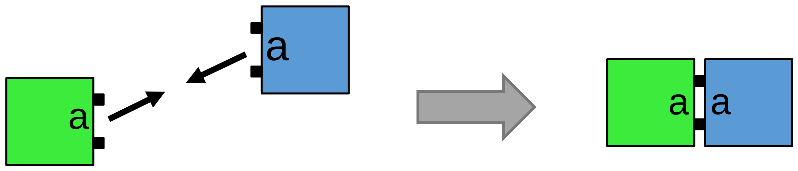
Computation with chemistry

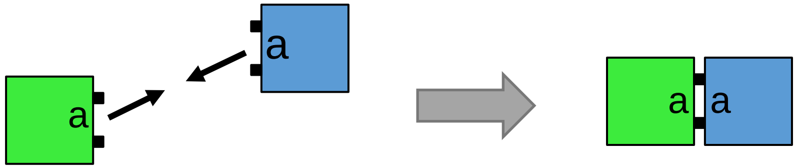
ECS 232: Theory of Molecular Computation, UC Davis

Chemical reaction networks

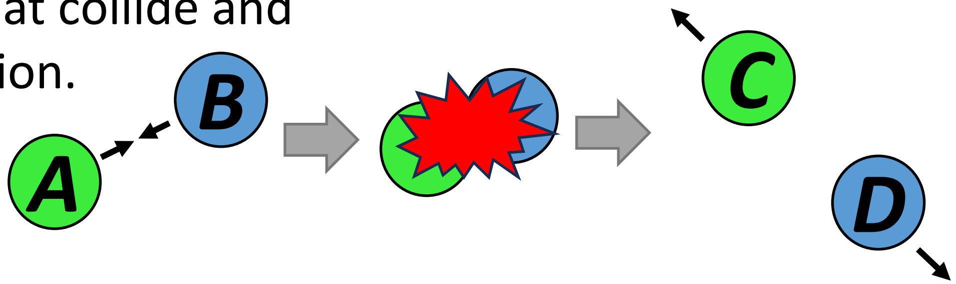


- aTAM self-assembly describes stateless molecules that collide and stick together.

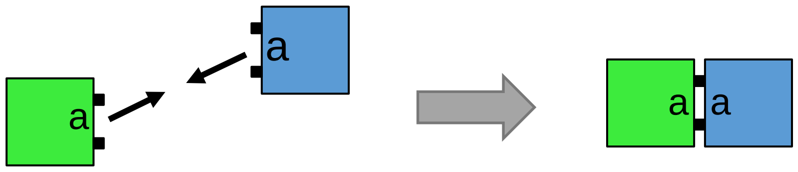
Chemical reaction networks



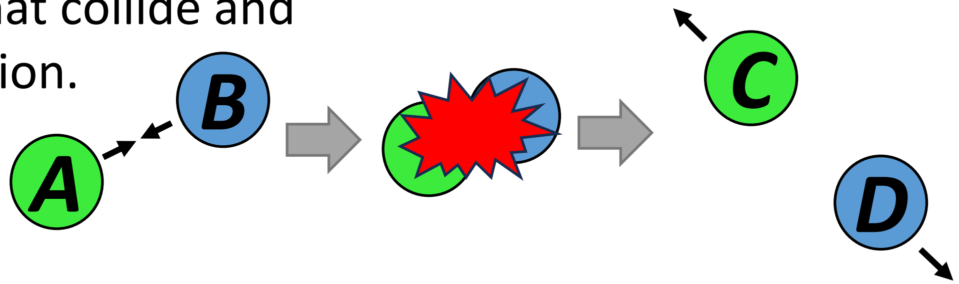
- aTAM self-assembly describes stateless molecules that collide and stick together.
- Chemical reaction network model describes stateful molecules that collide and bounce apart, but that might change state as a result of the collision.



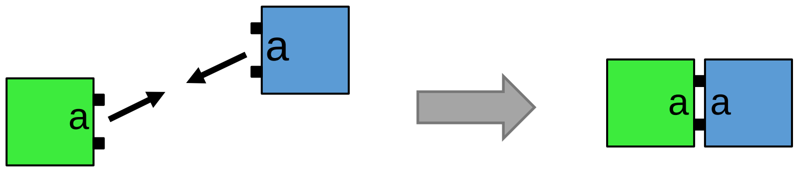
Chemical reaction networks



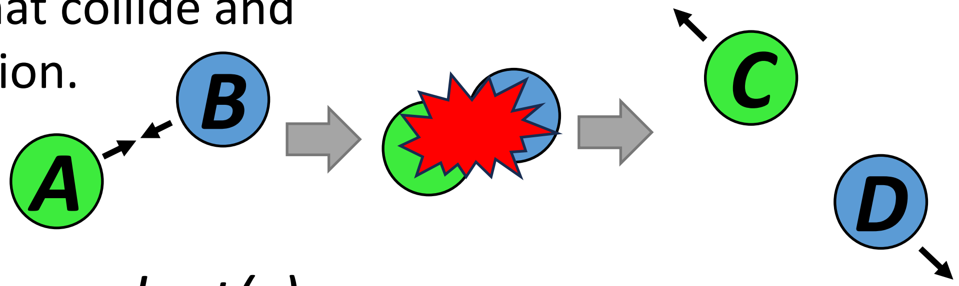
- aTAM self-assembly describes stateless molecules that collide and stick together.
- Chemical reaction network model describes stateful molecules that collide and bounce apart, but that might change state as a result of the collision.
- Allow more general reactions that produce/consume molecules.



Chemical reaction networks

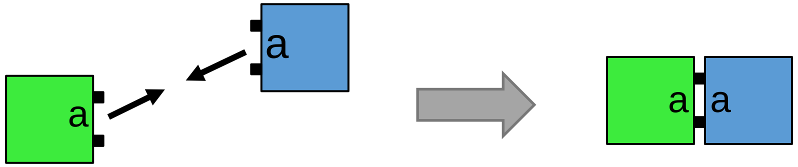


- aTAM self-assembly describes stateless molecules that collide and stick together.
- Chemical reaction network model describes stateful molecules that collide and bounce apart, but that might change state as a result of the collision.
- Allow more general reactions that produce/consume molecules.

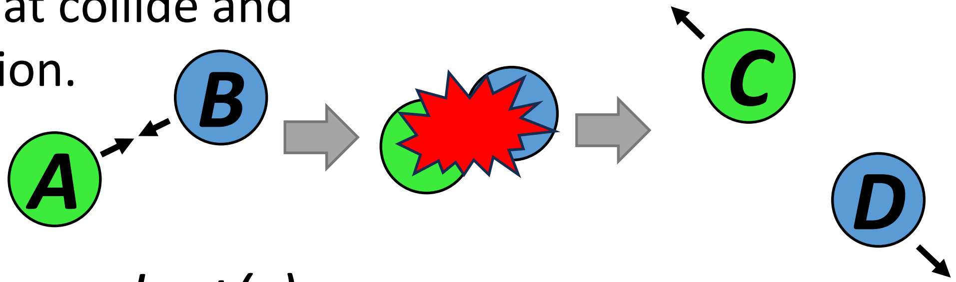


reactant(s) $$R \rightarrow P_1 + P_2$$ *product(s)*

Chemical reaction networks



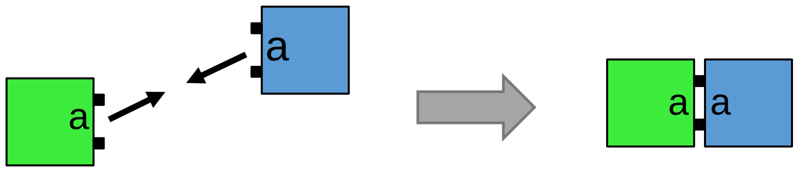
- aTAM self-assembly describes stateless molecules that collide and stick together.
- Chemical reaction network model describes stateful molecules that collide and bounce apart, but that might change state as a result of the collision.
- Allow more general reactions that produce/consume molecules.



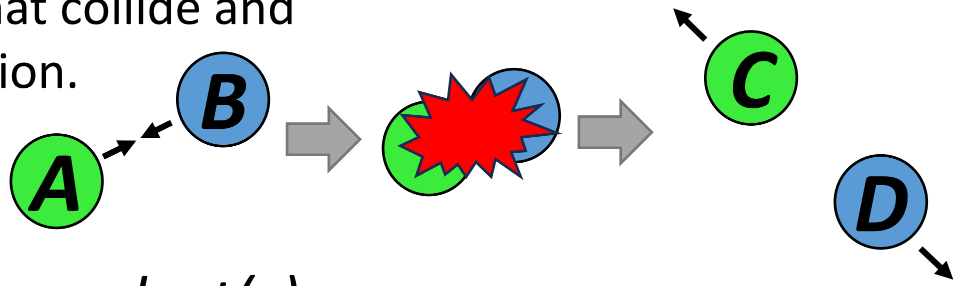
reactant(s) $R \rightarrow P_1 + P_2$ *product(s)*

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

Chemical reaction networks



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

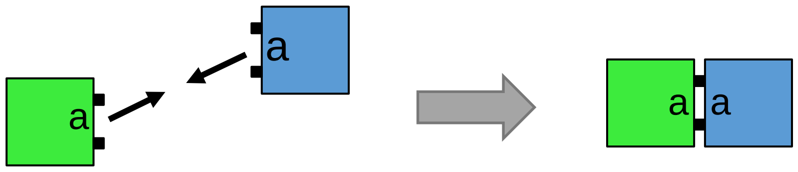


reactant(s) $R \rightarrow P_1 + P_2$ *product(s)*

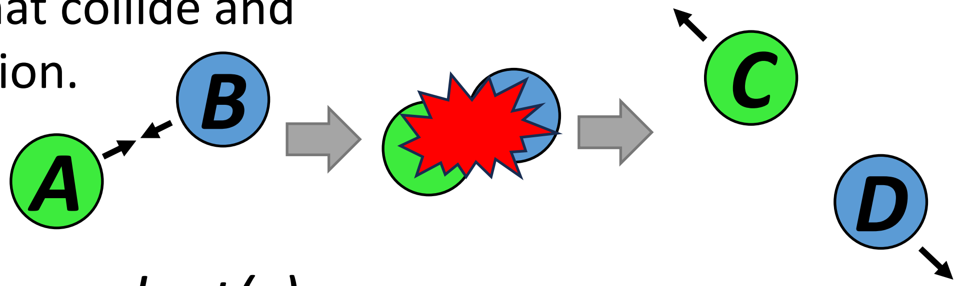
monomers $M_1 + M_2 \rightarrow D$ *dimer*

catalyst $C + X \rightarrow C + Y$

Chemical reaction networks



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



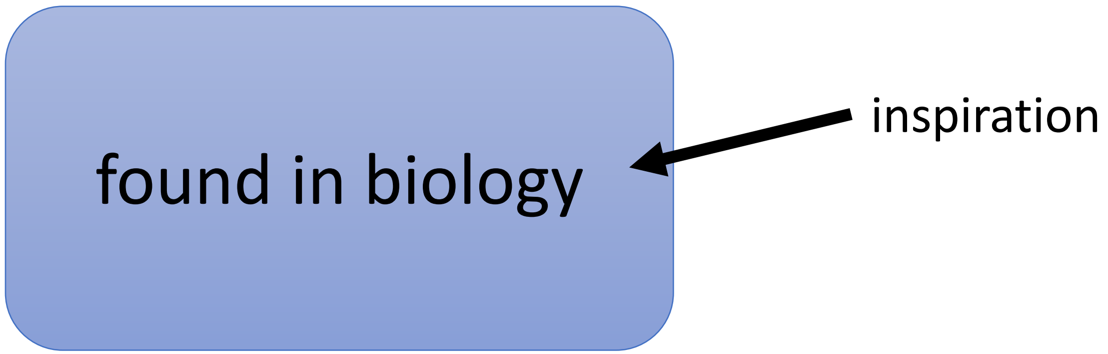
reactant(s) $R \to P_1+P_2$ *product(s)*

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

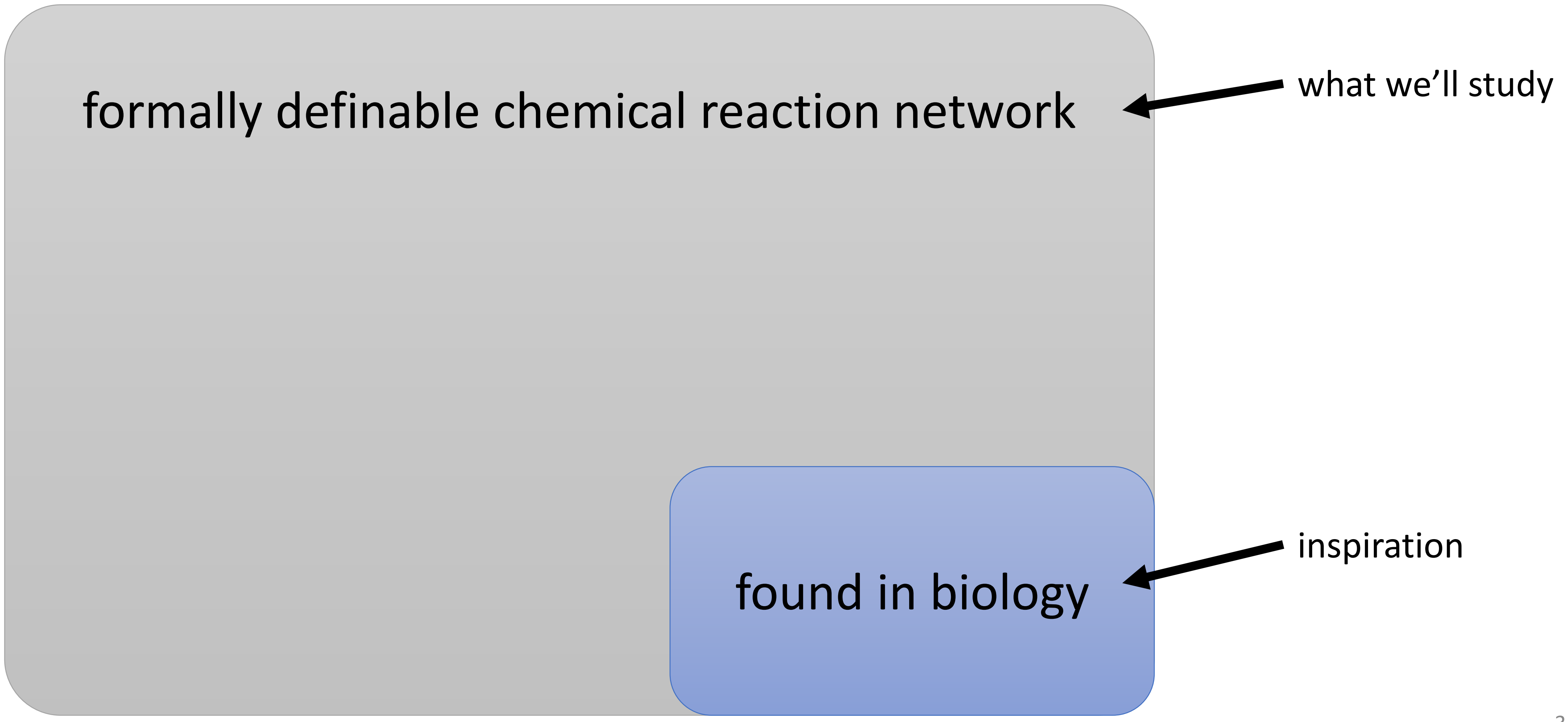
catalyst $C+X \to C+Y$

Traditionally a descriptive modeling language…
Let's instead use it as a prescriptive programming language

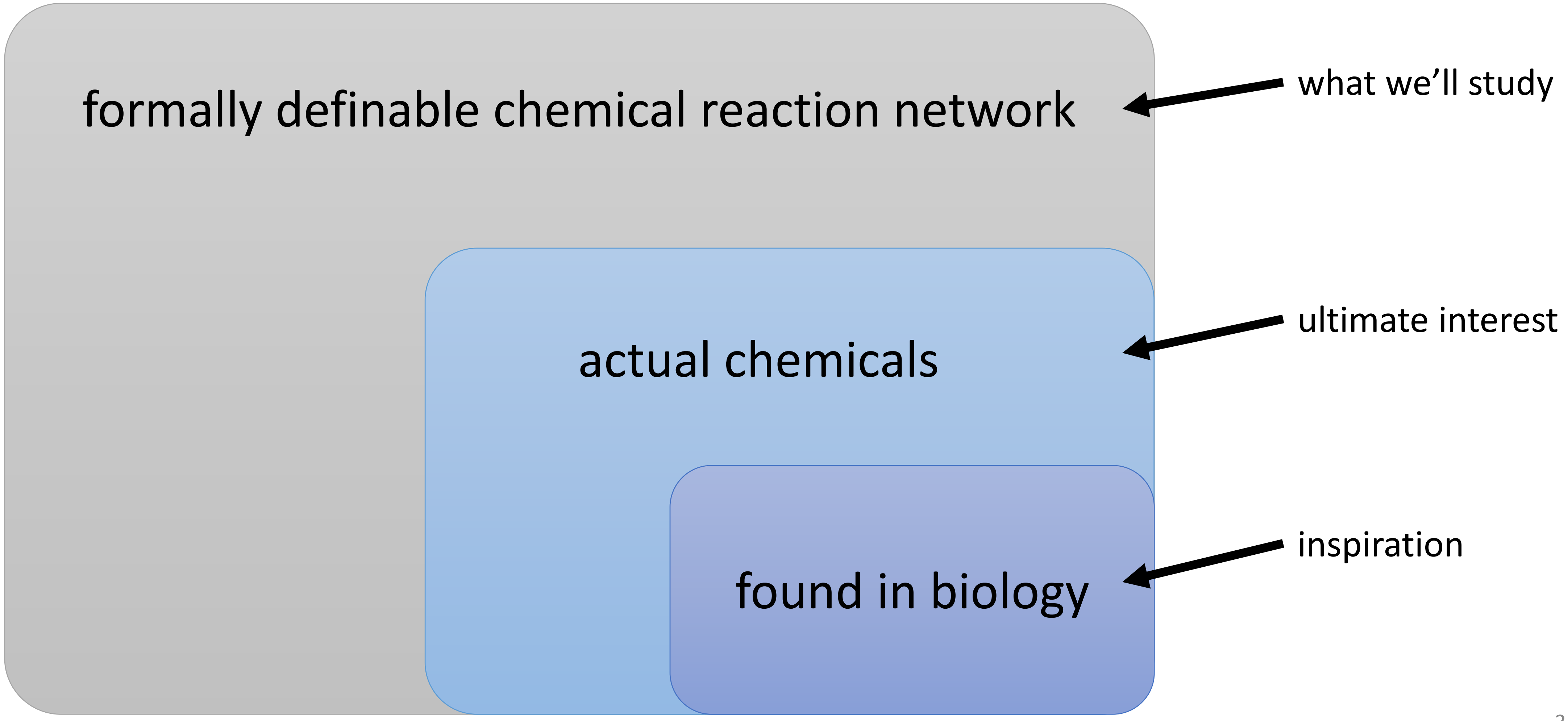
What behavior is possible for chemistry in principle?



What behavior is possible for chemistry in principle?



What behavior is possible for chemistry in principle?

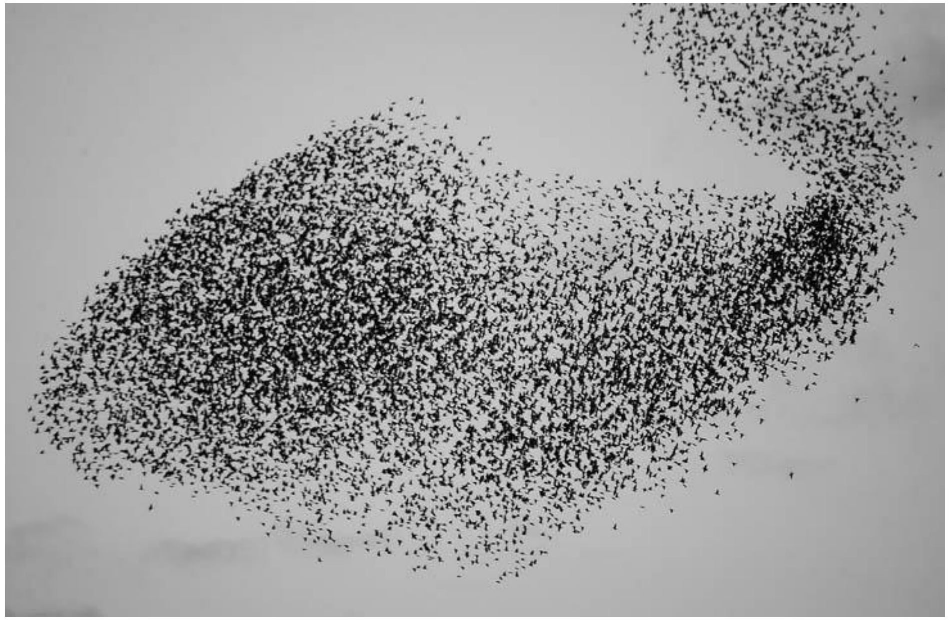


Computation with chemical reaction networks

- Key ideas setting chemical computation apart from others:
  - <u>cannot</u> control order in which molecules collide
  - <u>can</u> control how they react when they collide

Computation with chemical reaction networks

- Key ideas setting chemical computation apart from others:
  - cannot control order in which molecules collide
  - can control how they react when they collide
- Related model of distributed computing called *population protocols*
  - originally motivated by mobile wireless sensor networks, e.g., attached to a birds in a flock



