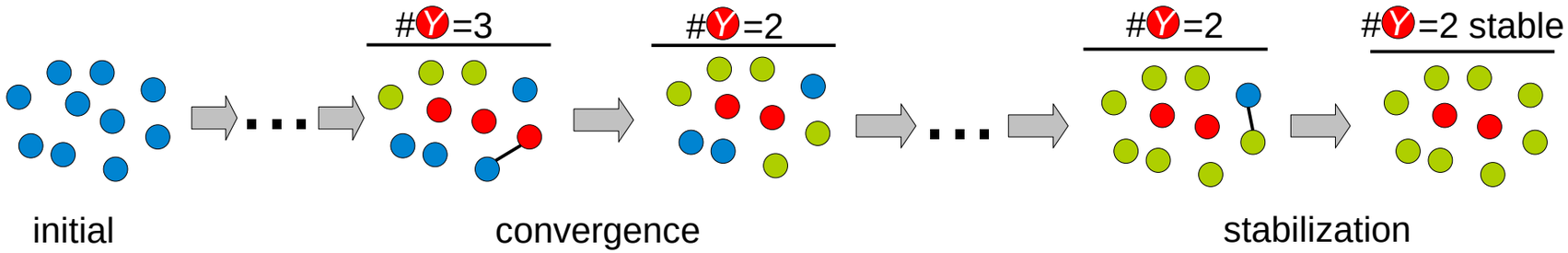
Convergence vs stabilization and leader vs anarchy



Theorem: Without a leader, all non-eventually constant predicates and non-eventually-$\mathbb{N}$-linear functions require at least $O(n)$ 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]

Convergence vs stabilization and leader vs anarchy



Theorem: Without a leader, all non-eventually constant predicates and non-eventually-$\mathbb{N}$-linear functions require at least $O(n)$ 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 $O(n)$ stabilization time.

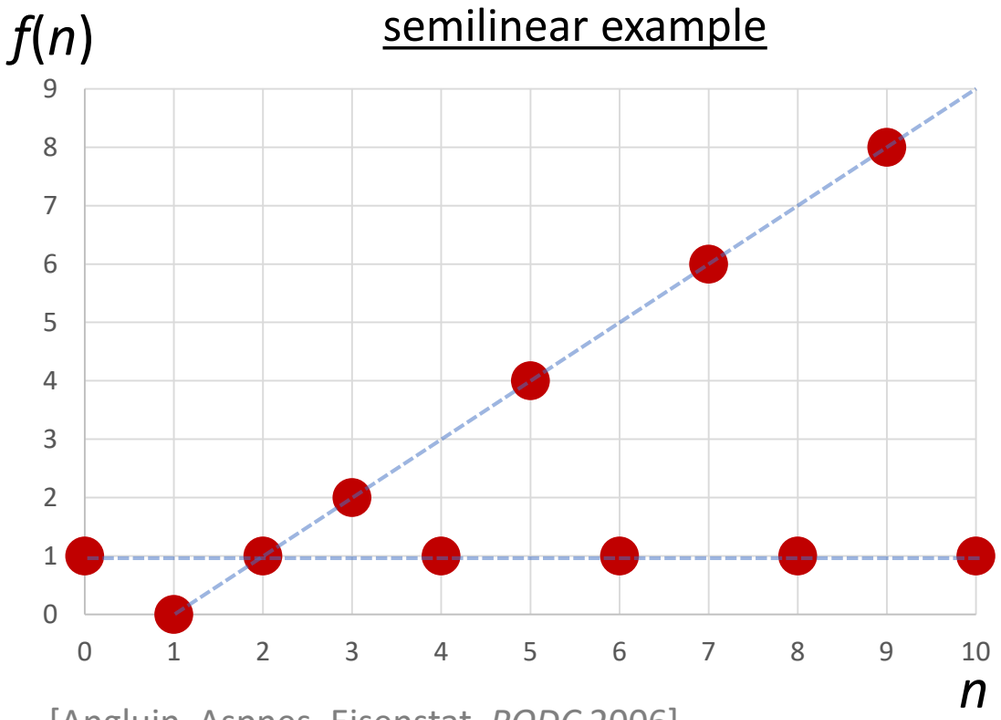
Conjecture: Without a leader, all non-detection predicates and non-$\mathbb{N}$-linear functions require at least $O(n)$ convergence time.

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.

What if we use real-valued concentrations?

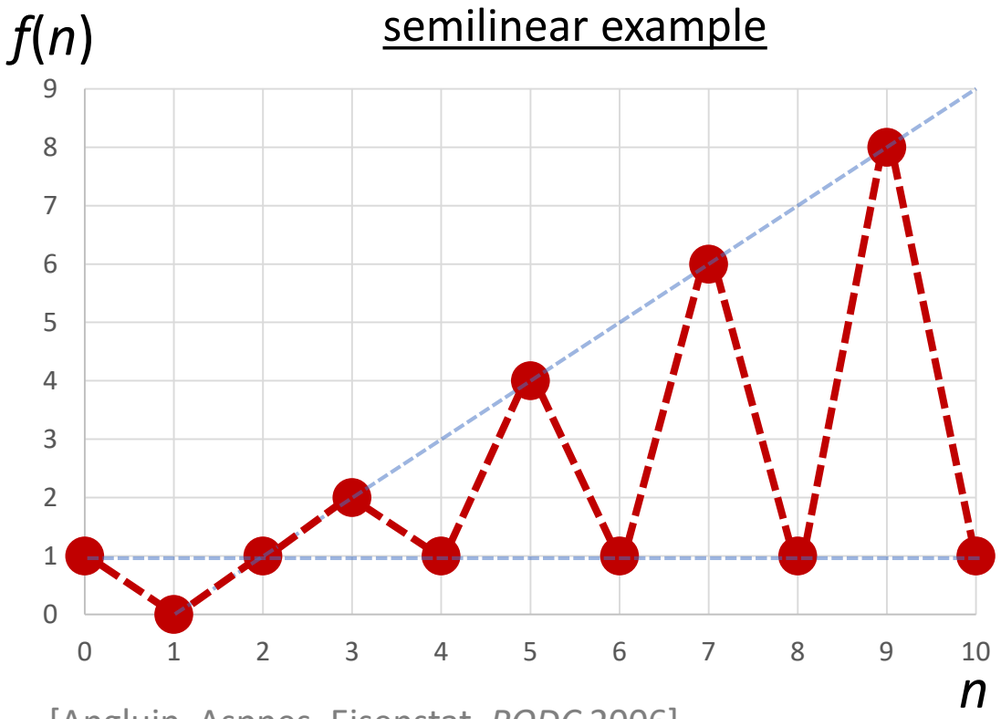
Theorem: A function is stably computable by an integer-valued chemical reaction network if and only if it is semilinear.



[Angluin, Aspnes, Eisenstat, *PODC* 2006]
[Chen, Doty, Soloveichik, *DNA* 2012]

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.