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

Example: Chemical caucusing

opposite
opinions cancel

$$X + Y \rightarrow U + U$$

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

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

Example: Chemical caucusing

opposite
opinions cancel

$$X + Y \rightarrow U + U$$

both opinions
influence the
unopinionated

$$X + U \rightarrow X + X$$

$$Y + U \rightarrow Y + Y$$

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

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

Example: Chemical caucusing

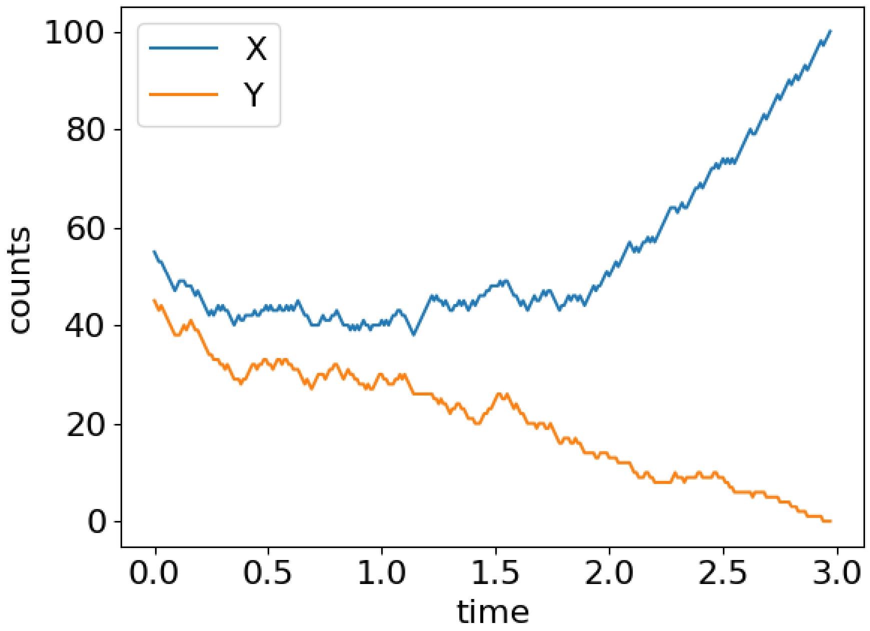
opposite opinions cancel

$$X+Y \rightarrow U+U$$

both opinions influence the unopinionated

$$X+U \rightarrow X+X$$

$$Y+U \rightarrow Y+Y$$



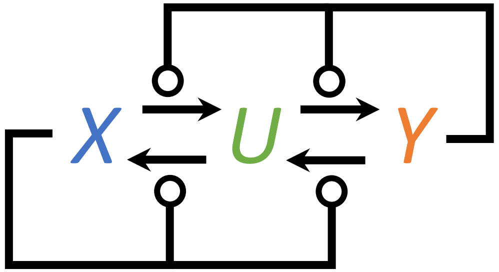
distributed algorithm for *"approximate majority"*:
initial majority (X or Y) quickly overtakes whole population (with high probability)

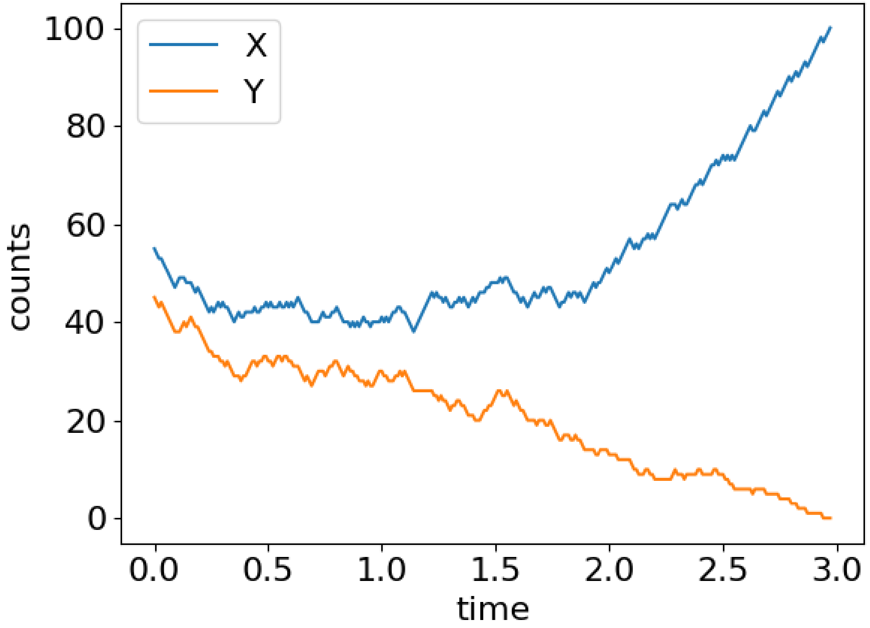
[Angluin, Aspnes, Eisenstat, *A simple population protocol for fast robust approximate majority*, DISC 2007]

Example: Chemical caucusing

opposite opinions cancel $X + Y \rightarrow U + U$

both opinions influence the unopinionated
$X + U \rightarrow X + X$
$Y + U \rightarrow Y + Y$

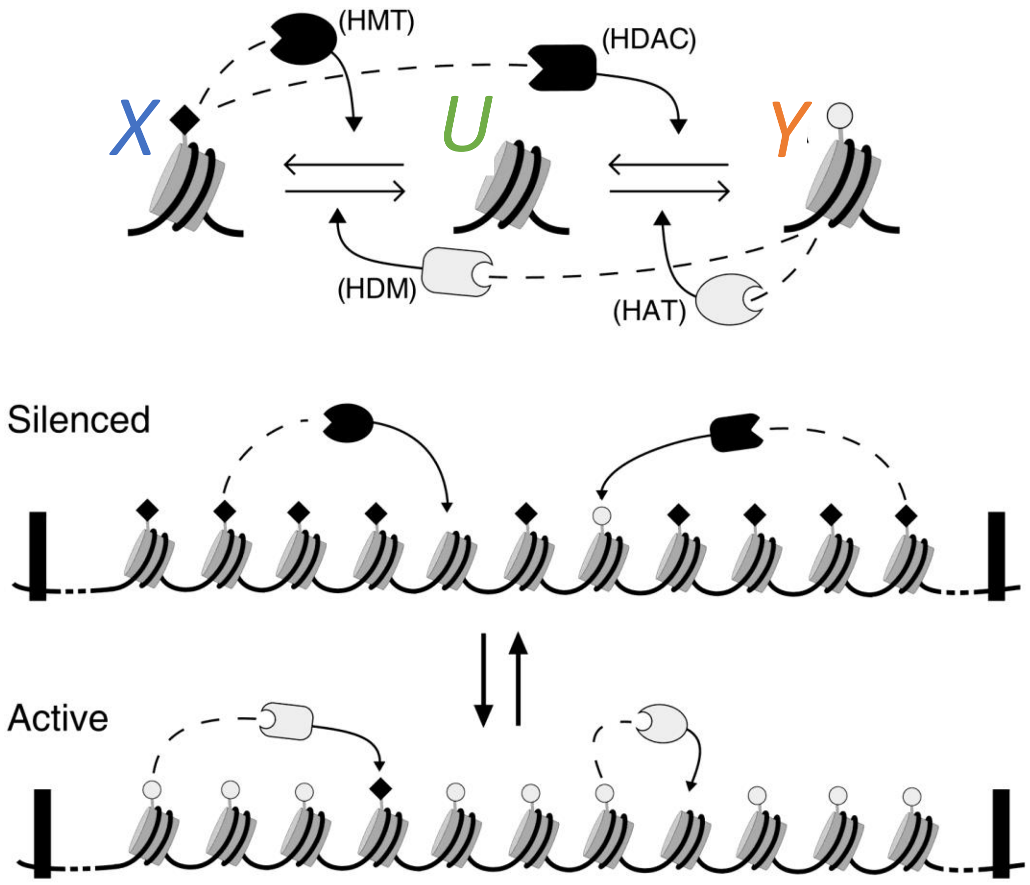




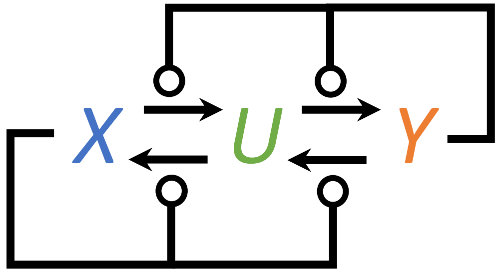
distributed algorithm for *"approximate majority"*:
initial majority (X or Y) quickly overtakes whole population (with high probability)

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

Does chemistry compute?

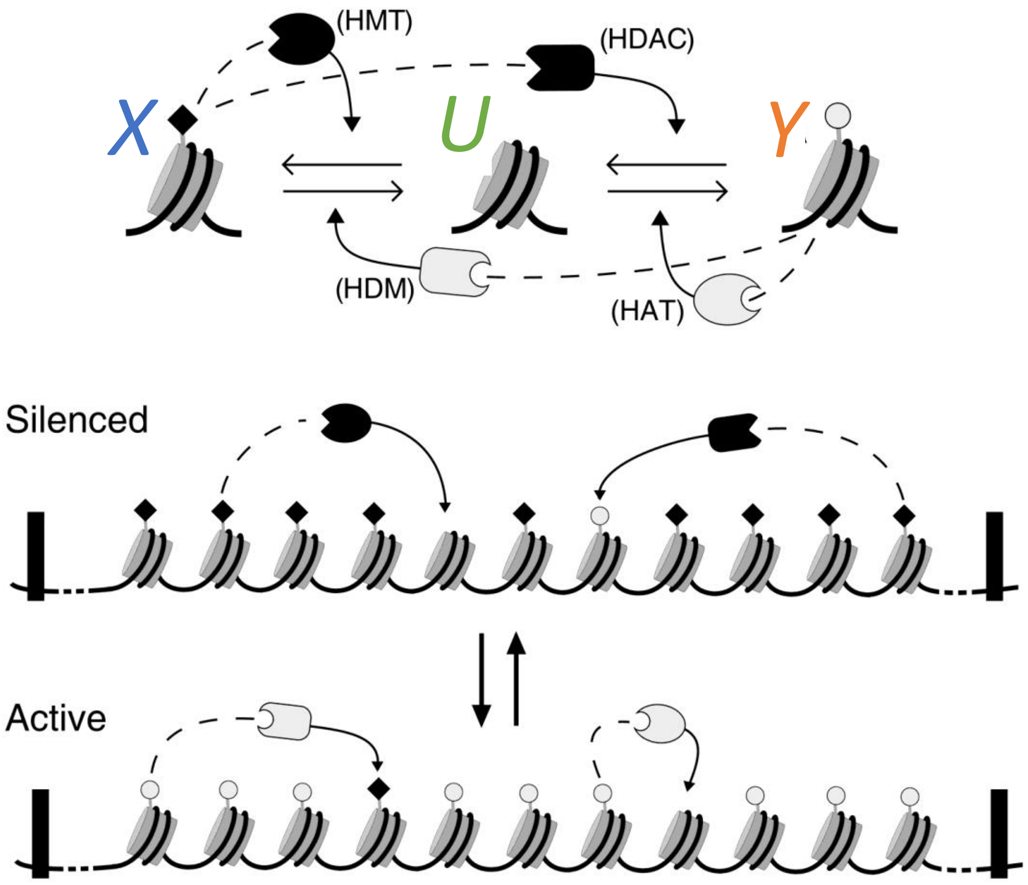


=

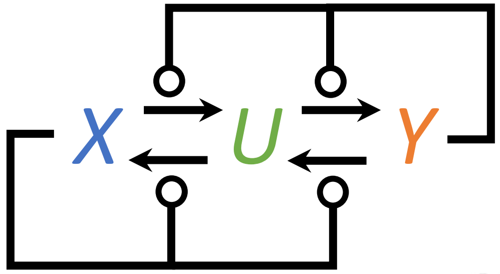


[Dodd, Micheelsen, Sneppen, Thon. Theoretical analysis of epigenetic cell memory by nucleosome modification, *Cell* 2007]

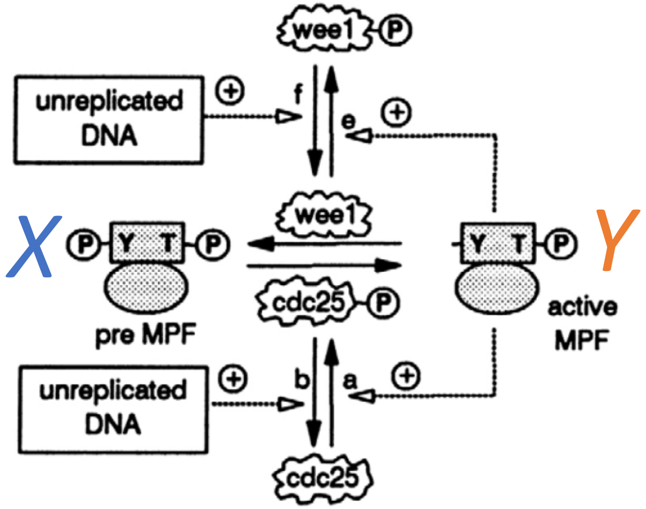
Does chemistry compute?



=



≈

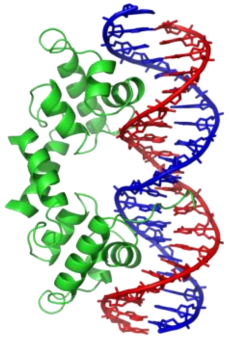


[Dodd, Micheelsen, Sneppen, Thon. Theoretical analysis of epigenetic cell memory by nucleosome modification, *Cell* 2007]

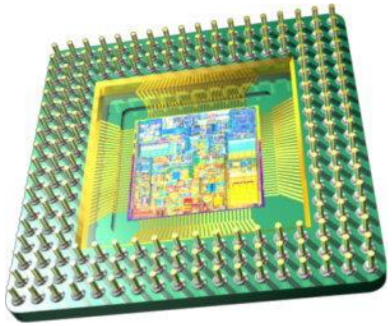
[Cardelli, Csikász-Nagy. The cell cycle switch computes approximate majority. *Nature Scientific Reports* 2012]

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

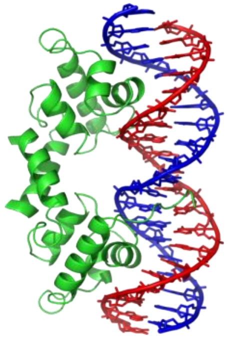
Why compute with chemistry?



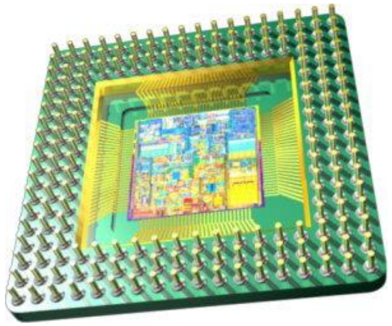
versus



Why compute with chemistry?

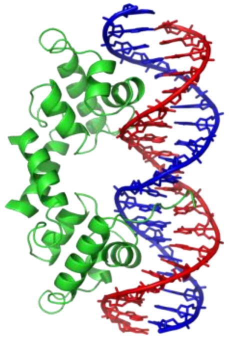


versus

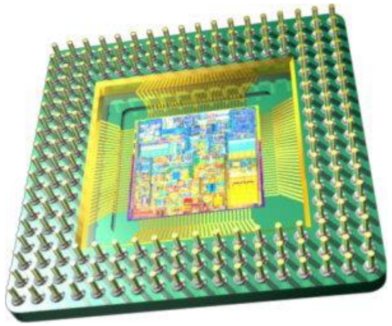


speed?

Why compute with chemistry?



versus

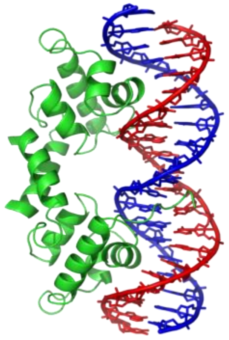


slow

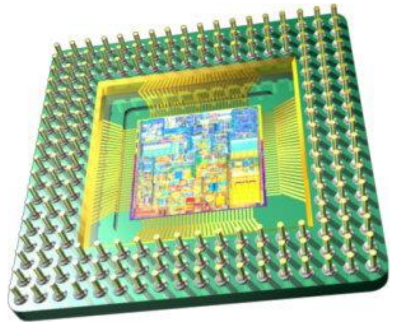
~~speed?~~

fast

Why compute with chemistry?



versus



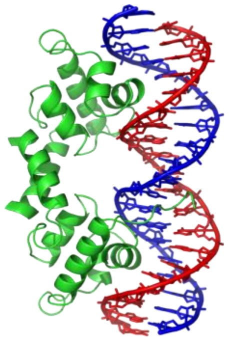
slow

~~speed?~~

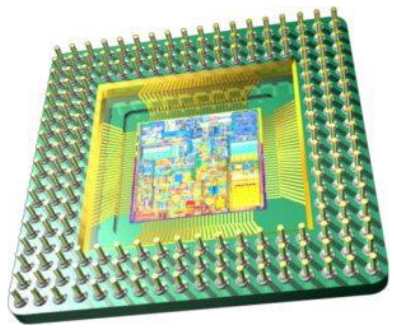
fast

component size?

Why compute with chemistry?

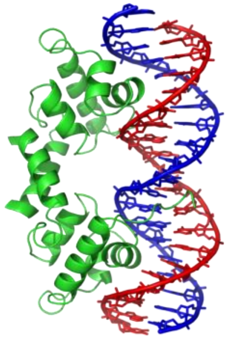
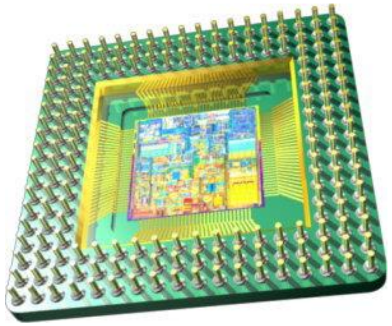


versus

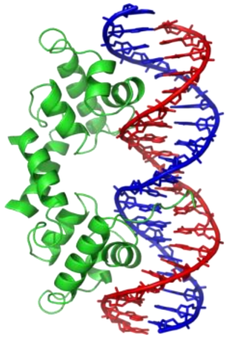


slow	~~speed?~~	fast
≈ 10-100 nm	~~component size?~~	≈ 10-100 nm

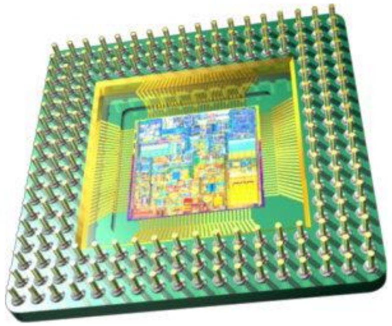
Why compute with chemistry?

	versus	
slow	speed?	fast
≈ 10-100 nm	component size?	≈ 10-100 nm
yes	compatible with “wet environments”?	not easily

Why compute with chemistry?

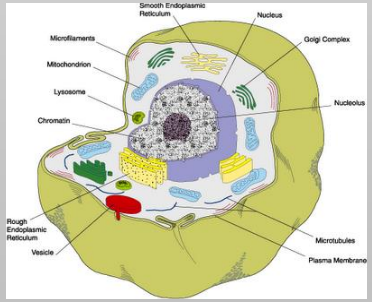


versus



slow	~~speed?~~	fast
≈ 10-100 nm	~~component size?~~	≈ 10-100 nm
yes	compatible with “wet environments”?	not easily

cells



smart drug released only in certain cellular conditions

DNA storage

in-place computation replacing expensive read/write lab steps

bioreactors

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

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

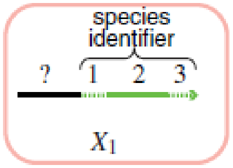
$$X_1 + X_2 \rightarrow X_3$$

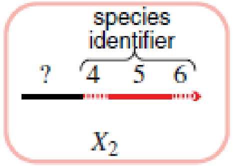
Can we compute with chemistry?

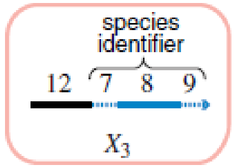
"Not every chemical reaction network describes real chemicals!", i.e. "where's the *compiler*?"

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

$$X_1 + X_2 \rightarrow X_3$$





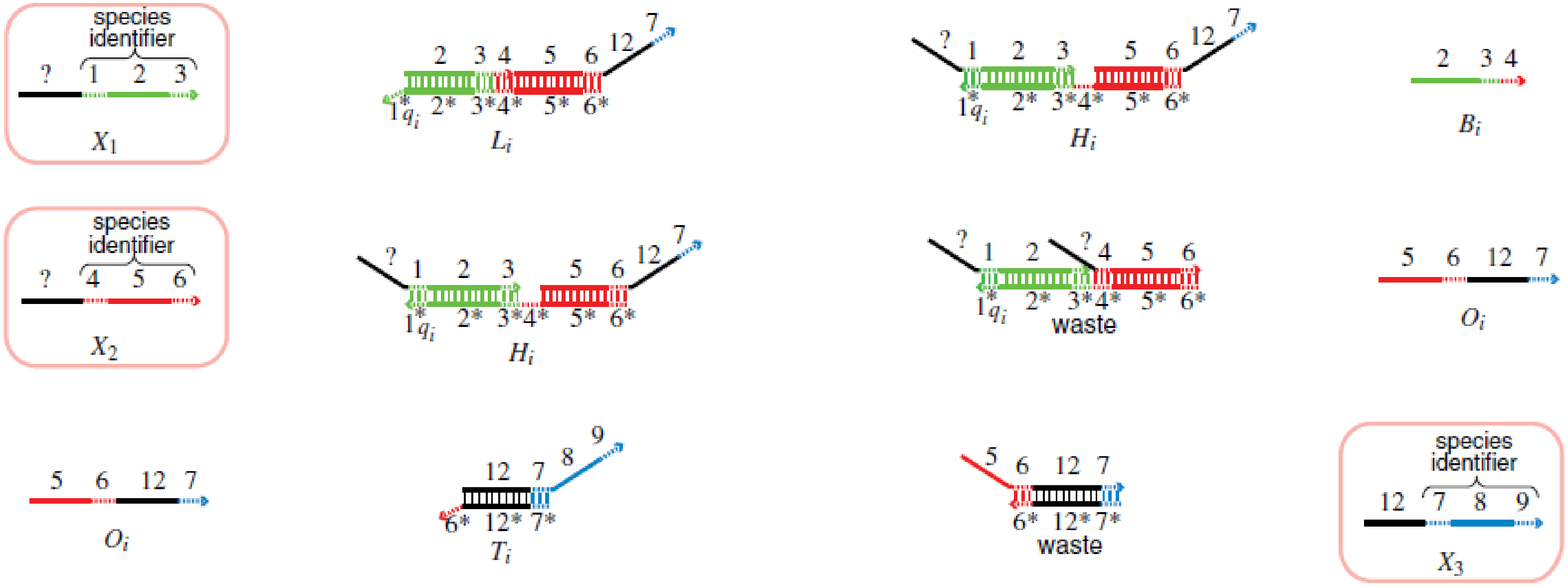


Can we compute with chemistry?