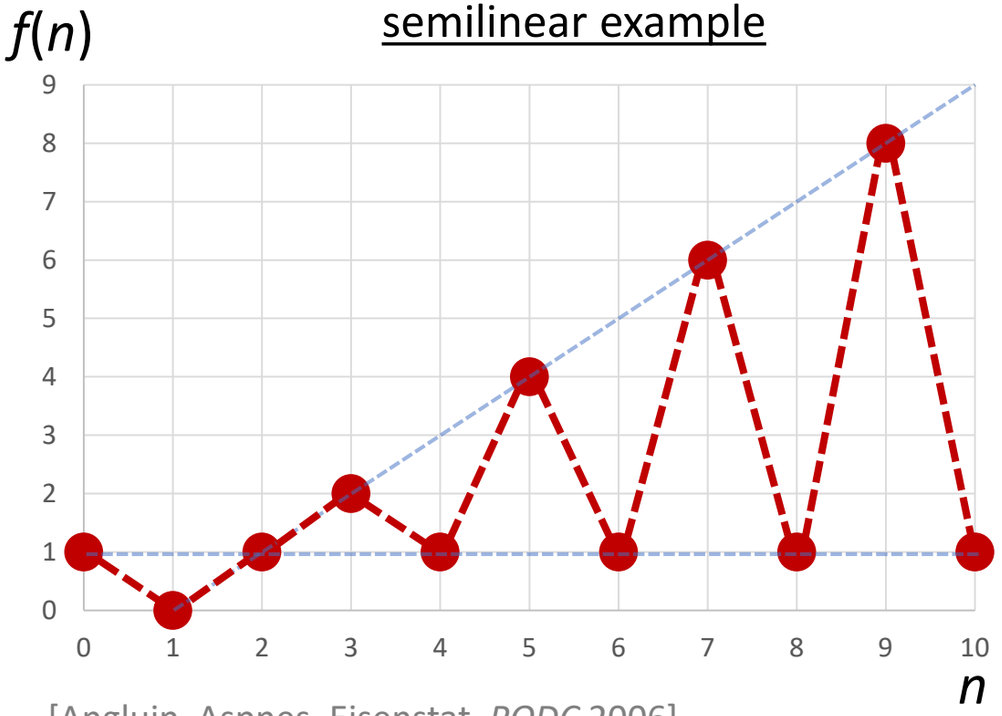


[Angluin, Aspnes, Eisenstat, *PODC* 2006]
[Chen, Doty, Soloveichik, *DNA* 2012]

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.

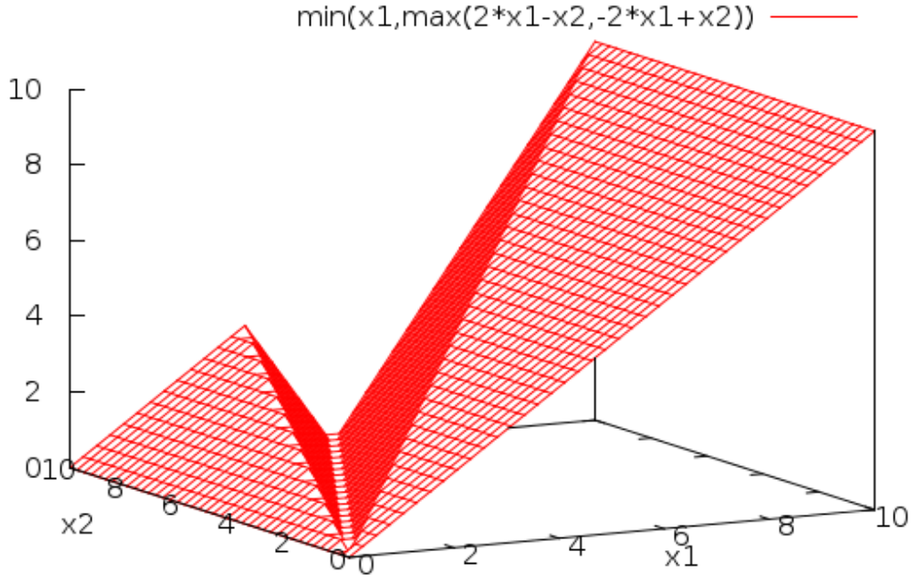
semilinear example



[Angluin, Aspnes, Eisenstat, *PODC* 2006]
[Chen, Doty, Soloveichik, *DNA* 2012]

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

continuous piecewise linear example



[Chen, Doty, Soloveichik, *ITCS* 2014]

What if we allow a small probability of error? (i.e., allow reaction rates to influence outcome)