"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*

$$X_1 + X_2 \rightarrow X_3$$

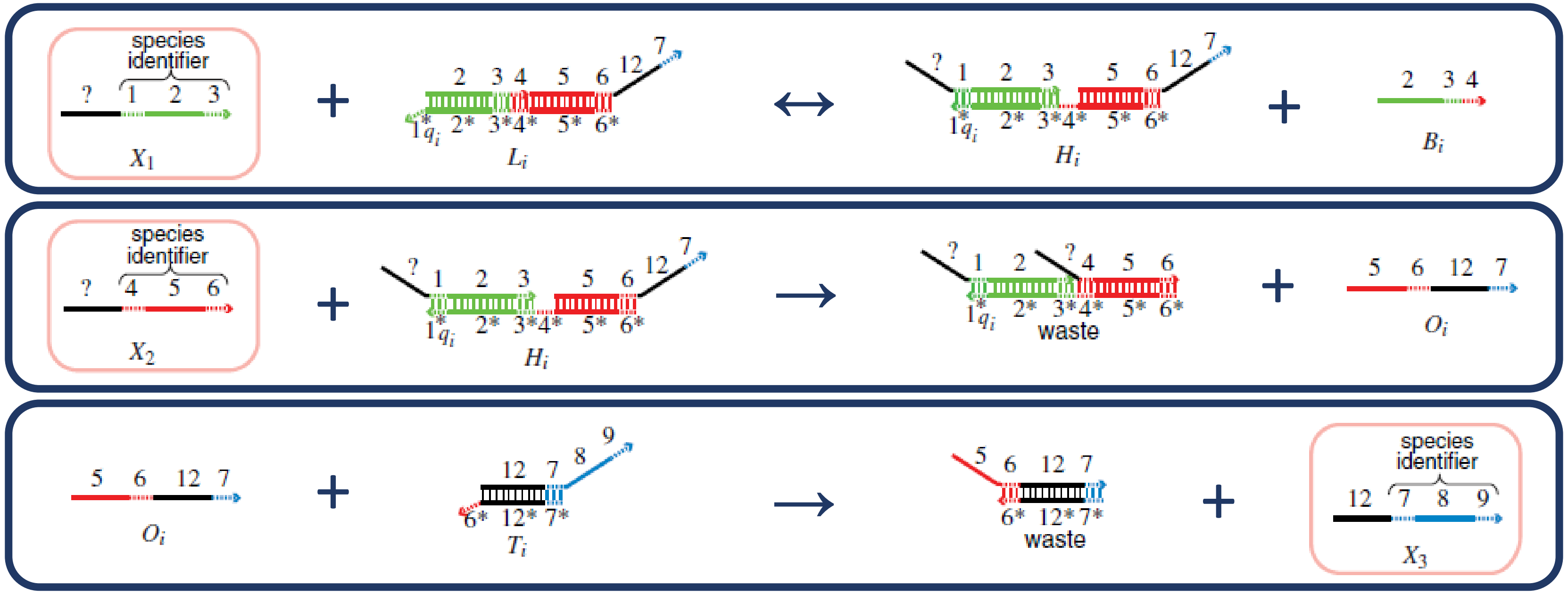


Can we compute with chemistry?

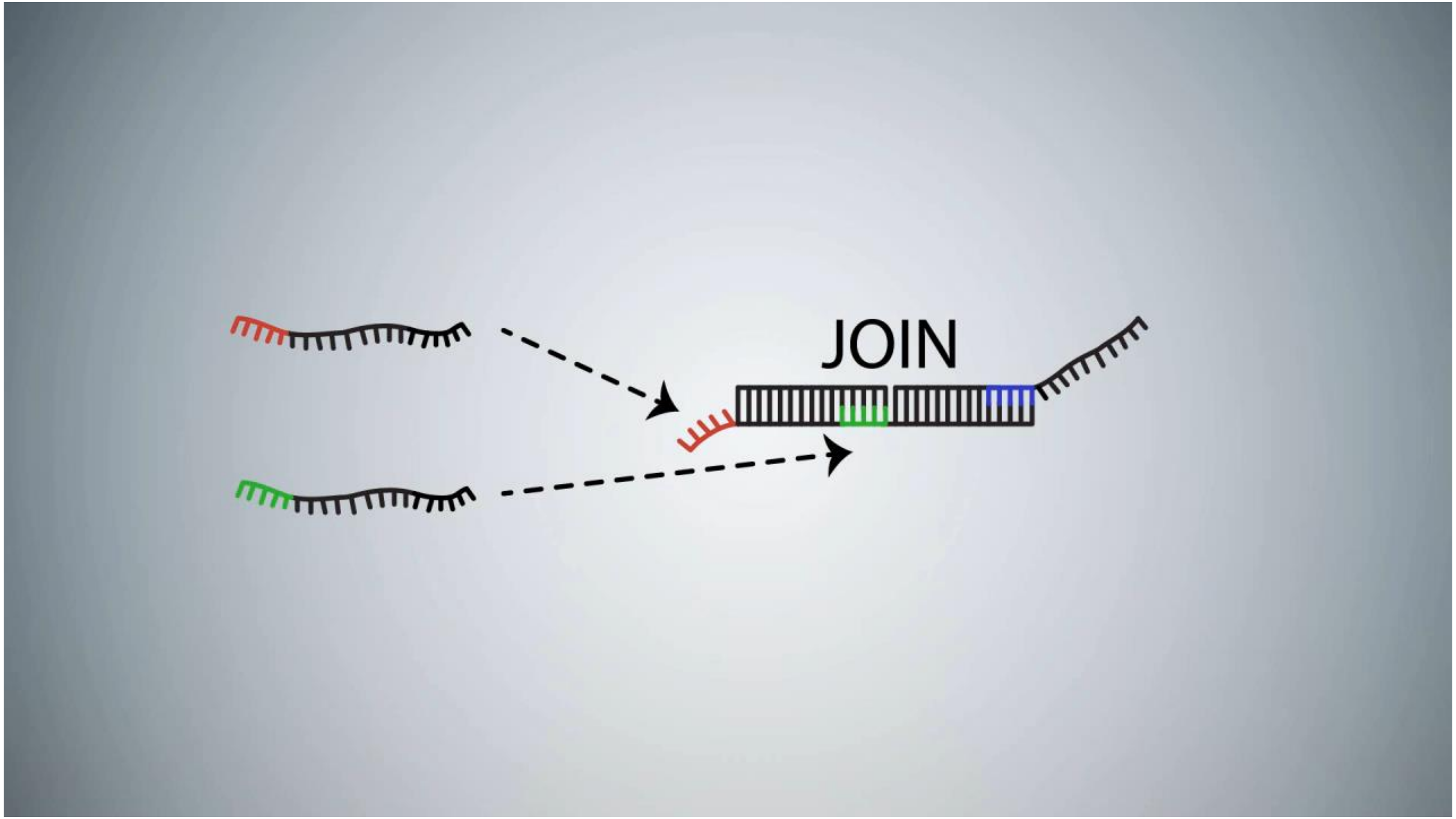
"Not every chemical reaction network describes real chemicals!", i.e. "where's the *compiler*?"

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

$$X_1 + X_2 \rightarrow X_3$$



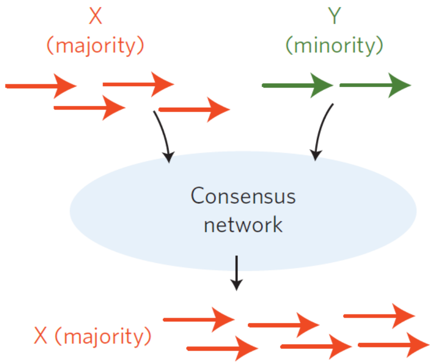
DNA strand displacement implementing A+B → C



video: Microsoft Research Cambridge

Experimental implementations of synthetic chemical reaction networks with DNA

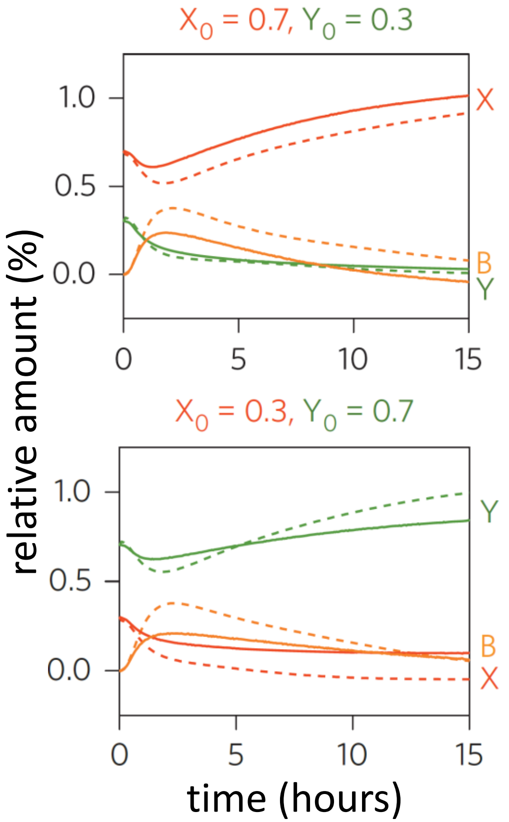
Analog majority computation



$X+Y \rightarrow B+B$

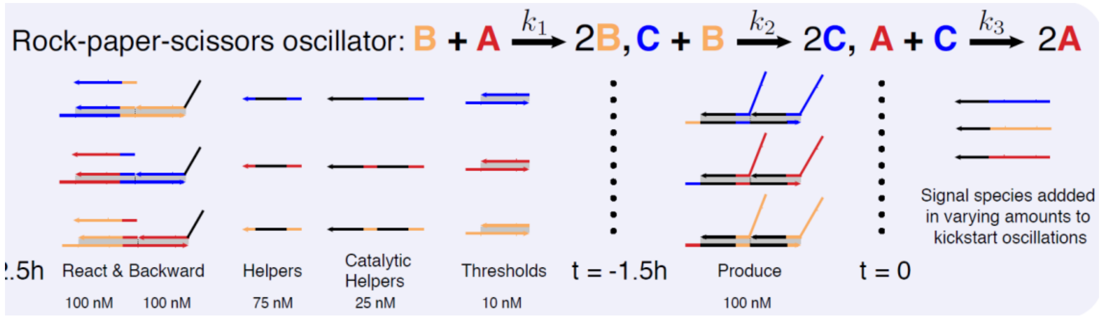
$X+B \rightarrow X+X$

$Y+B \rightarrow Y+Y$



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

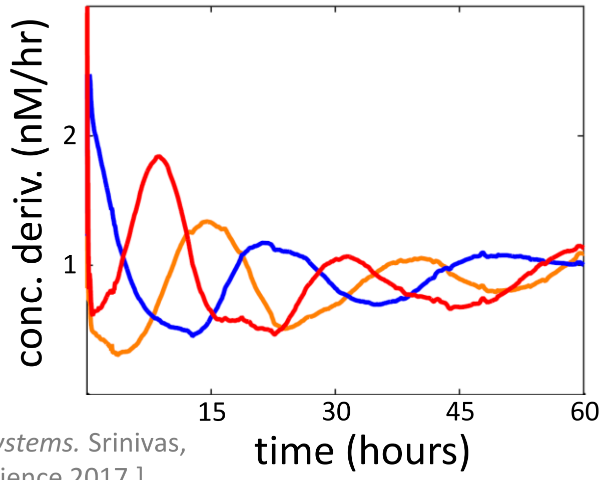
Chemical oscillator



$A+B \rightarrow B+B$

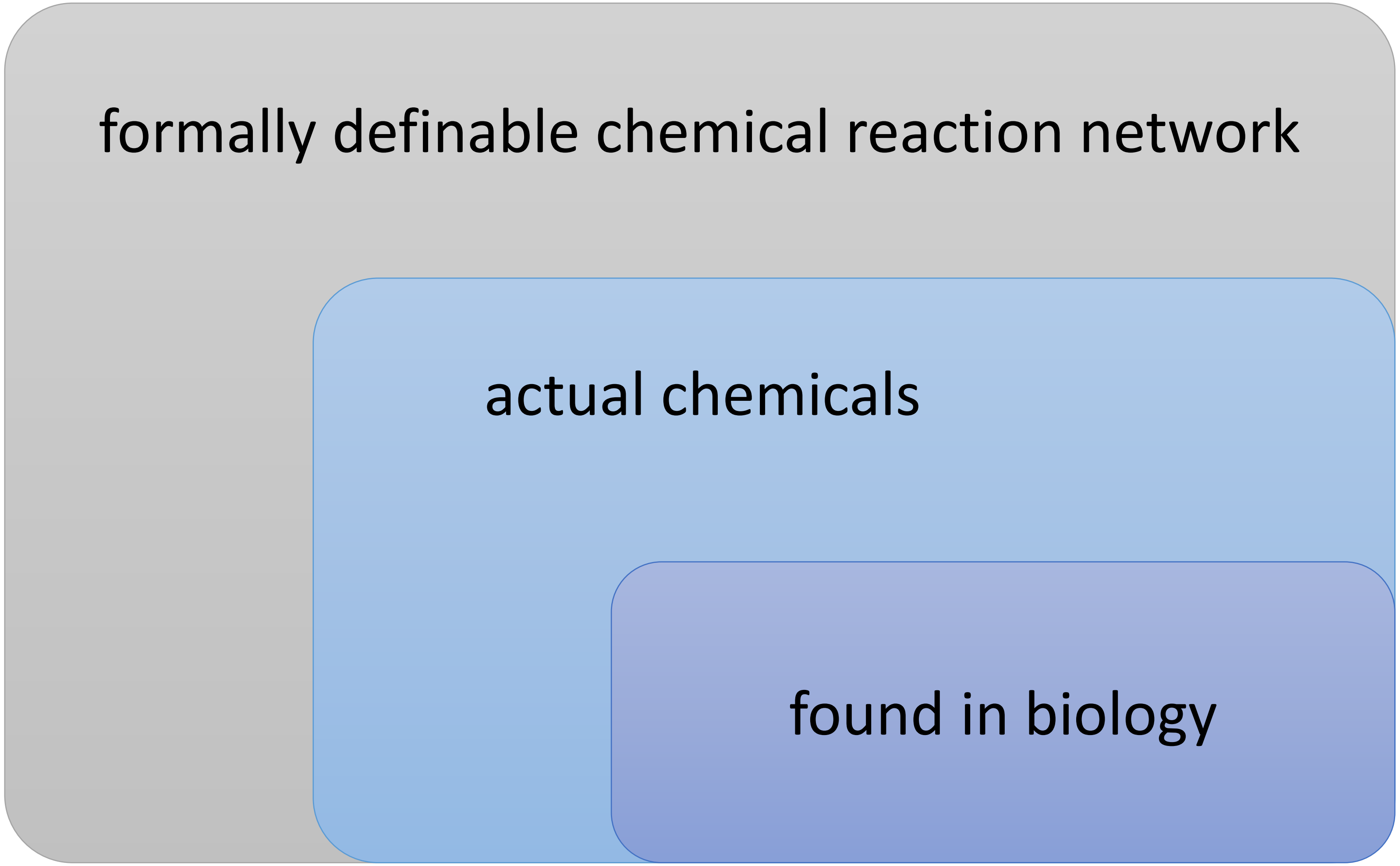
$B+C \rightarrow C+C$

$C+A \rightarrow A+A$

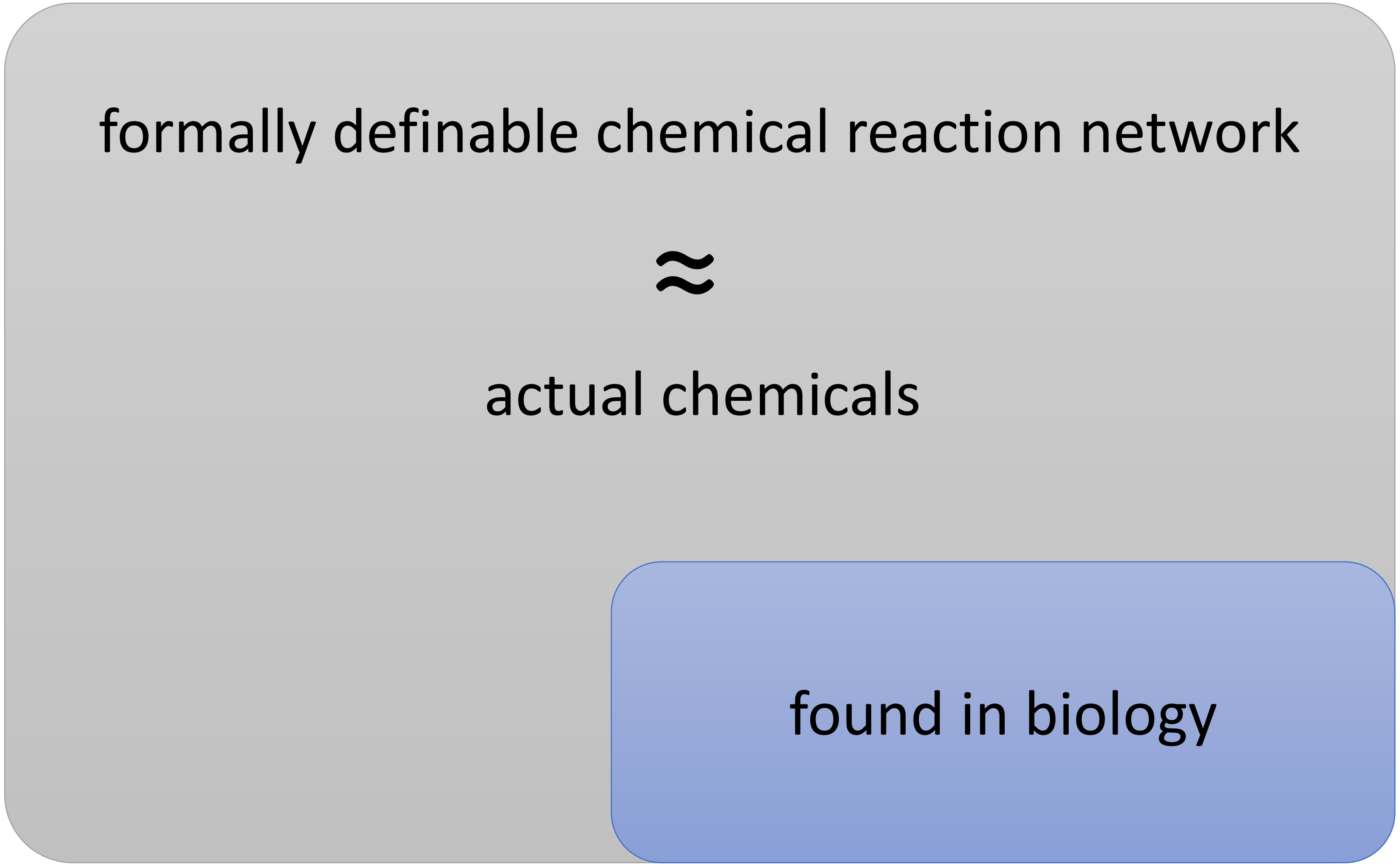


[*Enzyme-free nucleic acid dynamical systems.* Srinivas, Parkin, Seelig, Winfree, Soloveichik, Science 2017.]

What behavior is possible for chemistry in principle?

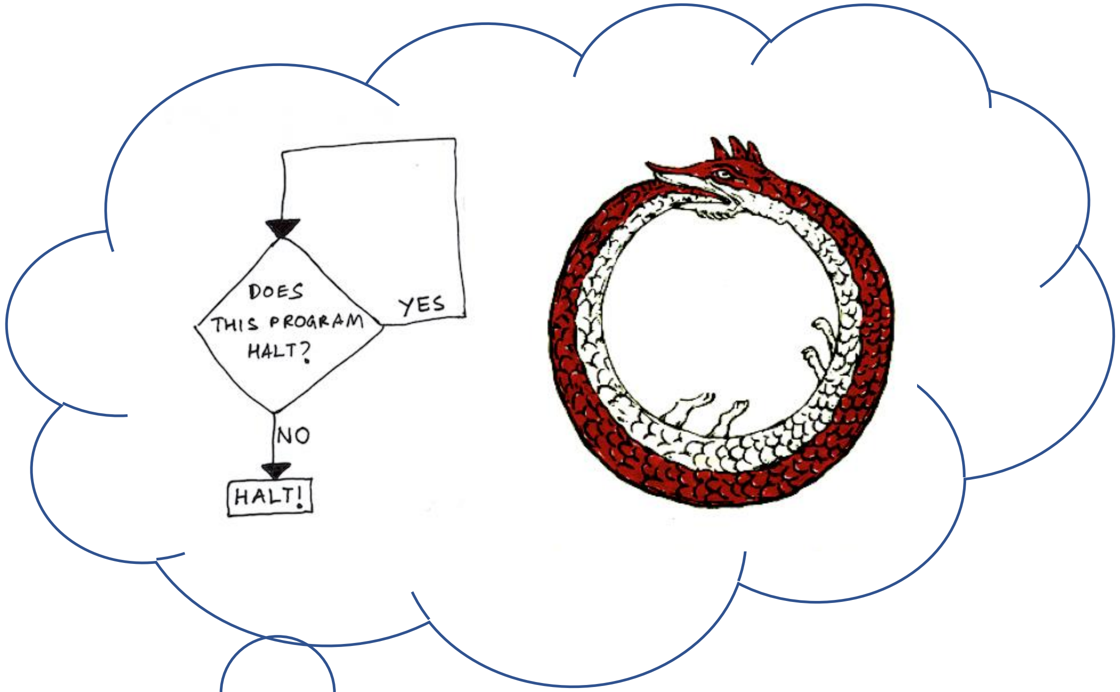


What behavior is possible for chemistry in principle?