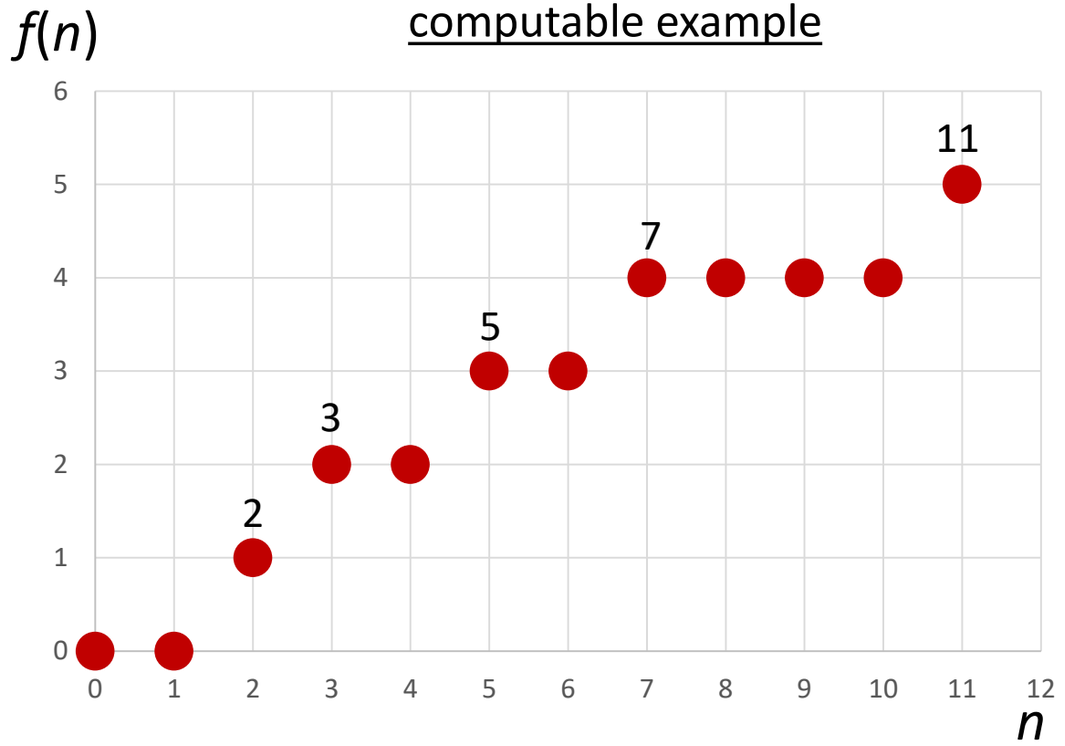
Theorem: A function is computable with probability of error < 1% by an integer-valued chemical reaction network if and only if it is computable by any algorithm whatsoever…

[Soloveichik, Cook, Bruck, Winfree, *Natural Computing* 2008]

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]



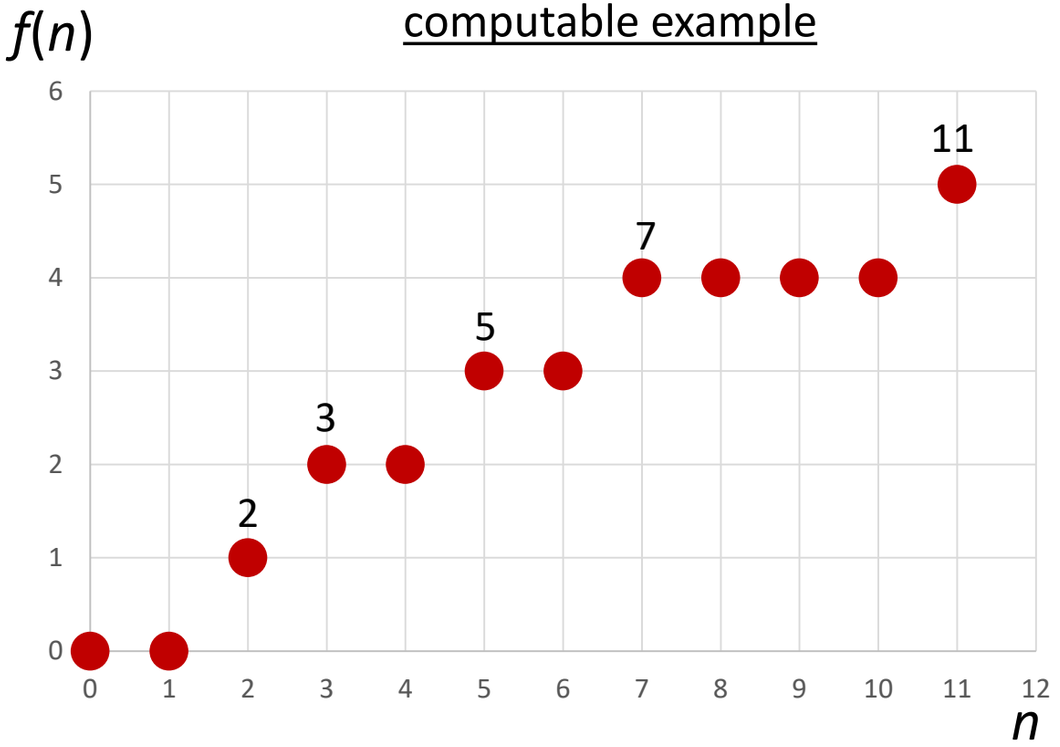
What if we allow a small probability of error? (i.e., allow reaction rates to influence outcome)

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 <u>any algorithm whatsoever</u>…

[Soloveichik, Cook, Bruck, Winfree, *Natural Computing* 2008]

… "efficiently" (polynomial-time slowdown) …

<u>computable example</u>



What if we allow a small probability of error? (i.e., allow reaction rates to influence outcome)

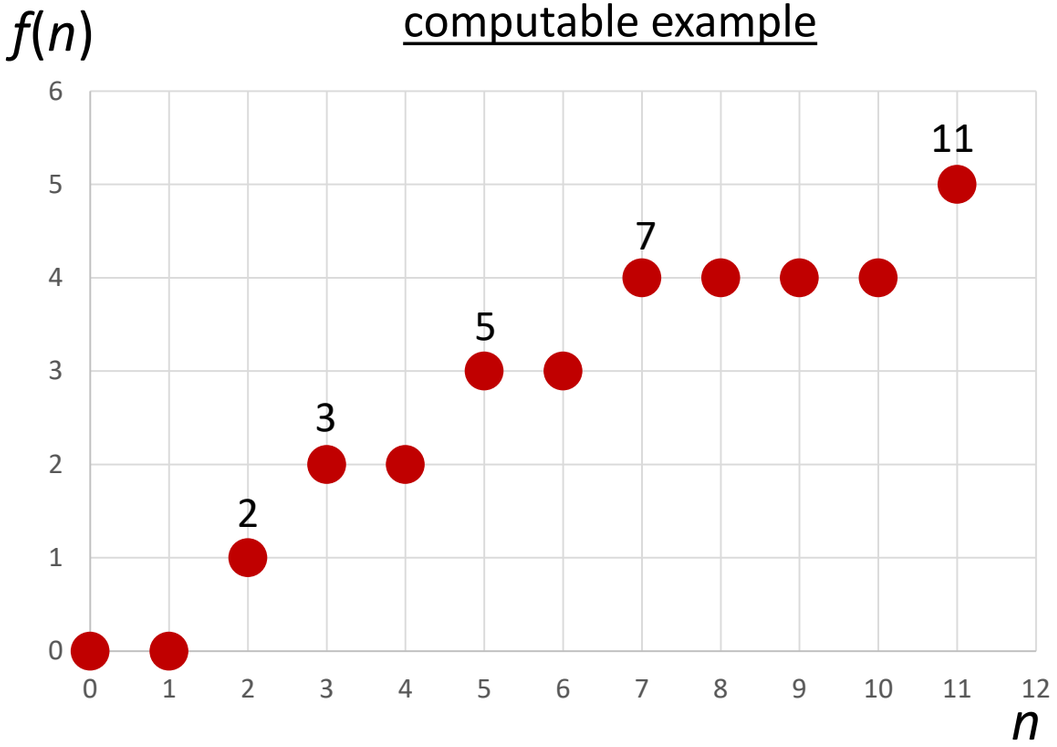
Theorem: A function is computable with probability of error < 1% by an integer-valued chemical reaction network if and only if it is computable by any algorithm whatsoever…

[Soloveichik, Cook, Bruck, Winfree, *Natural Computing* 2008]

… "efficiently" (polynomial-time slowdown) …

… **if** we have an initial leader.

computable example



What if we allow a small probability of error? (i.e., allow reaction rates to influence outcome)