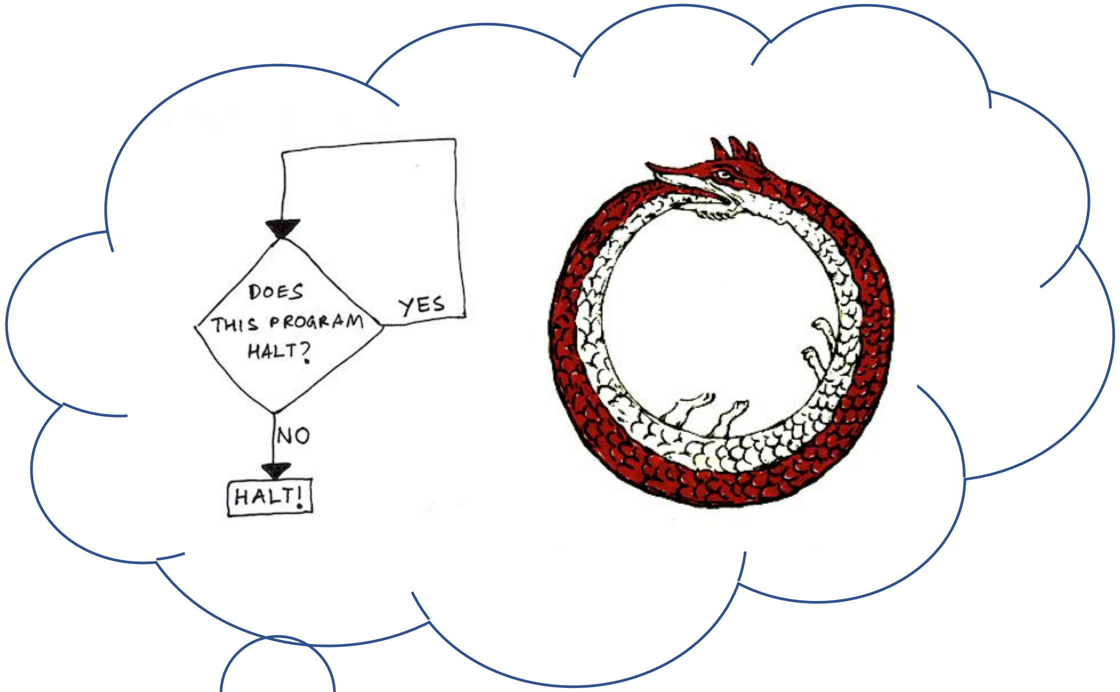
Theoretical Computer Science Approach

LIMITED Edition est. 1936



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*)

Chemical Reaction Networks (formal definition)

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

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

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

$$C + 2B \xrightarrow{k_3} C$$

Chemical Reaction Networks (formal definition)

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

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

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

$$C+2B \xrightarrow{k_3} C$$

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

Chemical Reaction Networks (formal definition)

- finite set of d <u>species</u> $\Lambda = \{ A, B, C, D, \ldots \}$

- finite set of <u>reactions</u>: *e.g.*

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

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

$$C+2B \xrightarrow{k_3} C$$

k_1, k_2, k_3 are called <u>rate constants</u>;
if not specified, assume = 1.

- <u>configuration</u> $\mathbf{x} \in \mathbb{N}^d$: molecular counts of each species

Chemical Reaction Networks (formal definition)

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

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

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

$$C+2B \xrightarrow{k_3} C$$

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

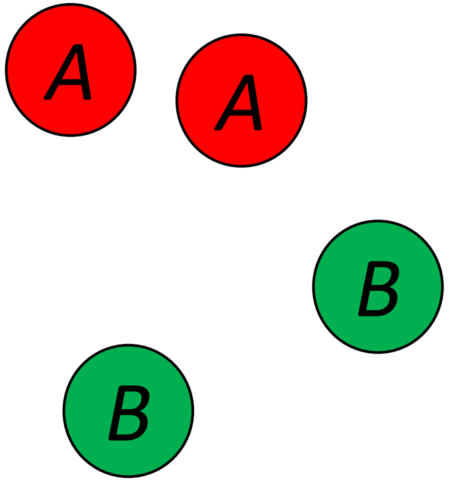
- configuration $\mathbf{x} \in \mathbb{N}^d$: molecular counts of each species
- reaction is applicable to **x** if **x** has enough of each reactant.

What is **possible**:
Example reaction sequence (a.k.a. *execution*)

α: $A+B \rightarrow A+C$

β: $C \rightarrow A+A$

	A	B	C	
x =	(2,	2,	0)	α applicable but not β



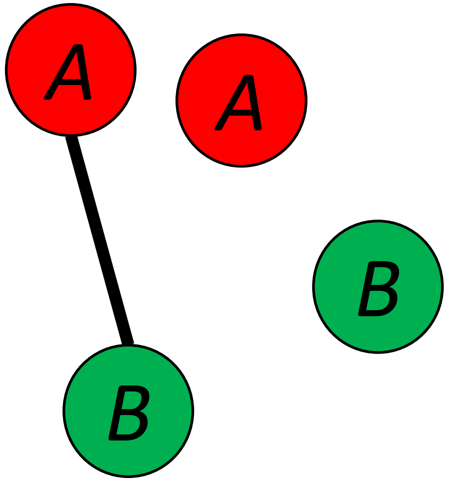
What is **possible**:
Example reaction sequence (a.k.a. *execution*)

α: $\boxed{A+B} \rightarrow A+C$

β: $C \rightarrow A+A$

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

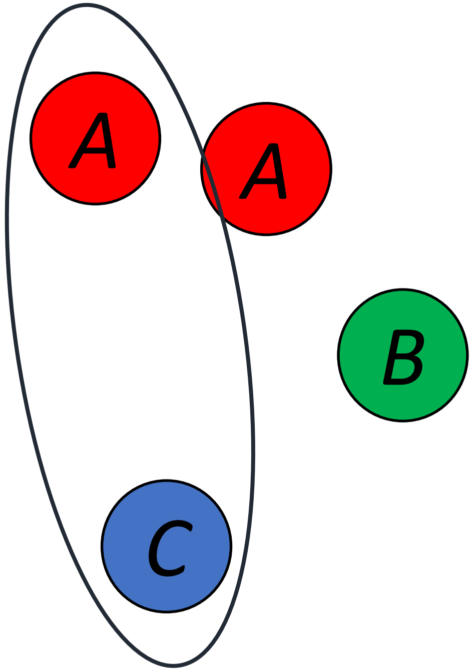
α applicable but not β



What is **possible**:
Example reaction sequence (a.k.a. *execution*)

α: $A+B \rightarrow \boxed{A+C}$

β: $C \rightarrow A+A$

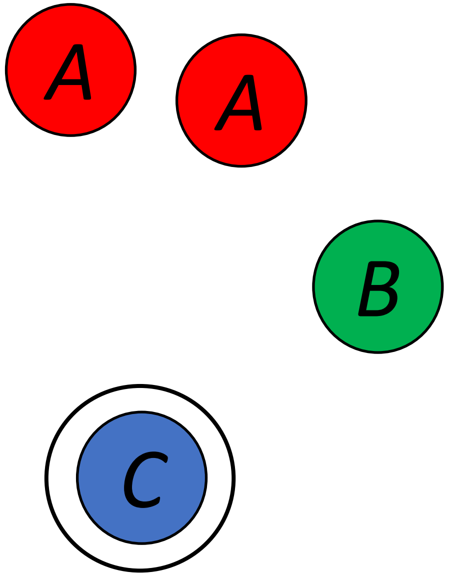


$$
\begin{array}{ccc}
 & A \quad B \quad C & \\
\mathbf{x} = & (2,\ 2,\ 0) & \text{α applicable but not β} \\
 & \alpha \Downarrow & \\
 & (2,\ 1,\ 1) & \text{α,β both applicable}
\end{array}
$$

What is **possible**:
Example reaction sequence (a.k.a. *execution*)

α: $A+B \rightarrow A+C$

β: $\boxed{C} \rightarrow A+A$



$$\begin{array}{ccc} A & B & C \end{array}$$

$\mathbf{x} = (2, 2, 0)$ α applicable but not β

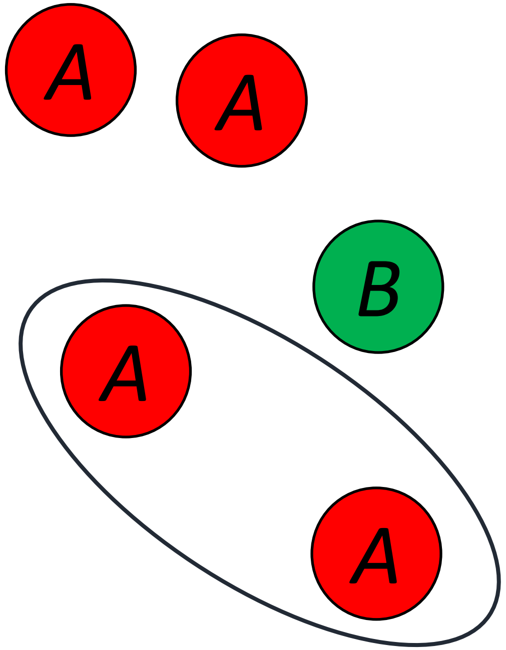
$\alpha \Downarrow$

$(2, 1, 1)$ α,β both applicable

What is **possible**:
Example reaction sequence (a.k.a. *execution*)

α: $A + B \to A + C$

β: $C \to \boxed{A + A}$



$$
\begin{array}{c}
\quad A \quad B \quad C \\
\mathbf{x} = (2,\ 2,\ 0) \\
\alpha \Downarrow \\
(2,\ 1,\ 1) \\
\beta \Downarrow \\
(4,\ 1,\ 0)
\end{array}
$$

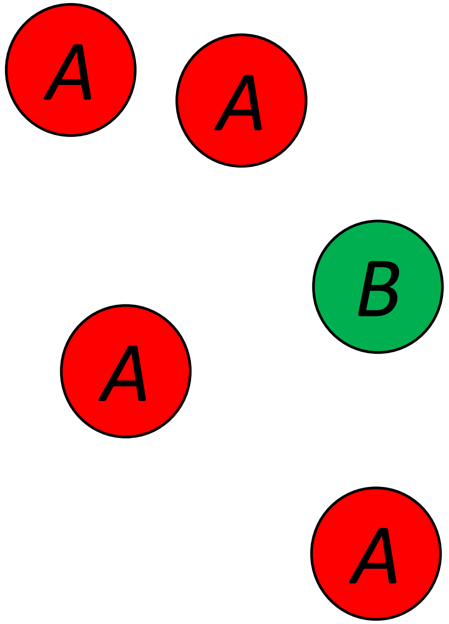
α applicable but not β

α,β both applicable

What is **possible**: Example reaction sequence (a.k.a. *execution*)

α: $A + B \rightarrow A + C$

β: $C \rightarrow A + A$



$$
\begin{array}{rccc}
 & A & B & C \\
x = & (2, & 2, & 0) \\
\end{array}
$$

α applicable but not β

$$\Downarrow \alpha$$

$$(2,\ 1,\ 1)$$

α,β both applicable

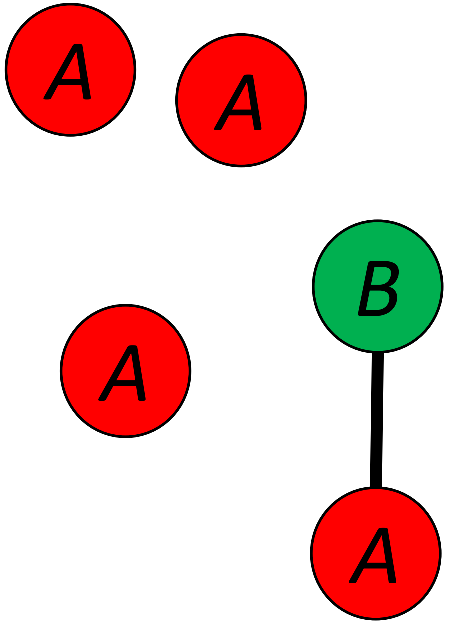
$$\Downarrow \beta \qquad\qquad \searrow \alpha \text{ (another possibility)}$$

$$(4,\ 1,\ 0) \qquad (2,\ 0,\ 2)$$

What is **possible**:
Example reaction sequence (a.k.a. *execution*)

α: $\boxed{A+B} \to A+C$

β: $C \to A+A$



$$
\begin{array}{cccc}
 & A & B & C \\
\mathbf{x} = & (2, & 2, & 0) \\
 & & \alpha \Downarrow & \\
 & (2, & 1, & 1) \\
 & & \beta \Downarrow & \\
 & (4, & 1, & 0)
\end{array}
$$

α applicable but not β

α,β both applicable

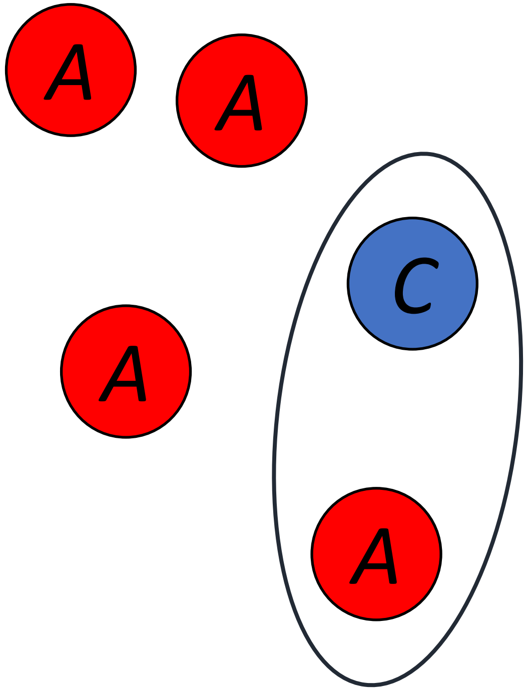
$\searrow$ α (another possibility)

(2, 0, 2)

What is possible:
Example reaction sequence (a.k.a. *execution*)

α: $A+B \to \boxed{A+C}$

β: $C \to A+A$



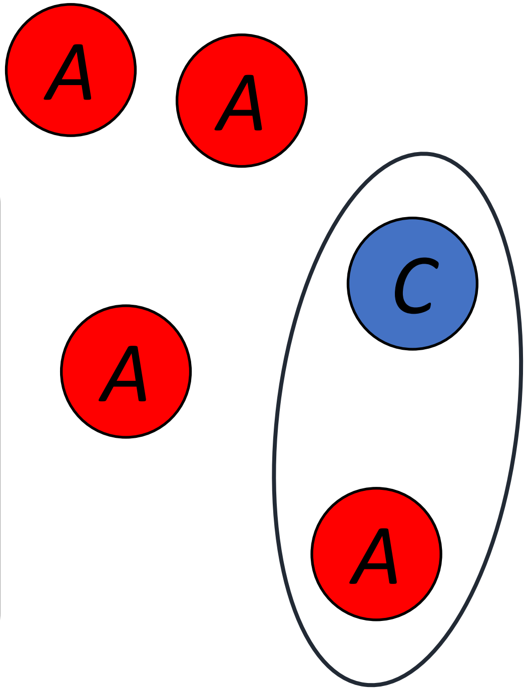
$$
\begin{array}{lcccl}
 & A & B & C & \\
\mathbf{x} = & (2, & 2, & 0) & \text{α applicable but not β} \\
 & & \alpha \Downarrow & & \\
 & (2, & 1, & 1) & \text{α,β both applicable} \\
 & & \beta \Downarrow & & \searrow \alpha \text{ (another possibility)} \\
 & (4, & 1, & 0) & (2,\ 0,\ 2) \\
 & & \alpha \Downarrow & & \\
 & (4, & 0, & 1) & \\
 & & \ldots & &
\end{array}
$$

What is possible:

Example reaction sequence (a.k.a. *execution*)

α: $A+B \rightarrow \boxed{A+C}$

β: $C \rightarrow A+A$



	A	B	C	
$\mathbf{x}$ =	(2,	2,	0)	α applicable but not β

α ⇓

(2,	1,	1)	α,β both applicable

β ⇓ ⇘ α (another possibility)

(4, 1, 0) (2, 0, 2)

α ⇓

(4, 0, 1)

…

Formally, an execution is a sequence of *configurations* $\mathbf{x}_1$, $\mathbf{x}_2$, … such that each $\mathbf{x}_i \Longrightarrow \mathbf{x}_{i+1}$ by a single reaction.
If initial configuration $\mathbf{x}_1$ is understood, the sequence of *reactions* is sometimes called the execution.

Some simple reactions

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

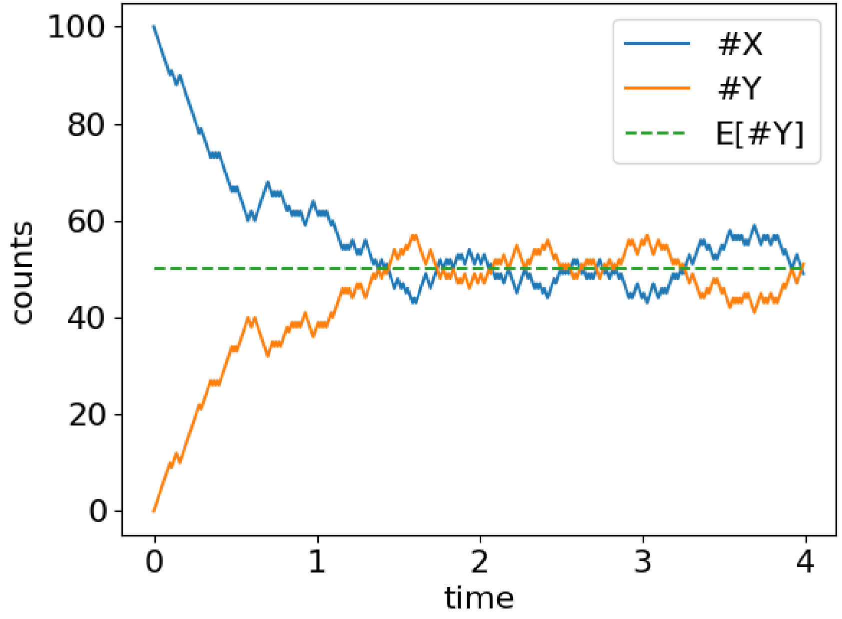
start with n copies of molecule X

Some simple reactions

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

start with n copies of molecule X

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



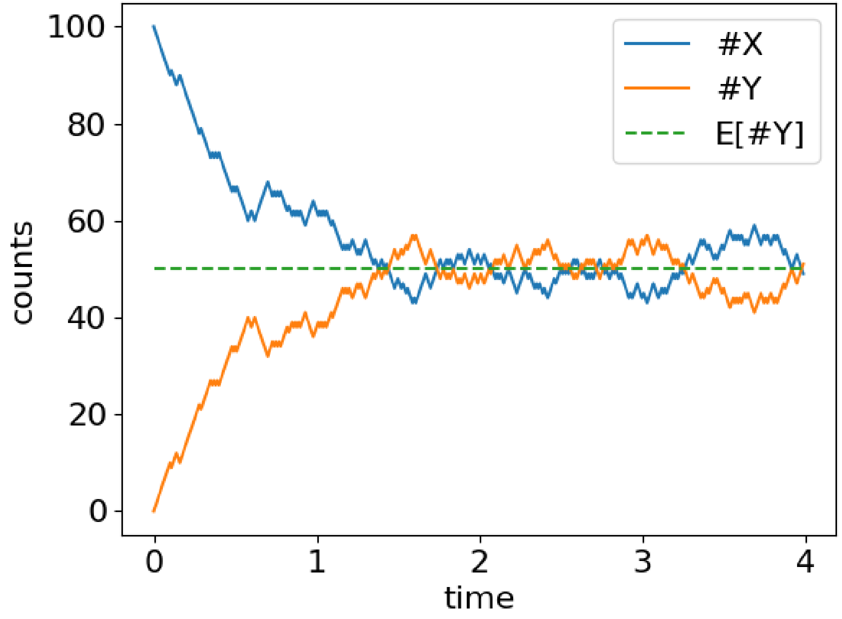
Some simple reactions

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

Count of Y never stabilizes

start with n copies of molecule X

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



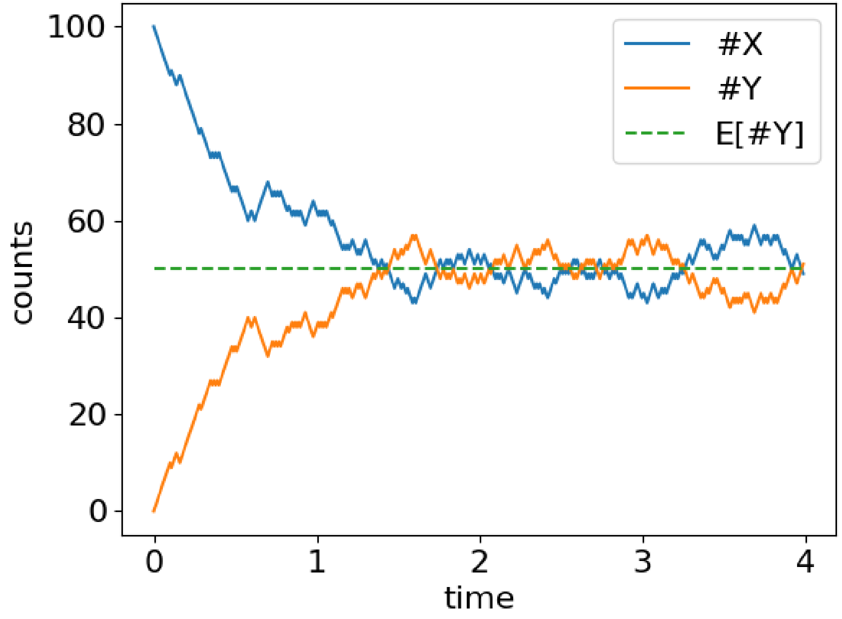
Some simple reactions

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

Count of Y never stabilizes

start with n copies of molecule X

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



$$X \xrightarrow{1} Y$$

$$X \xrightarrow{1}$$

Some simple reactions

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

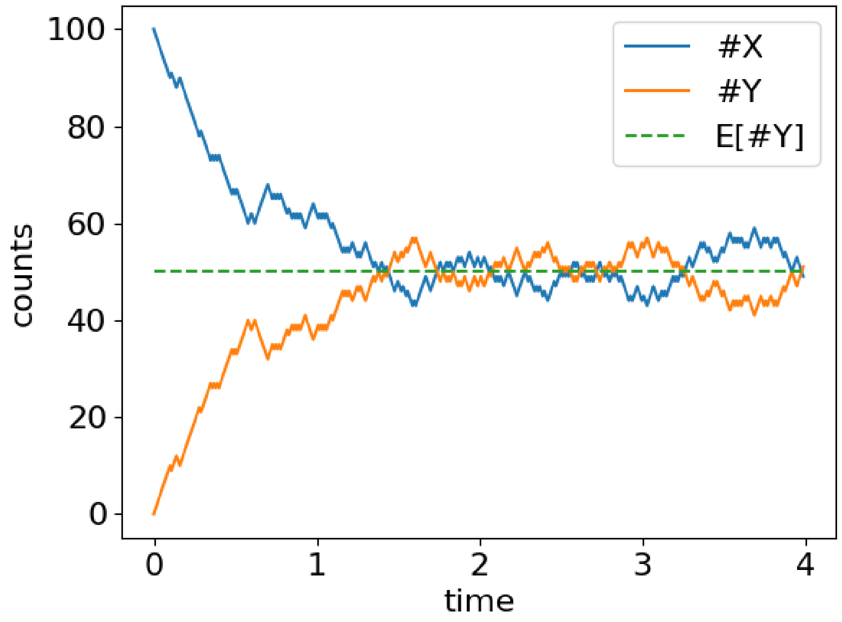
Count of Y never stabilizes

start with n copies of molecule X

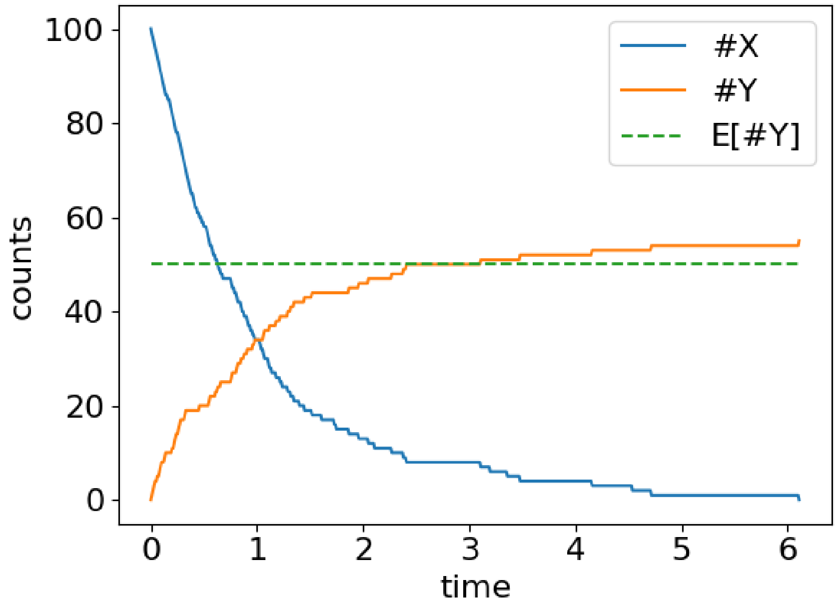
$X \xrightarrow{1} Y$

$X \xrightarrow{1}$

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



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



Some simple reactions

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

Count of Y never stabilizes

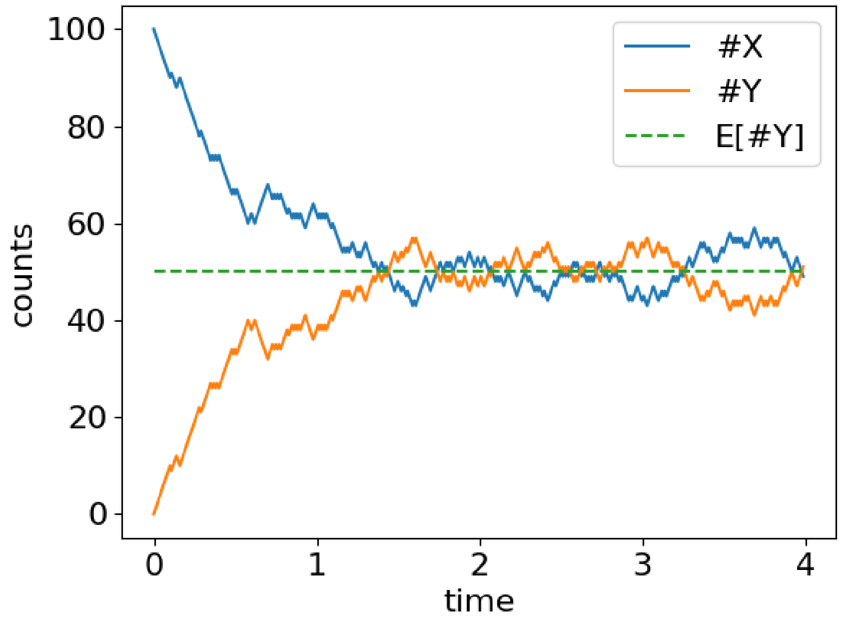
start with n copies of molecule X

$X \xrightarrow{1} Y$

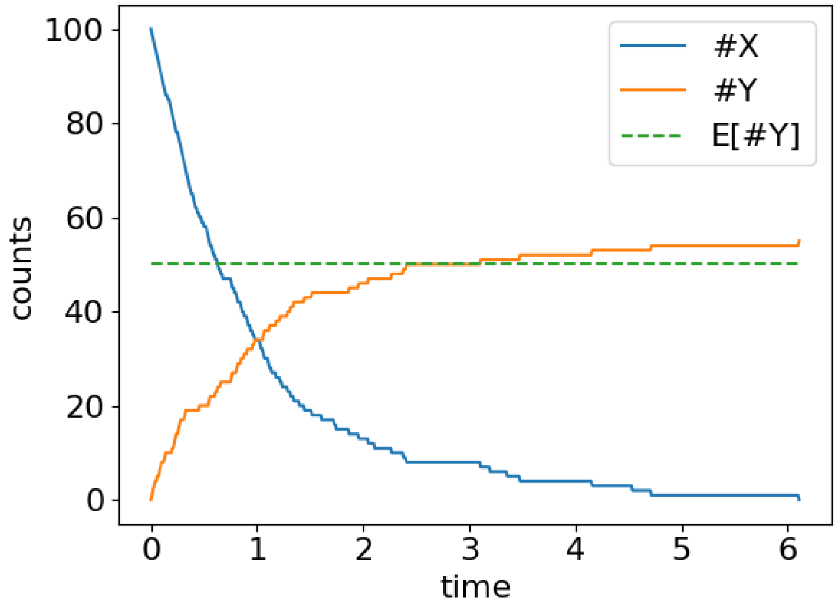
$X \xrightarrow{1}$

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

#Y = $n/2$ expected at equilibrium



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



Some simple reactions

Worse yet, both depend crucially on rate constants.

$$X \underset{\cancel{1}\ 2}{\overset{1}{\rightleftarrows}} Y$$

Count of Y never stabilizes

$$X \xrightarrow{1} Y$$
$$X \xrightarrow{\cancel{1}\ 2}$$

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

start with n copies of molecule X

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

#Y stabilizes, with expected value ~~$n/2$~~ $n/3$

Some simple reactions

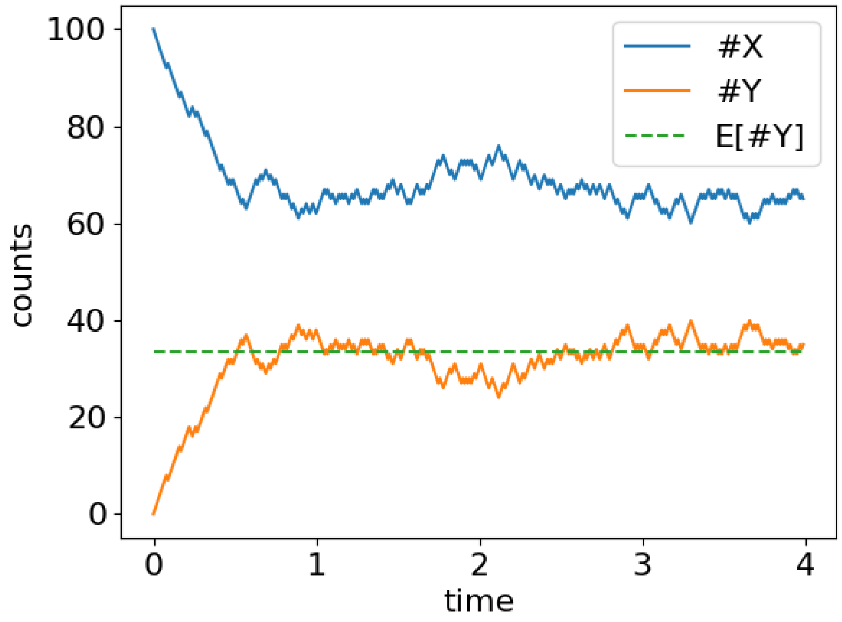
Worse yet, both depend crucially on rate constants.

$X \underset{\cancel{1}\ 2}{\overset{1}{\rightleftharpoons}} Y$

Count of Y never stabilizes

start with n copies of molecule X

$\#Y = \overset{n/3}{\cancel{n/2}}$ expected at equilibrium

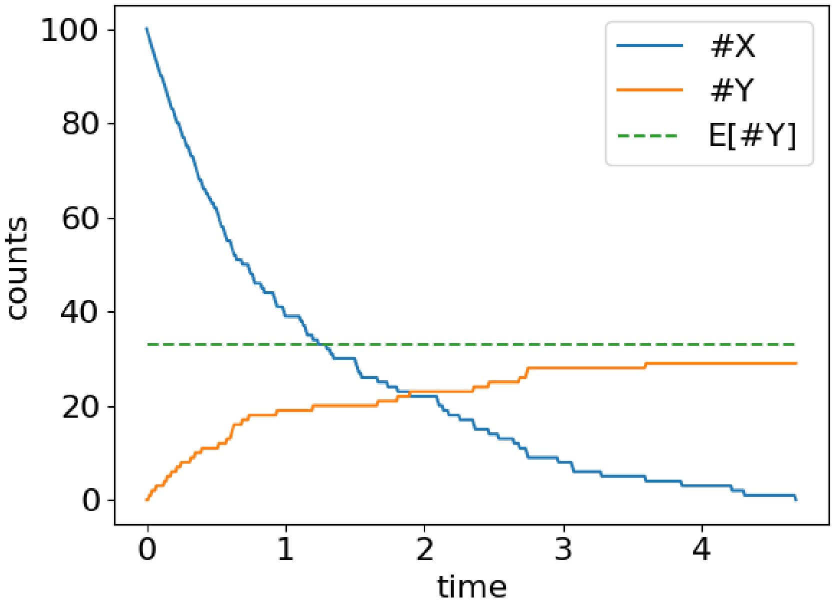


$X \xrightarrow{1} Y$

$X \xrightarrow{\cancel{1}\ 2}$

Count of Y stabilizes, but <u>not</u> to a deterministic value based on initial count of X

$\#Y$ <u>stabilizes</u>, with expected value $\overset{n/3}{\cancel{n/2}}$

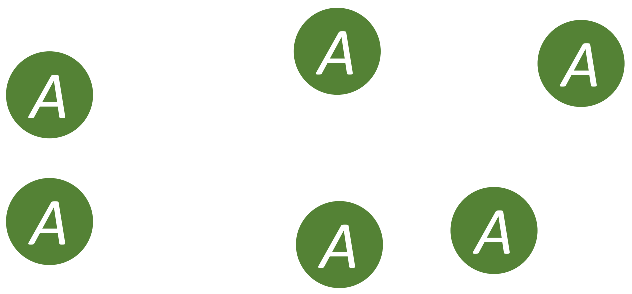


Examples of **stable** (*rate-independent*) CRN computation

Examples of function computation

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

goal: end up with $a/2$ copies of Y

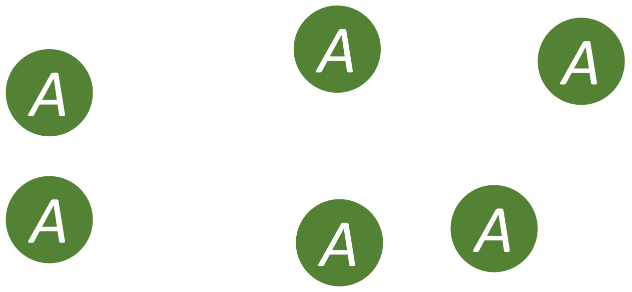


Examples of function computation

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

goal: end up with $a/2$ copies of Y

$$2A \rightarrow Y$$

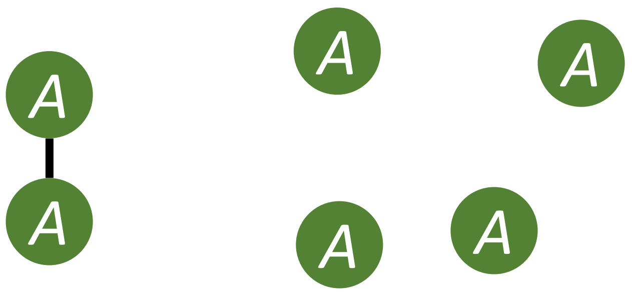


Examples of function computation

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

goal: end up with $a/2$ copies of Y

$$2A \rightarrow Y$$

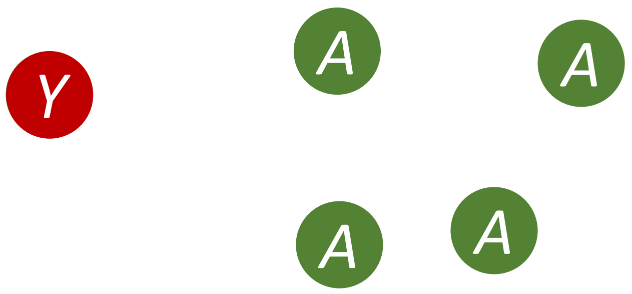


Examples of function computation

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

goal: end up with $a/2$ copies of Y

$$2A \rightarrow Y$$

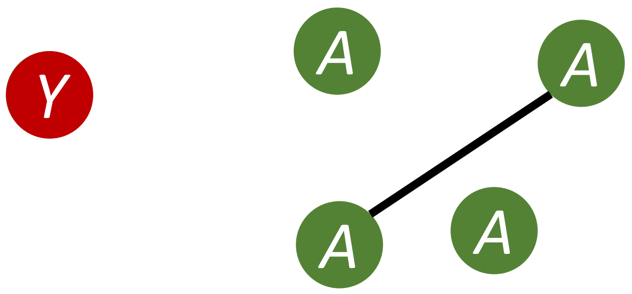


Examples of function computation

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

goal: end up with $a/2$ copies of Y

$$2A \rightarrow Y$$



Examples of function computation

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

goal: end up with $a/2$ copies of Y

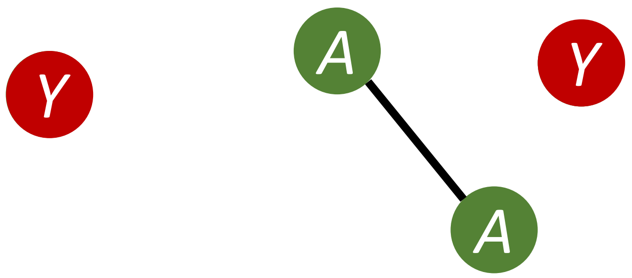
$$2A \rightarrow Y$$

Examples of function computation

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

goal: end up with $a/2$ copies of Y

$$2A \to Y$$



Examples of function computation

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

goal: end up with $a/2$ copies of Y

$$2A \rightarrow Y$$

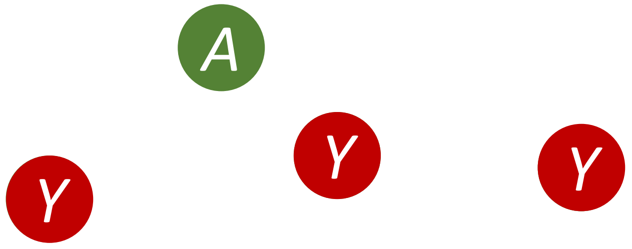
Examples of function computation

??