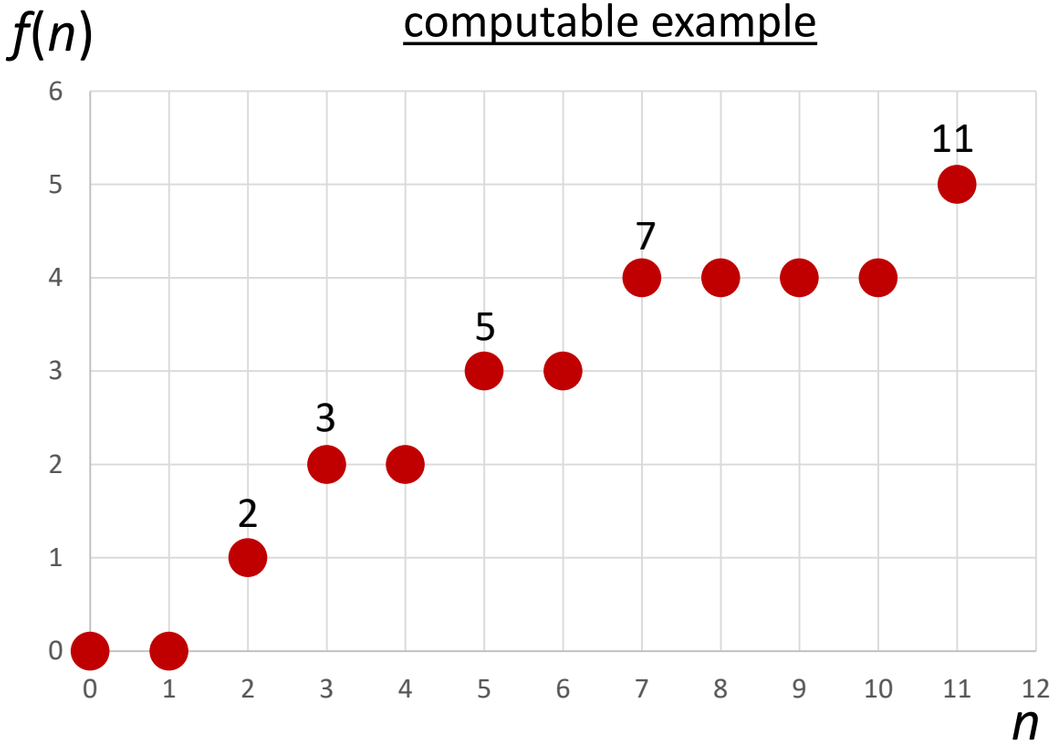
Theorem: A function is computable with probability of error < 1% by an integer-valued chemical reaction network if and only if it is computable by any algorithm whatsoever…

[Soloveichik, Cook, Bruck, Winfree, *Natural Computing* 2008]

… "efficiently" (polynomial-time slowdown) …

… **if** we have an initial leader.

computable example



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)

What if we allow a small probability of error? (i.e., allow reaction rates to influence outcome)