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

goal: end up with $a/2$ copies of Y

$$2A \rightarrow Y$$

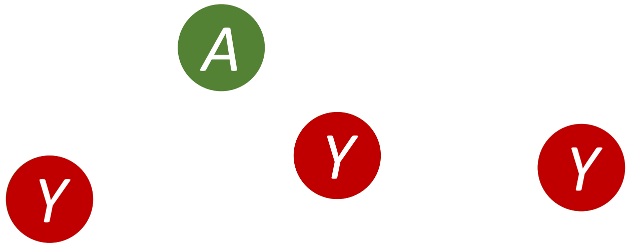


Examples of function computation

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

goal: end up with $a/2$ copies of Y

$$2A \rightarrow Y$$

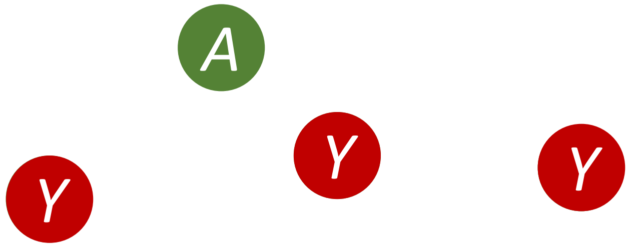


Examples of function computation

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

goal: end up with $a/2$ copies of Y

$$2A \rightarrow Y$$



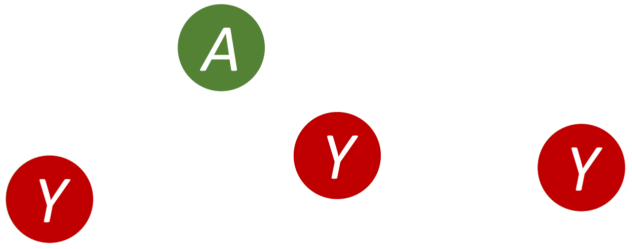
multiplication by 2: $f(a) = 2a$

Examples of function computation

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

goal: end up with $a/2$ copies of Y

$$2A \rightarrow Y$$



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

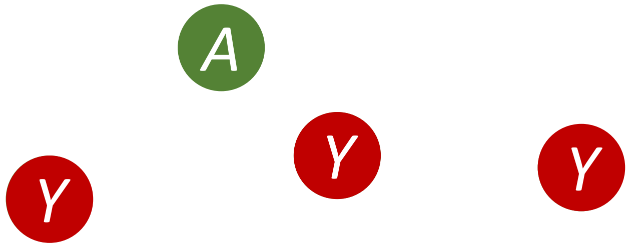
$$A \rightarrow 2Y$$

Examples of function computation

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

goal: end up with $a/2$ copies of Y

$$2A \rightarrow Y$$



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

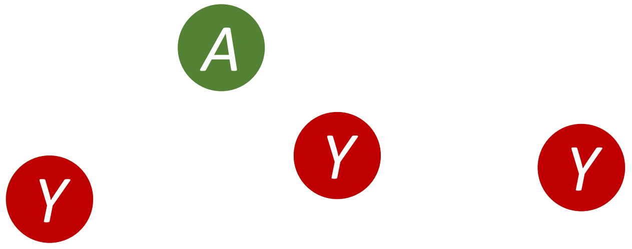
$$A \rightarrow 2Y$$

Examples of function computation

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

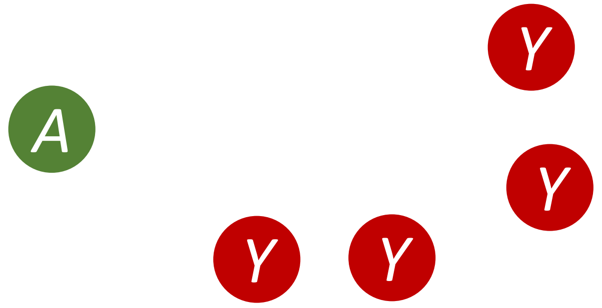
goal: end up with $a/2$ copies of Y

$$2A \rightarrow Y$$



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

$$A \rightarrow 2Y$$

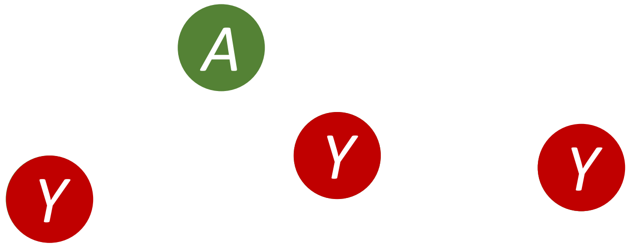


Examples of function computation

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

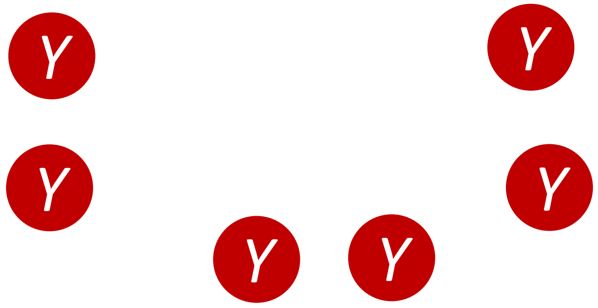
goal: end up with $a/2$ copies of Y

$$2A \rightarrow Y$$



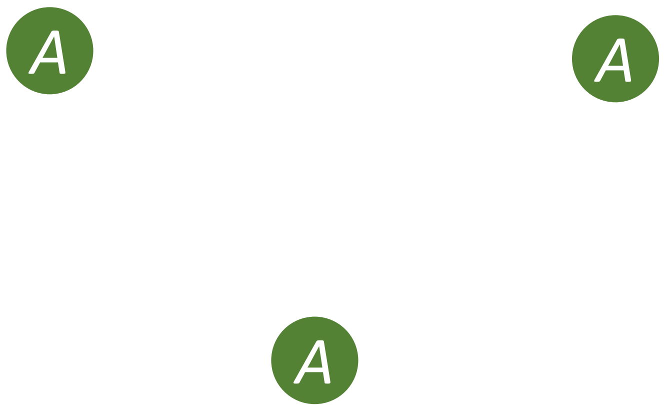
multiplication by 2: $f(a) = 2a$

$$A \rightarrow 2Y$$



Examples of function computation

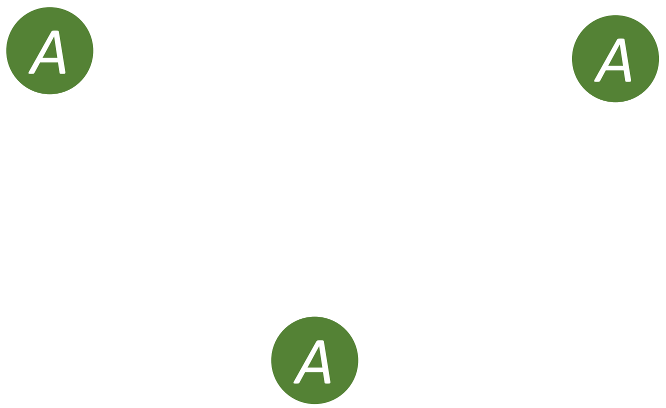
multiplication by 3: $f(a) = 3a$



Examples of function computation

multiplication by 3: $f(a) = 3a$

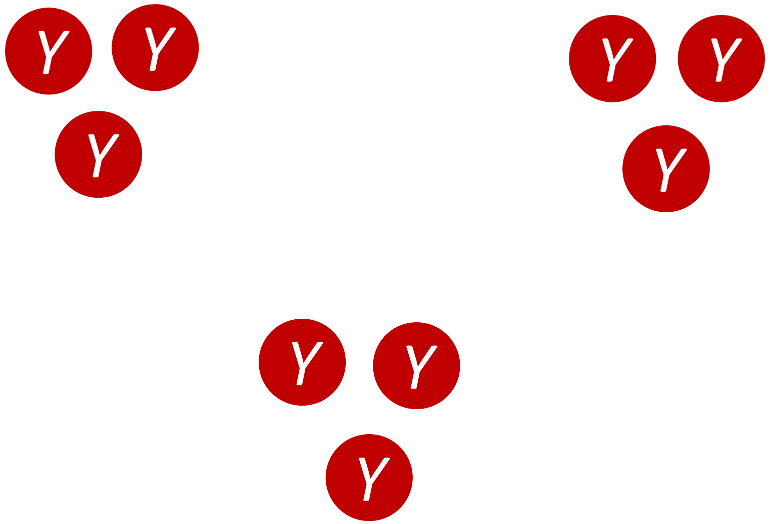
$$A \rightarrow 3Y$$



Examples of function computation

multiplication by 3: $f(a) = 3a$

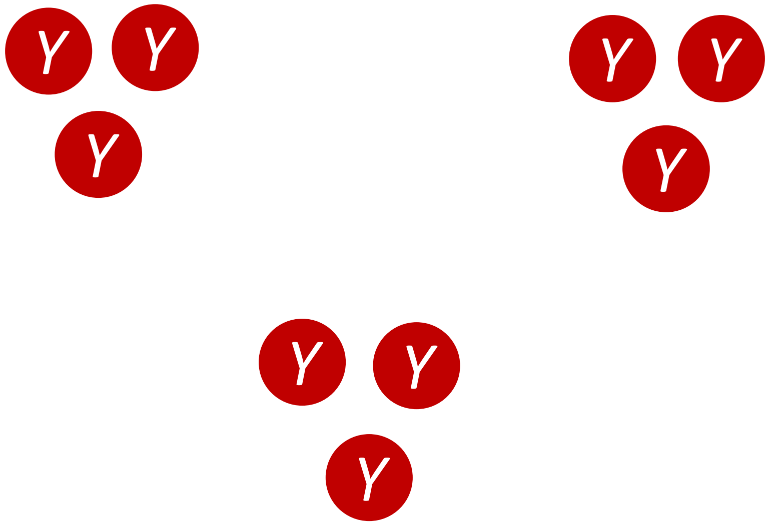
$$A \rightarrow 3Y$$



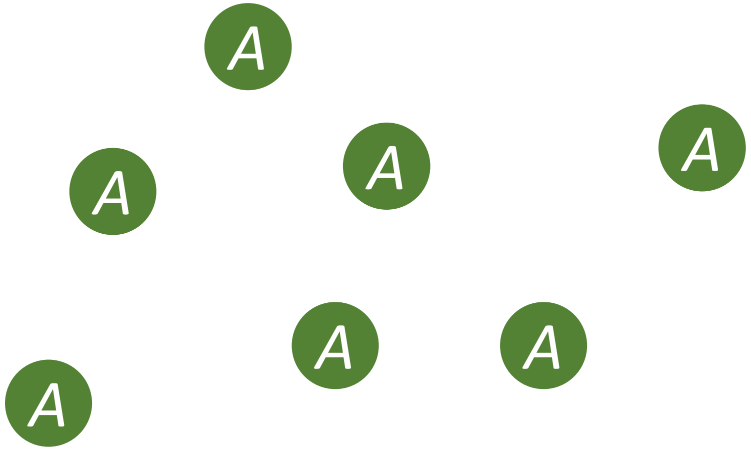
Examples of function computation

multiplication by 3: $f(a) = 3a$

$$A \rightarrow 3Y$$



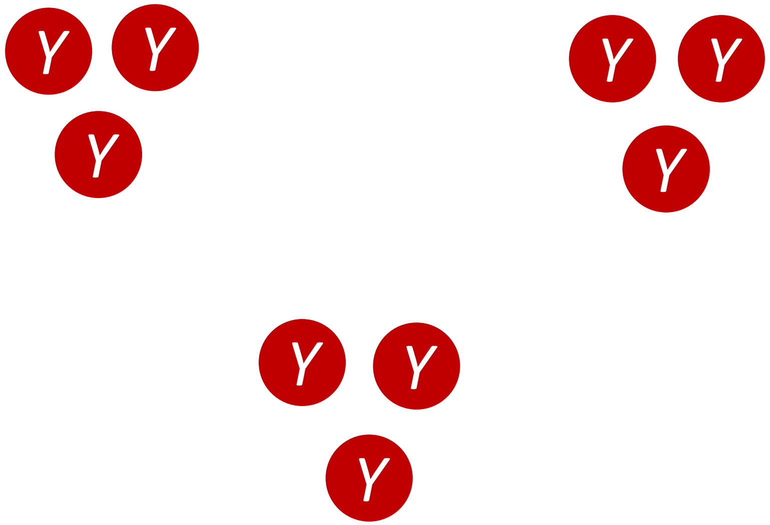
division by 3: $f(a) = \lfloor a/3 \rfloor$



Examples of function computation

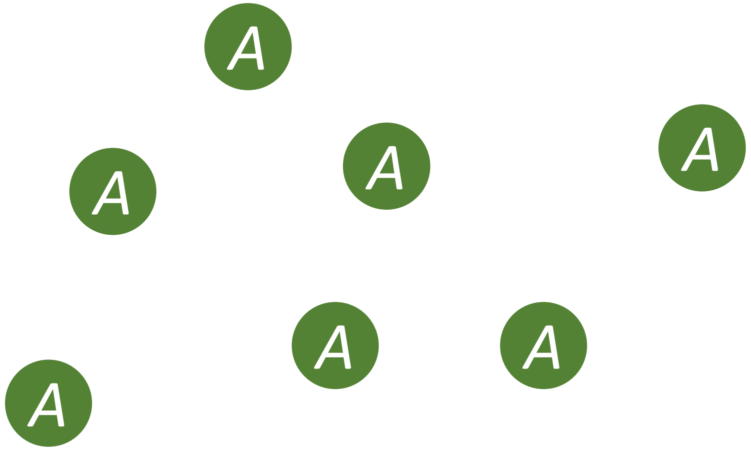
multiplication by 3: $f(a) = 3a$

$$A \rightarrow 3Y$$



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

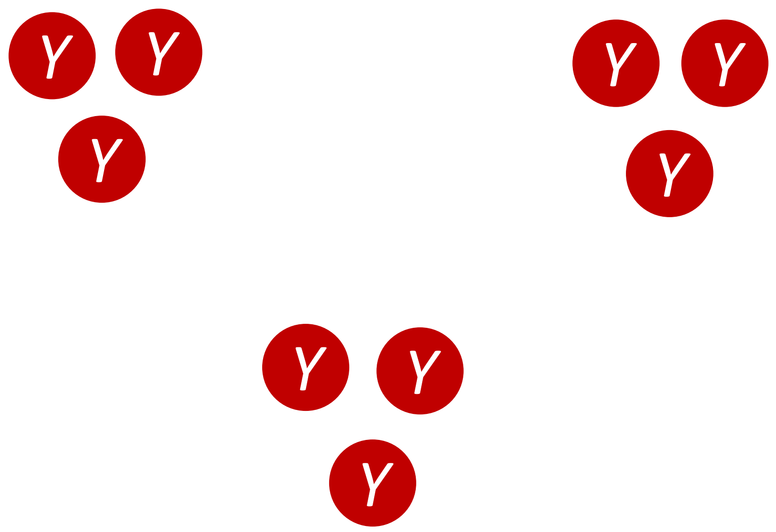
$$3A \rightarrow Y$$



Examples of function computation

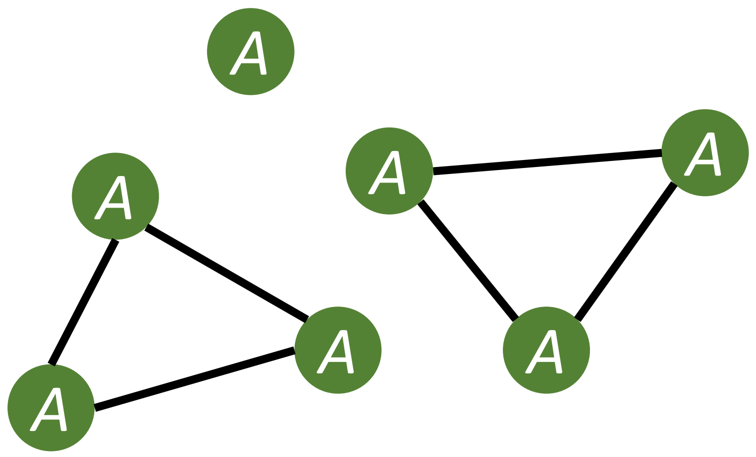
multiplication by 3: $f(a) = 3a$

$$A \rightarrow 3Y$$



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

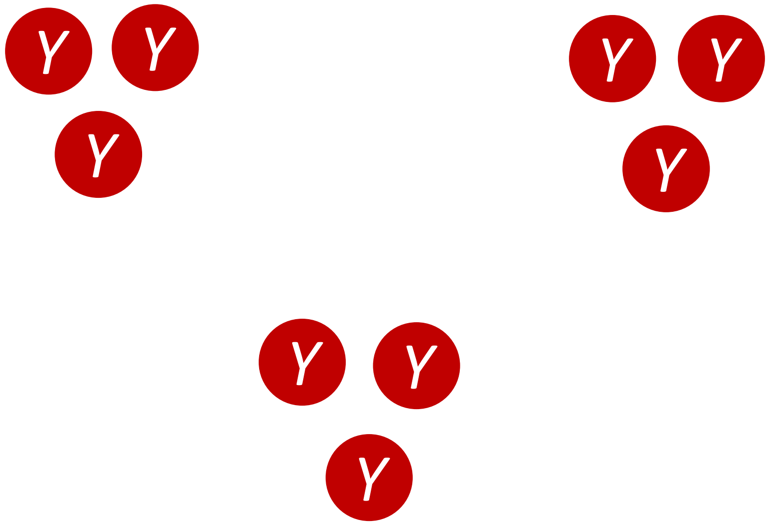
$$3A \rightarrow Y$$



Examples of function computation

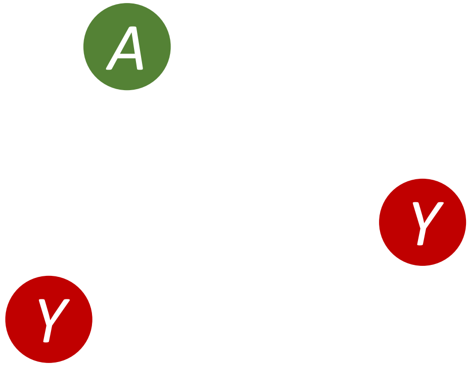
multiplication by 3: $f(a) = 3a$

$$A \rightarrow 3Y$$



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

$$3A \rightarrow Y$$



Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$

Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

$$\boxed{A \rightarrow Y + Y'}$$
$$Y' \rightarrow 2Y$$

Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

$$A \rightarrow Y + Y'$$

$$Y' \rightarrow 2Y$$

Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

$$A \to Y + Y'$$

$$\boxed{Y' \to 2Y}$$

Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$

Examples of function computation

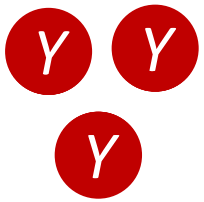
$f(a) = 3a$ using (≤ 2)-product reactions

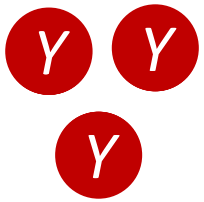
$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$

Examples of function computation

$f(a) = 3a$ using ($\leq$ 2)-product reactions

$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$



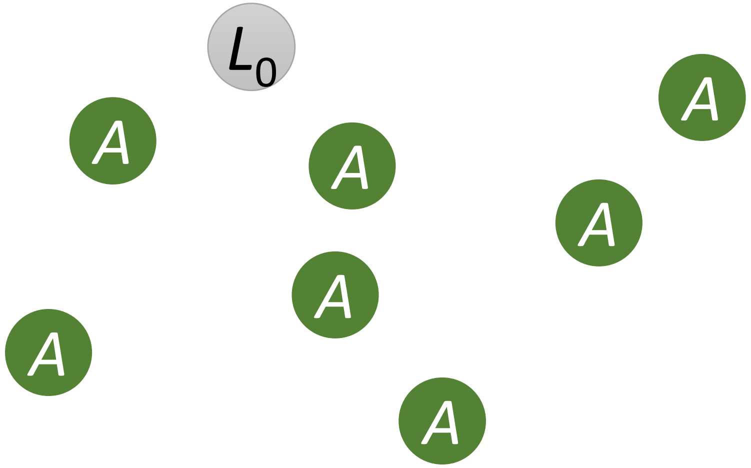


Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$

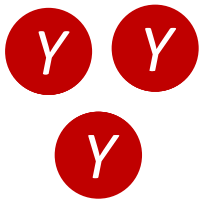
$f(a) = \lfloor a/3 \rfloor$ using bimolecular ((≤ 2)-reactant) reactions, starting in config $\{ 1\ L_0, a\ A \}$ (a.k.a., *leader-driven*)

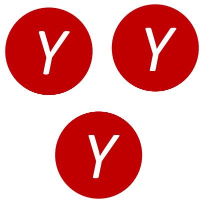


Examples of function computation

$f(a) = 3a$ using ($\leq$ 2)-product reactions

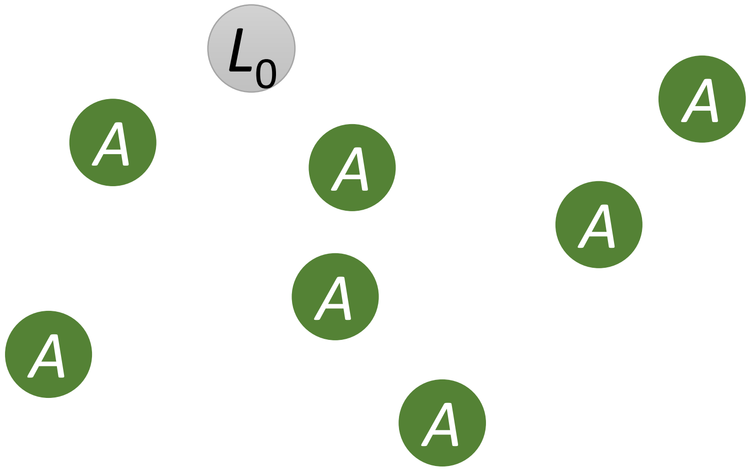
$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$





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

$$L_0 + A \rightarrow L_1$$
$$L_1 + A \rightarrow L_2$$
$$L_2 + A \rightarrow L_0 + Y$$



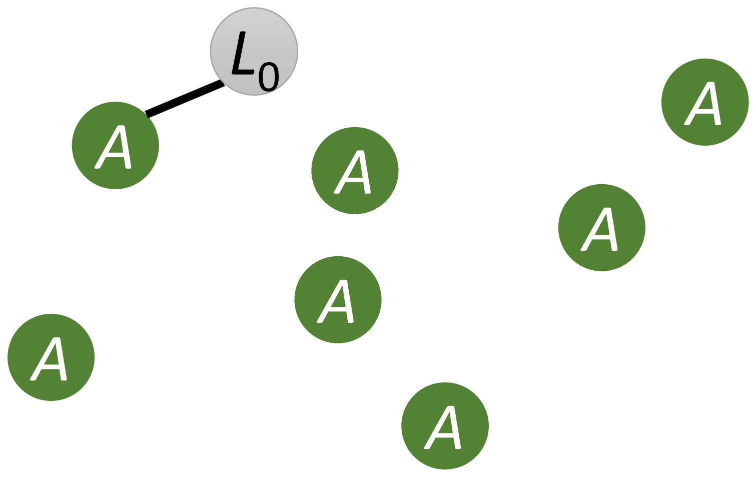
Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$

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

$$L_0 + A \rightarrow L_1$$
$$L_1 + A \rightarrow L_2$$
$$L_2 + A \rightarrow L_0 + Y$$



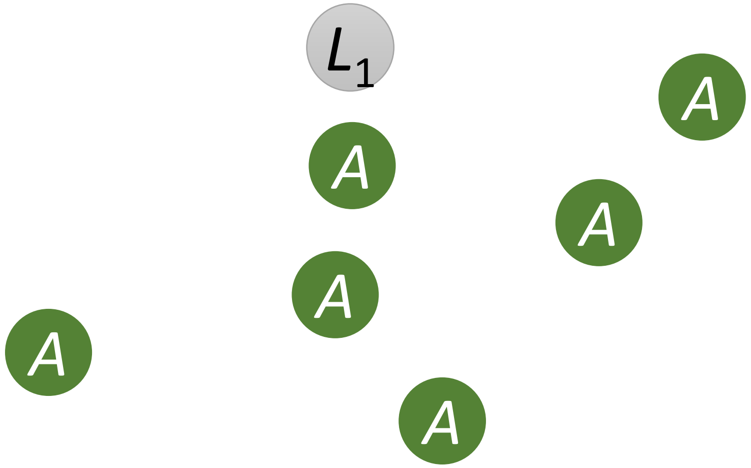
Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$

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

$$L_0 + A \rightarrow L_1$$
$$L_1 + A \rightarrow L_2$$
$$L_2 + A \rightarrow L_0 + Y$$



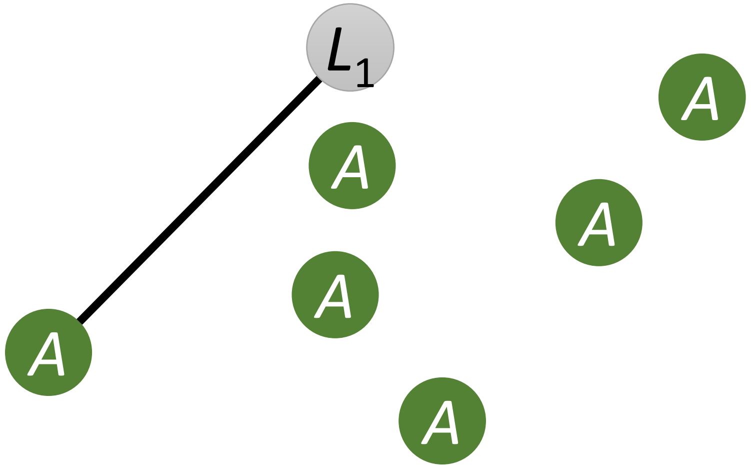
Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$

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

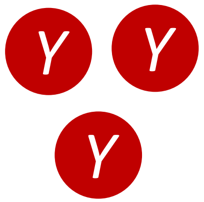
$$L_0 + A \rightarrow L_1$$
$$L_1 + A \rightarrow L_2$$
$$L_2 + A \rightarrow L_0 + Y$$

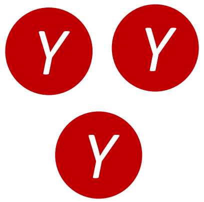


Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

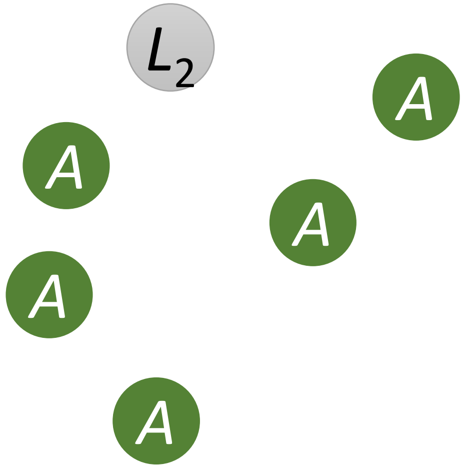
$$A \to Y + Y'$$
$$Y' \to 2Y$$





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

$$L_0 + A \to L_1$$
$$L_1 + A \to L_2$$
$$L_2 + A \to L_0 + Y$$



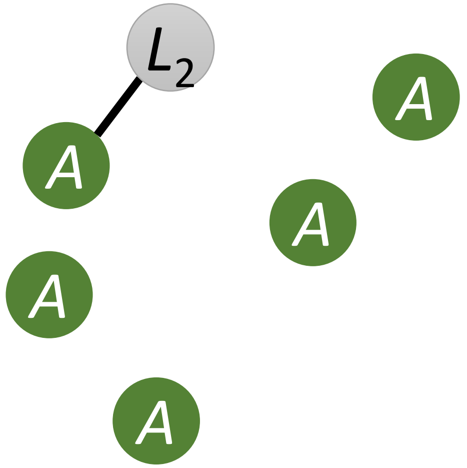
Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

$$A \to Y + Y'$$
$$Y' \to 2Y$$

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

$$L_0 + A \to L_1$$
$$L_1 + A \to L_2$$
$$L_2 + A \to L_0 + Y$$



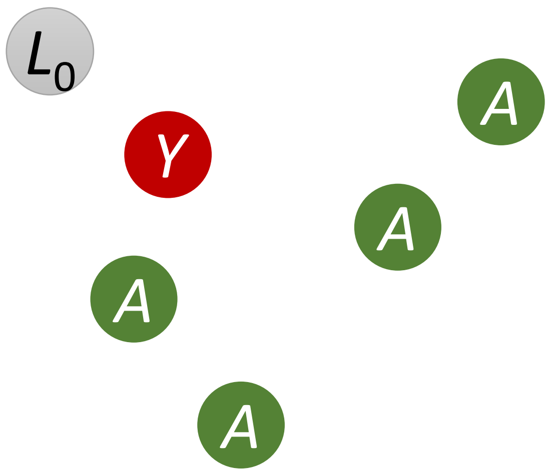
Examples of function computation

$f(a) = 3a$ using ($\leq$ 2)-product reactions

$$A \to Y + Y'$$
$$Y' \to 2Y$$

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

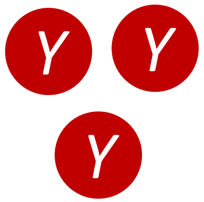
$$L_0 + A \to L_1$$
$$L_1 + A \to L_2$$
$$L_2 + A \to L_0 + Y$$

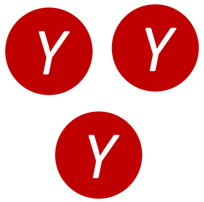


Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

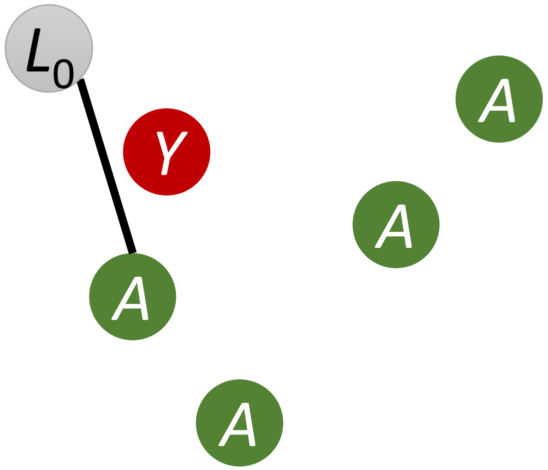
$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$





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

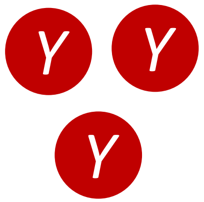
$$L_0 + A \rightarrow L_1$$
$$L_1 + A \rightarrow L_2$$
$$L_2 + A \rightarrow L_0 + Y$$

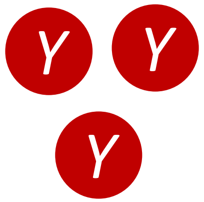


Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

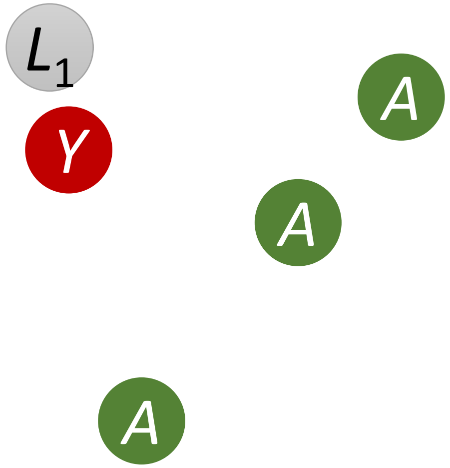
$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$





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

$$L_0 + A \rightarrow L_1$$
$$L_1 + A \rightarrow L_2$$
$$L_2 + A \rightarrow L_0 + Y$$



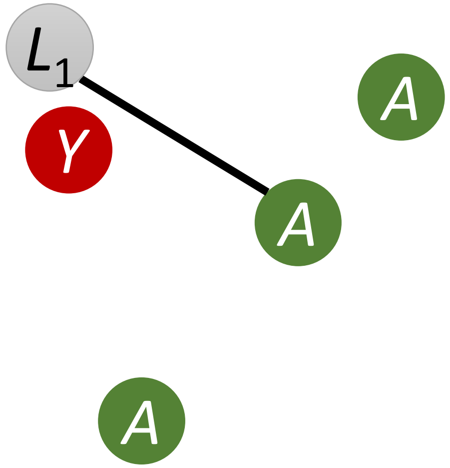
Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$

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

$$L_0 + A \rightarrow L_1$$
$$L_1 + A \rightarrow L_2$$
$$L_2 + A \rightarrow L_0 + Y$$



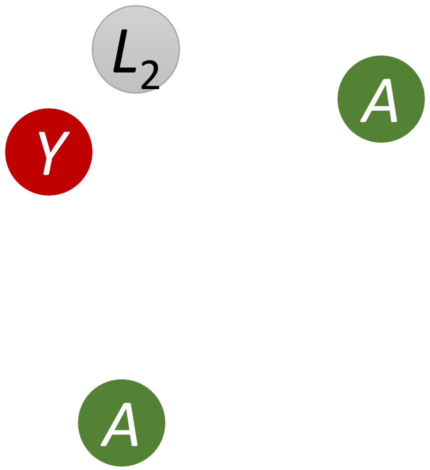
Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$

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

$$L_0 + A \rightarrow L_1$$
$$L_1 + A \rightarrow L_2$$
$$L_2 + A \rightarrow L_0 + Y$$



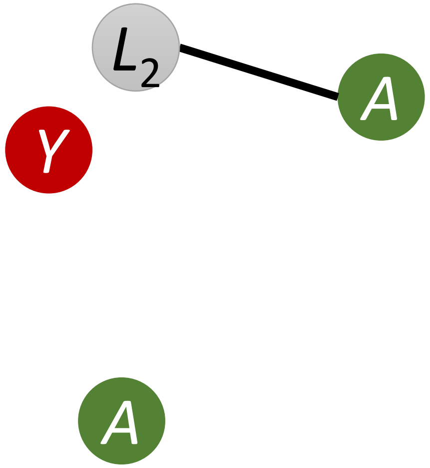
Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$

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

$$L_0 + A \rightarrow L_1$$
$$L_1 + A \rightarrow L_2$$
$$L_2 + A \rightarrow L_0 + Y$$



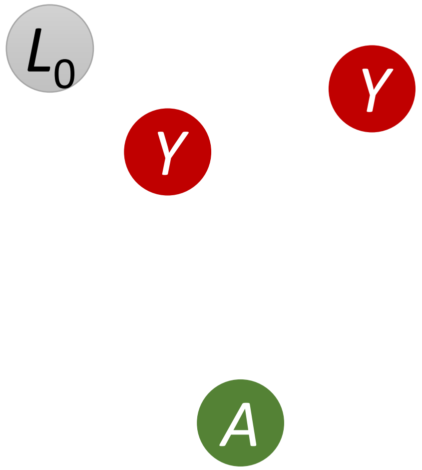
Examples of function computation

$f(a) = 3a$ using ($\leq$ 2)-product reactions

$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$

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

$$L_0 + A \rightarrow L_1$$
$$L_1 + A \rightarrow L_2$$
$$L_2 + A \rightarrow L_0 + Y$$



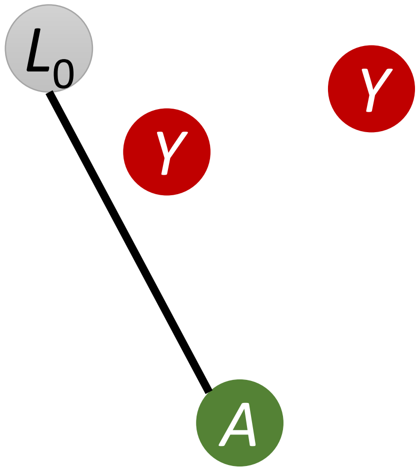
Examples of function computation

$f(a) = 3a$ using (≤ 2)-product reactions

$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$

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

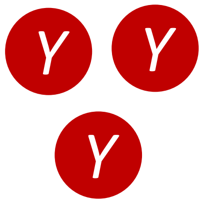
$$L_0 + A \rightarrow L_1$$
$$L_1 + A \rightarrow L_2$$
$$L_2 + A \rightarrow L_0 + Y$$

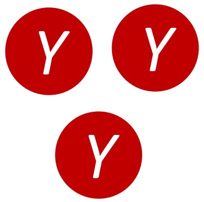


Examples of function computation

$f(a) = 3a$ using ($\leq$ 2)-product reactions

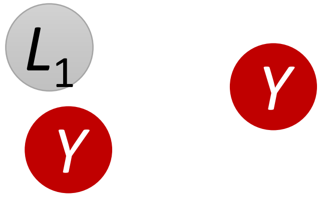
$$A \rightarrow Y + Y'$$
$$Y' \rightarrow 2Y$$





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

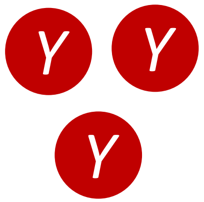
$$L_0 + A \rightarrow L_1$$
$$L_1 + A \rightarrow L_2$$
$$L_2 + A \rightarrow L_0 + Y$$

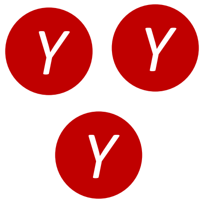


Examples of function computation

$f(a) = 3a$ using ($\leq$ 2)-product reactions

$$A \to Y + Y'$$
$$Y' \to 2Y$$

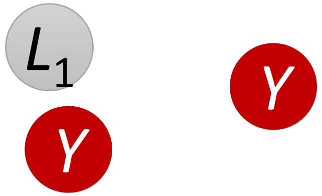




$f(a) = \lfloor a/3 \rfloor$ using bimolecular (($\leq$ 2)-reactant) reactions, starting in config $\{ 1\ L_0,\ a\ A \}$ (a.k.a., *leader-driven*)

$$L_0 + A \to L_1$$
$$L_1 + A \to L_2$$
$$L_2 + A \to L_0 + Y$$

ends with 1 copy of L_i for i = ???



Examples of function computation

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

Examples of function computation

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

$$A + A \rightarrow A_2$$
$$A_2 + A \rightarrow Y$$

Examples of function computation

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

$$A + A \to A_2$$
$$A_2 + A \to Y$$
$$A_2 + A_2 \to A + Y$$

Examples of function computation

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

$$A + A \to A_2$$
$$A_2 + A \to Y$$
$$A_2 + A_2 \to A + Y$$

Calling $A = A_1$, in general to divide by constant c:

$A_i + A_j \to A_k$	if $i+j < c$, where $k = i + j$
$A_i + A_j \to A_k + Y$	if $i+j > c$, where $k = i + j - c$
$A_i + A_j \to Y$	if $i+j = c$

Examples of function computation

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

$$A + A \to A_2$$
$$A_2 + A \to Y$$
$$A_2 + A_2 \to A + Y$$

Calling $A = A_1$, in general to divide by constant c:

$A_i + A_j \to A_k$ if $i+j < c$, where $k = i + j$

$A_i + A_j \to A_k + Y$ if $i+j > c$, where $k = i + j - c$

$A_i + A_j \to Y$ if $i+j = c$

i.e., A's start with 1 "ball" and pass balls to each other; whenever someone gets ≥ c balls, throw away c balls and produce a Y

Examples of function computation

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

Examples of function computation

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

$A \rightarrow Y$

$B \rightarrow Y$

Examples of function computation

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

$A \rightarrow Y$

$B \rightarrow Y$

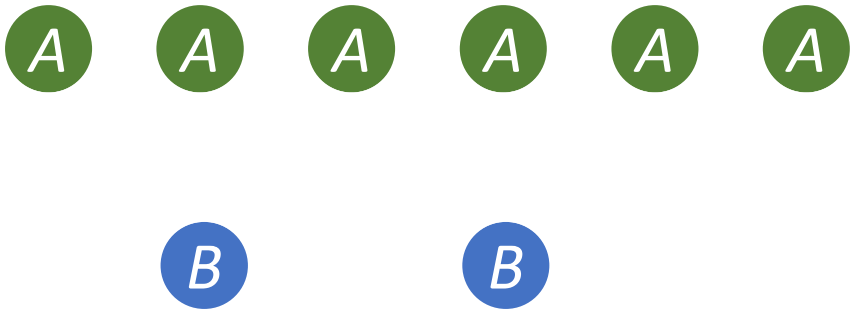
Examples of function computation

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

$A \rightarrow Y$

$B \rightarrow Y$

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



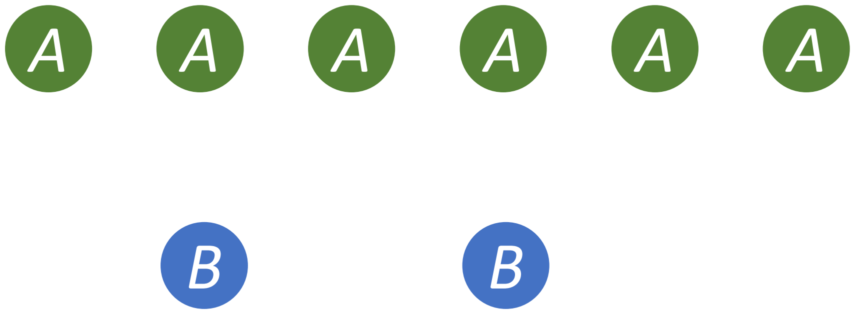
Examples of function computation

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

$$A \to Y$$
$$B \to Y$$

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

$$A \to Y$$
$$B+Y \to \emptyset$$



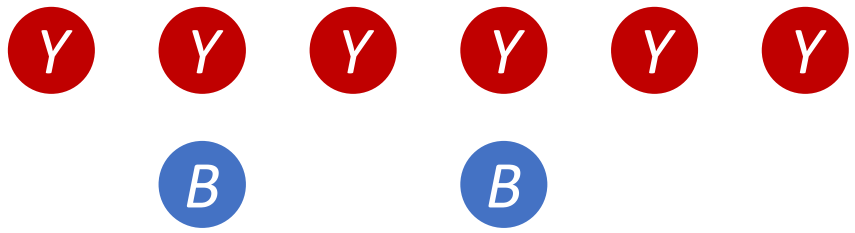
Examples of function computation

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

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

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

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



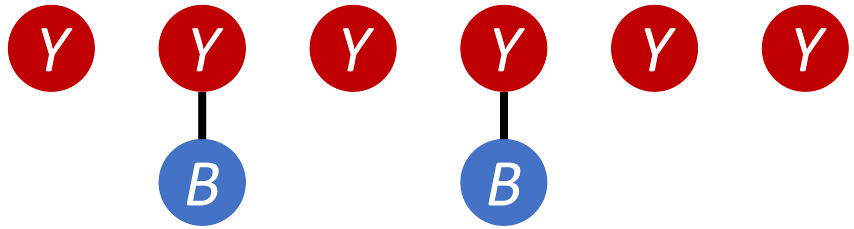
Examples of function computation

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

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

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

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



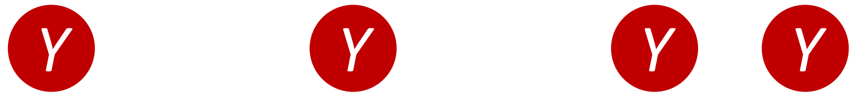
Examples of function computation

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

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

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

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



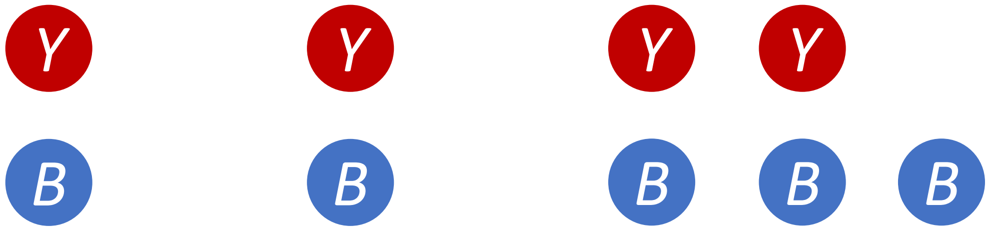
Examples of function computation

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

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

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

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



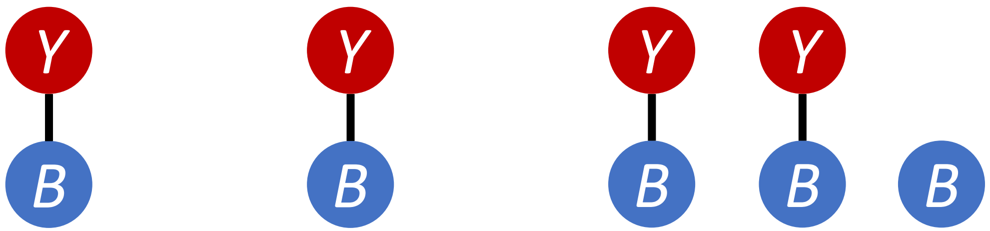
Examples of function computation

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

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

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

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



Examples of function computation

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

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

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

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

Examples of function computation

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

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

???

subtraction: $f(a,b) = \cancel{a-b}$

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

Examples of function computation

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

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

subtraction: $f(a,b) = \cancel{a-b}$ $\max(0, a-b)$

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

Examples of function computation

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

Examples of function computation

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

$$A \rightarrow 3Y$$

$$B + Y \rightarrow \emptyset$$

Examples of function computation

composition: $f(a,b) = \cancel{3a-b}$???

$3a-(b/2)$

$$A \to 3Y$$

$$B + Y \to \emptyset$$

Examples of function computation

composition: $f(a,b) = \cancel{3a-b}$???
$3a-(b/2)$

$A \rightarrow 3Y$

$2B+Y \rightarrow \emptyset$

Examples of function computation

composition: $f(a,b) = \cancel{3a-b}$???
$3a-(b/2)$

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

only linear functions computable?

Examples of function computation

composition: $f(a,b)$ = $\cancel{3a-b}$???

$3a-(b/2)$

$$A \rightarrow 3Y$$

$$2B+Y \rightarrow \emptyset$$

only linear functions computable?

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

Examples of function computation

composition: $f(a,b) = \cancel{3a-b}$???
$3a-(b/2)$

$$A \to 3Y$$

$$2B+Y \to \emptyset$$

only linear functions computable?

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

$$A+B \to Y$$

Examples of function computation

composition: $f(a,b) = \cancel{3a-b}$???

$3a-(b/2)$

$$A \rightarrow 3Y$$

$$2B+Y \rightarrow \emptyset$$

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

only linear functions computable?

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

$$A+B \rightarrow Y$$

Examples of function computation

composition: $f(a,b) = \cancel{3a-b}$???

$3a-(b/2)$

$A \rightarrow 3Y$

$2B + Y \rightarrow \emptyset$

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

only linear functions computable?

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

$A + B \rightarrow Y$

Examples of function computation

composition: $f(a,b) = \cancel{3a-b}$???
$3a-(b/2)$

$$A \to 3Y$$
$$2B+Y \to \emptyset$$

only linear functions computable?

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

$$A+B \to Y$$

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

$$A \to Y+A_2$$
$$B \to Y+B_2$$

addition

Examples of function computation

composition: $f(a,b)$ = $\cancel{3a-b}$???

$3a-(b/2)$

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

only linear functions computable?

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

$$A+B \rightarrow Y$$

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

$$A \rightarrow Y+A_2$$
$$B \rightarrow Y+B_2$$

addition

$$A_2+B_2 \rightarrow K$$

minimum

Examples of function computation

composition: $f(a,b) = \cancel{3a - b}$???

$3a-(b/2)$

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

only linear functions computable?

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

$$A+B \rightarrow Y$$

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

$A \rightarrow Y+A_2$ $B \rightarrow Y+B_2$	addition
$A_2+B_2 \rightarrow K$	minimum
$K+Y \rightarrow \emptyset$	subtraction

Examples of function computation

constant: $f(a) = 1$

Examples of function computation

constant: $f(a) = 1$

$$A \rightarrow Y$$
$$2Y \rightarrow Y$$

a.k.a. "leader election"

Examples of function computation

constant: $f(a) = 1$

$$A \rightarrow Y$$
$$2Y \rightarrow Y$$

a.k.a. "leader election"

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

Examples of function computation

constant: $f(a) = 1$

$$A \to Y$$
$$2Y \to Y$$

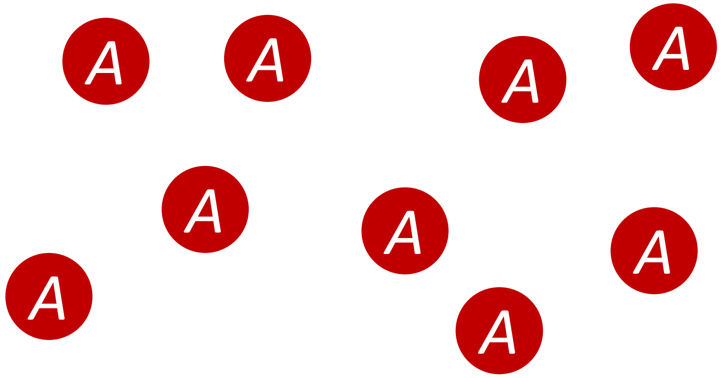
a.k.a. "leader election"