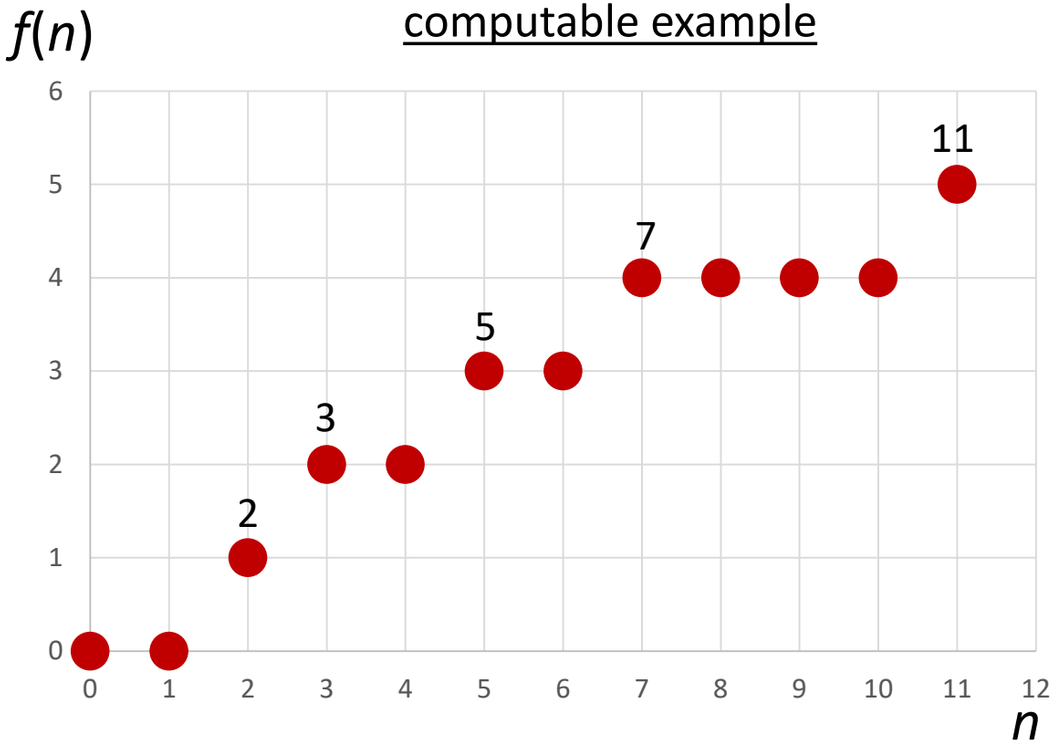
Theorem: A function is computable with probability of error < 1% by an integer-valued chemical reaction network if and only if it is computable by any algorithm whatsoever…

[Soloveichik, Cook, Bruck, Winfree, *Natural Computing* 2008]

… "efficiently" (polynomial-time slowdown) …

… **if** we have an initial leader.

computable example



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.

[Fages, Le Guludec, Bournez, Pouly. Strong Turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs. *Computational Methods in Systems Biology – CMSB* 2017]

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 <u>any algorithm whatsoever</u>.

[Fages, Le Guludec, Bournez, Pouly. Strong Turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs. *Computational Methods in Systems Biology – CMSB* 2017]

mass-action kinetics:

$$X \xrightarrow{k_1} Y+Y$$

$$Y+Z \xrightarrow{k_2} X$$

$$\dot{[X]} = -k_1[X] + k_2[Y][Z]$$

$$\dot{[Y]} = 2k_1[X] - k_2[Y][Z]$$

$$\dot{[Z]} = \quad - k_2[Y][Z]$$

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 <u>any algorithm whatsoever</u>.

[Fages, Le Guludec, Bournez, Pouly. Strong Turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs. *Computational Methods in Systems Biology – CMSB* 2017]

mass-action kinetics:

$$X \xrightarrow{k_1} Y+Y$$

$$Y+Z \xrightarrow{k_2} X$$

$$\dot{[X]} = -k_1[X] + k_2[Y][Z]$$

$$\dot{[Y]} = 2k_1[X] - k_2[Y][Z]$$

$$\dot{[Z]} = \quad - k_2[Y][Z]$$

… with only a polynomial-time slowdown.

[Bournez, Graça, Pouly. Polynomial time corresponds to solutions of polynomial ordinary differential equations of polynomial length. *Journal of the ACM* 2017]

Fast approximate division by 2

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

$$X + A \to B + Y$$
$$X + B \to A$$

guaranteed to get
$Y = n/2 \pm \varepsilon n$
E[time] = O(log n) / ε

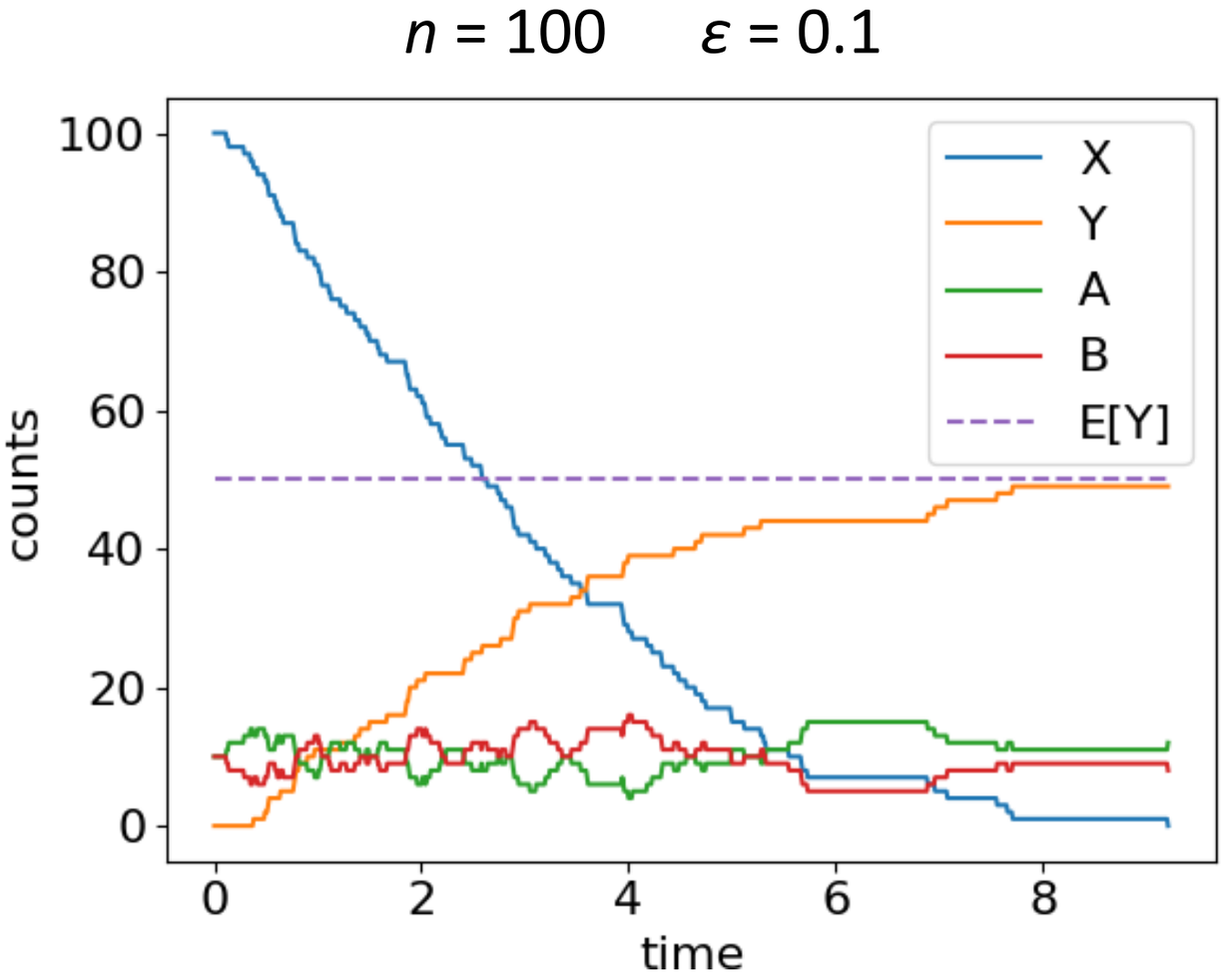
[Belleville, Doty, Soloveichik, Hardness of computing and approximating predicates and functions with leaderless population protocols, *ICALP* 2017]

Fast approximate division by 2

initial configuration:
{ n X, εn A, εn B }

$$X + A \rightarrow B + Y$$
$$X + B \rightarrow A$$

guaranteed to get
$Y = n/2 \pm \varepsilon n$
E[time] = O(log n) / ε



[Belleville, Doty, Soloveichik, Hardness of computing and approximating predicates and functions with leaderless population protocols, *ICALP* 2017]

Thank you!

Questions?

Proofs of conjectures?

Criticism of model?

suggestions: 1) lack of reverse reactions (energy dissipated = ∞), 2) assumption of single initial copy of leader, 3) reactions aren't mass-conserving, 4) what about leak reactions?, 5) no discussion of diffusion rates, 6) this has nothing to do with how biological cells compute