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

$$2A \to A + Y$$

Examples of predicate computation

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

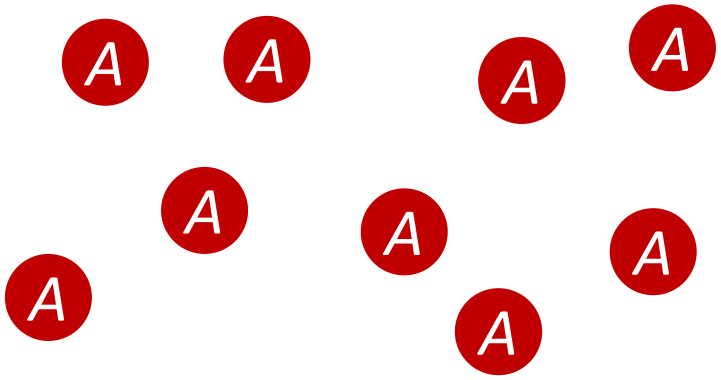


Examples of predicate computation

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

$$B+A \rightarrow 2B$$

A votes no; *B* votes yes

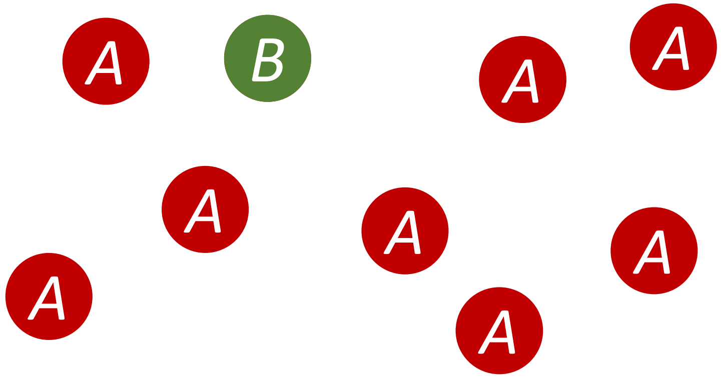


Examples of predicate computation

Detection: $\varphi(a,b)$ = yes ⇔ $b > 0$

$$B + A \rightarrow 2B$$

A votes no; *B* votes yes

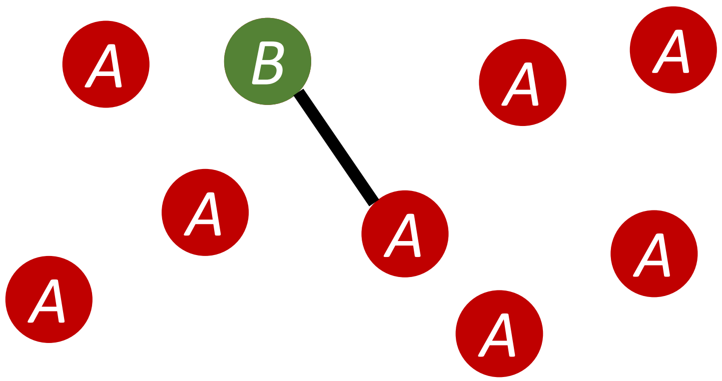


Examples of predicate computation

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

$$B + A \rightarrow 2B$$

A votes no; *B* votes yes

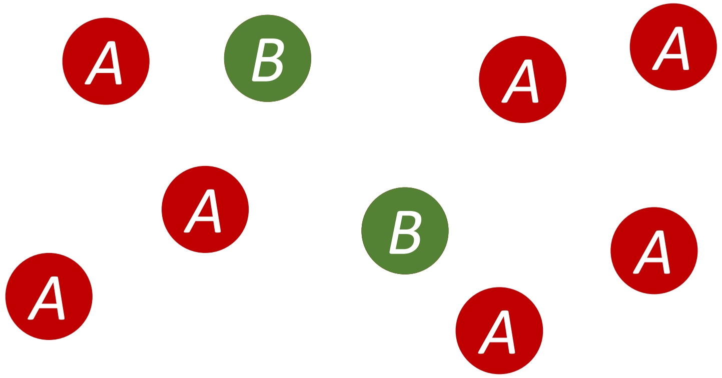


Examples of predicate computation

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

$$B + A \rightarrow 2B$$

A votes no; B votes yes

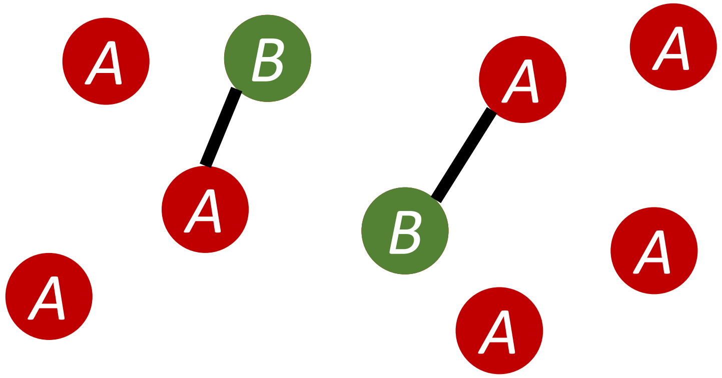


Examples of predicate computation

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

$$B + A \to 2B$$

A votes no; *B* votes yes

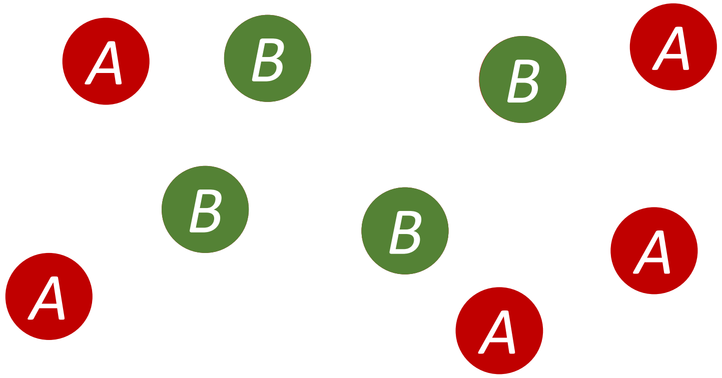


Examples of predicate computation

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

$$B + A \rightarrow 2B$$

A votes no; *B* votes yes

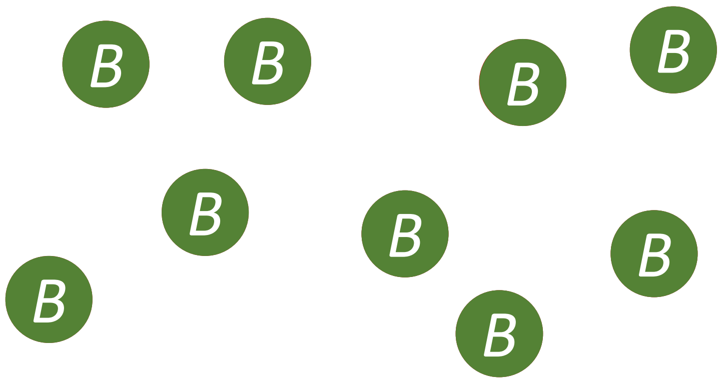


Examples of predicate computation

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

$$B + A \rightarrow 2B$$

A votes no; *B* votes yes

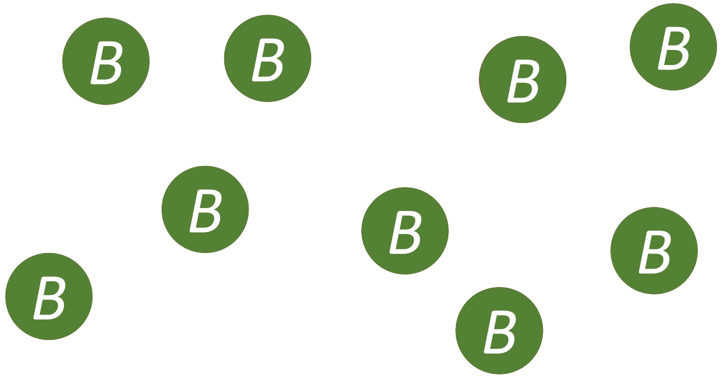


Examples of predicate computation

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

$$B + A \rightarrow 2B$$

A votes no; B votes yes



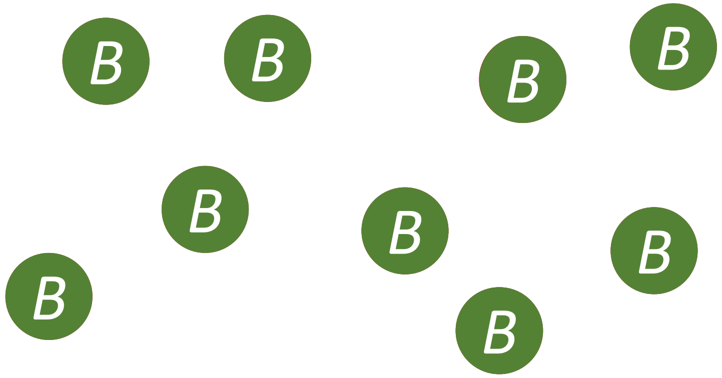
Counting: $\varphi(a,b)$ = yes $\Leftrightarrow$ $b > 1$

Examples of predicate computation

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

$$B + A \rightarrow 2B$$

A votes no; *B* votes yes



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

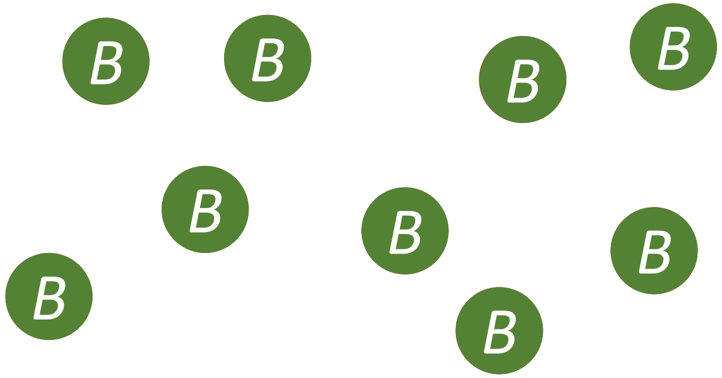
$$2B \rightarrow 2Y$$

Examples of predicate computation

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

$$B + A \rightarrow 2B$$

A votes no; B votes yes



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

$$2B \rightarrow 2Y$$

$$Y + B \rightarrow 2Y$$

$$Y + A \rightarrow 2Y$$

A,B vote no; Y votes yes

Examples of predicate computation

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

Examples of predicate computation

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

$A+B \rightarrow A_f+B_f$ (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: $\varphi(a,b)$ = yes $\Leftrightarrow a \geq b$

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

$A+B_f \rightarrow A+A_f$ (leader changes vote of follower)

$B+A_f \rightarrow B+B_f$ (leader changes vote of follower)

[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: $\varphi(a,b)$ = yes $\Leftrightarrow a \geq b$

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

$A+B_f \rightarrow A+A_f$ (leader changes vote of follower)

$B+A_f \rightarrow B+B_f$ (leader changes vote of follower)

$A_f+B_f \rightarrow A_f+A_f$ (tiebreaker if no leaders left when a=b)

[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 $\Leftrightarrow$ 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 <u>leader</u>)

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

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

Formal definition of CRN computation

Modeling choices in formalizing "*Computing with chemistry*"

Modeling choices in formalizing "*Computing with chemistry*"

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

Modeling choices in formalizing "*Computing with chemistry*"

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

we'll start with these choices

Modeling choices in formalizing "*Computing with chemistry*"

- integer counts ("*stochastic*") or real concentrations ("*mass-action*"/"*deterministic*")?
- what is the object being "computed"?
  - yes/no decision problem? *"#A's > #B's?"*
  - numerical function? *"set #Y = #X/2"*

we'll start with these choices

Modeling choices in formalizing "*Computing with chemistry*"

- integer counts ("*stochastic*") or real concentrations ("*mass-action*"/"*deterministic*")?
- what is the object being "computed"?
  - yes/no decision problem? *"#A's > #B's?"*
  - numerical function? *"set #Y = #X/2"*

we'll start with these choices

Modeling choices in formalizing "*Computing with chemistry*"

- integer counts ("*stochastic*") or real concentrations ("*mass-action*"/"*deterministic*")?
- what is the object being "computed"?
  - yes/no decision problem? *"#A's > #B's?"*
  - numerical function? *"set #Y = #X/2"*
- guaranteed to get correct answer? or allow small probability of error?
  - if Pr[error] = 0, system works *no matter the reaction rates*

we'll start with these choices

Modeling choices in formalizing "*Computing with chemistry*"

- integer counts ("*stochastic*") or real concentrations ("*mass-action*"/"*deterministic*")?
- what is the object being "computed"?
  - yes/no decision problem? *"#A's > #B's?"*
  - numerical function? *"set #Y = #X/2"*
- guaranteed to get correct answer? or allow small probability of error?
  - if Pr[error] = 0, system works *no matter the reaction rates*

we'll start with these choices

Modeling choices in formalizing "*Computing with chemistry*"

- integer counts ("*stochastic*") or real concentrations ("*mass-action*"/"*deterministic*")?
- what is the object being "computed"?
  - yes/no decision problem? *"#A's > #B's?"*
  - numerical function? *"set #Y = #X/2"*
- guaranteed to get correct answer? or allow small probability of error?
  - if Pr[error] = 0, system works *no matter the reaction rates*
- to represent input $a_1,\ldots,a_k$, what is the initial configuration?
  - only input species $A_1, \ldots, A_k$ present?
  - auxiliary species can be present?

we'll start with these choices

Modeling choices in formalizing "*Computing with chemistry*"

- integer counts ("*stochastic*") or real concentrations ("*mass-action*"/"*deterministic*")?
- what is the object being "computed"?
  - yes/no decision problem? *"#A's > #B's?"*
  - numerical function? *"set #Y = #X/2"*
- guaranteed to get correct answer? or allow small probability of error?
  - if Pr[error] = 0, system works *no matter the reaction rates*
- to represent input $a_1,\ldots,a_k$, what is the initial configuration?
  - only input species $A_1, \ldots, A_k$ present?
  - auxiliary species can be present?

we'll start with these choices

Modeling choices in formalizing "*Computing with chemistry*"

- integer counts ("*stochastic*") or real concentrations ("*mass-action*"/"*deterministic*")?
- what is the object being "computed"?
  - yes/no decision problem? *"#A's > #B's?"*
  - numerical function? *"set #Y = #X/2"*
- guaranteed to get correct answer? or allow small probability of error?
  - if Pr[error] = 0, system works *no matter the reaction rates*
- to represent input $a_1,\ldots,a_k$, what is the initial configuration?
  - only input species $A_1, \ldots, A_k$ present?
  - auxiliary species can be present?
- when is the computation finished? when…
  - the output stops changing? (convergence)
  - the output becomes unable to change? (stabilization)
  - a certain species *T* is first produced? (termination)

we'll start with these choices

Modeling choices in formalizing "*Computing with chemistry*"

- integer counts ("*stochastic*") or real concentrations ("*mass-action*"/"*deterministic*")?
- what is the object being "computed"?
  - yes/no decision problem? *"#A's > #B's?"*
  - numerical function? *"set #Y = #X/2"*
- guaranteed to get correct answer? or allow small probability of error?
  - if Pr[error] = 0, system works *no matter the reaction rates*
- to represent input $a_1,\ldots,a_k$, what is the initial configuration?
  - only input species $A_1, \ldots, A_k$ present?
  - auxiliary species can be present?
- when is the computation finished? when…
  - the output stops changing? (convergence)
  - the output becomes unable to change? (stabilization)
  - a certain species T is first produced? (termination)

we'll start with these choices

Modeling choices in formalizing "*Computing with chemistry*"

- integer counts ("*stochastic*") or real concentrations ("*mass-action*"/"*deterministic*")?
- what is the object being "computed"?
  - yes/no decision problem? *"#A's > #B's?"*
  - numerical function? *"set #Y = #X/2"*
- guaranteed to get correct answer? or allow small probability of error?
  - if Pr[error] = 0, system works *no matter the reaction rates*
- to represent input $a_1,\ldots,a_k$, what is the initial configuration?
  - only input species $A_1, \ldots, A_k$ present?
  - auxiliary species can be present?
- when is the computation finished? when…
  - the output stops changing? (convergence)
  - the output becomes unable to change? (stabilization)
  - a certain species T is first produced? (termination)
- require exact numerical answer? or allow an approximation?

we'll start with these choices

Modeling choices in formalizing "*Computing with chemistry*"

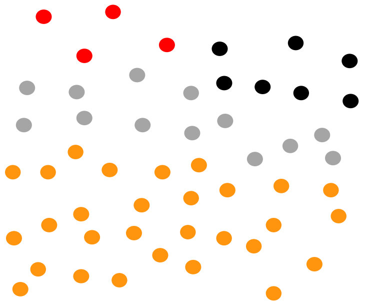
- integer counts ("*stochastic*") or real concentrations ("*mass-action*"/"*deterministic*")?
- what is the object being "computed"?
  - yes/no decision problem? *"#A's > #B's?"*
  - numerical function? *"set #Y = #X/2"*
- guaranteed to get correct answer? or allow small probability of error?
  - if Pr[error] = 0, system works *no matter the reaction rates*
- to represent input $a_1,\ldots,a_k$, what is the initial configuration?
  - only input species $A_1, \ldots, A_k$ present?
  - auxiliary species can be present?
- when is the computation finished? when…
  - the output stops changing? (convergence)
  - the output becomes unable to change? (stabilization)
  - a certain species *T* is first produced? (termination)
- require exact numerical answer? or allow an approximation?

we'll start with these choices

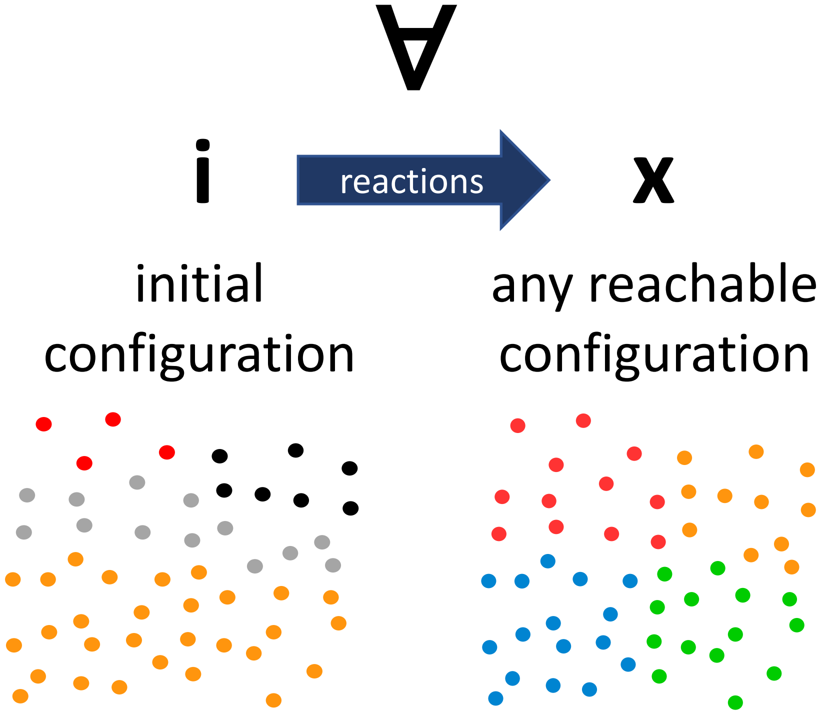
Defining **stable** computation

Defining **stable** computation

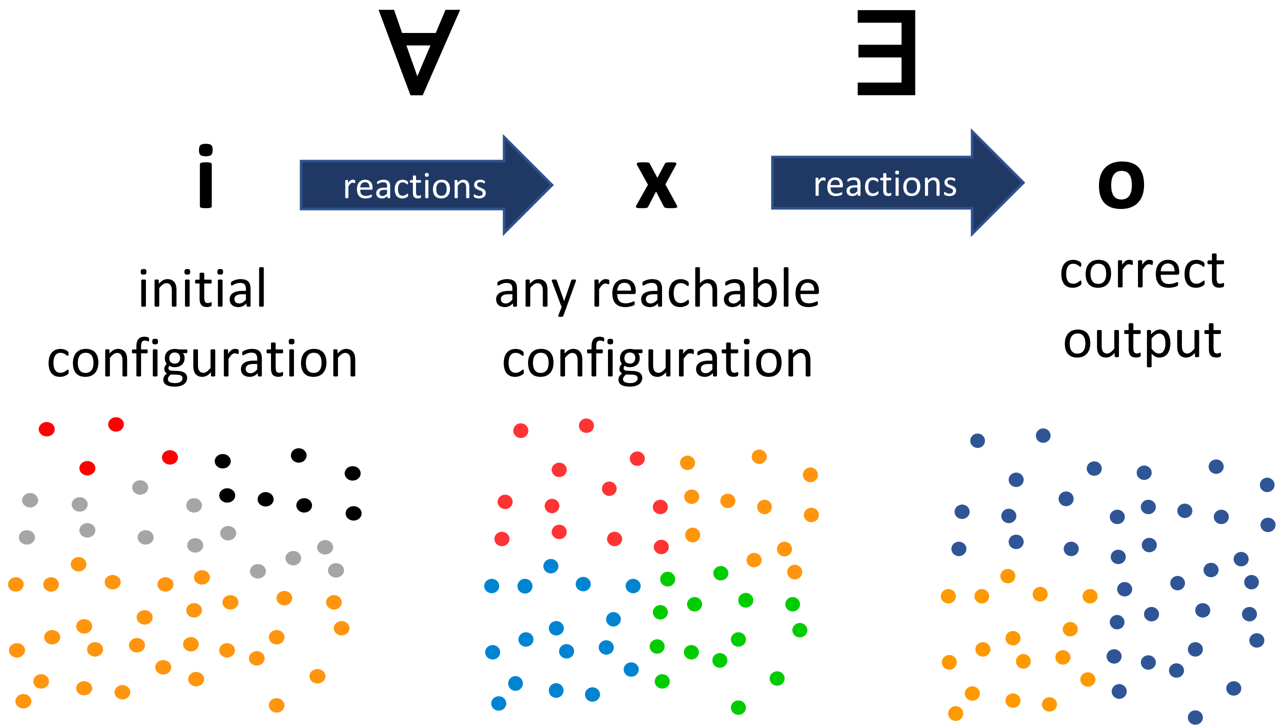
initial
configuration



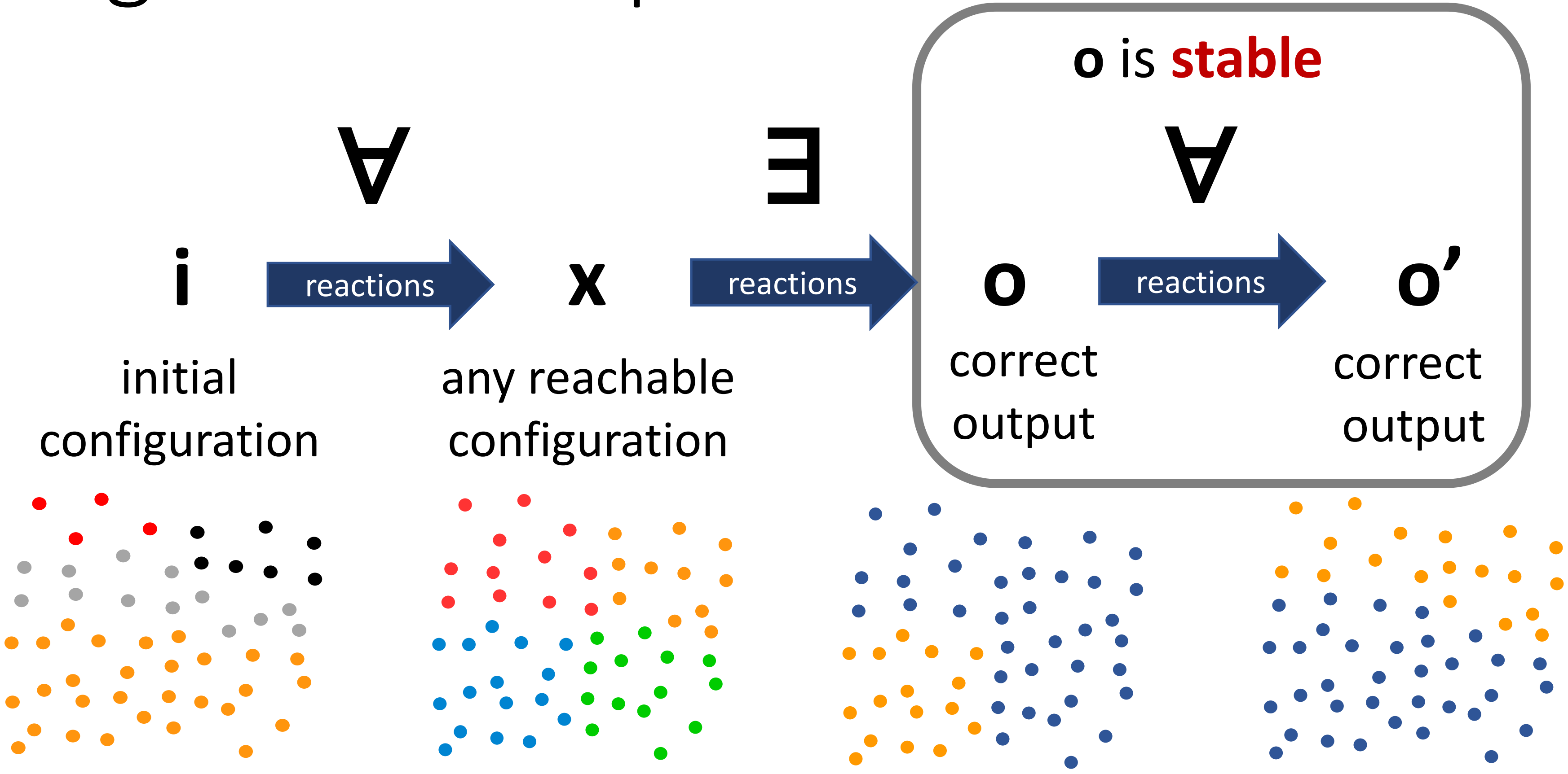
Defining **stable** computation



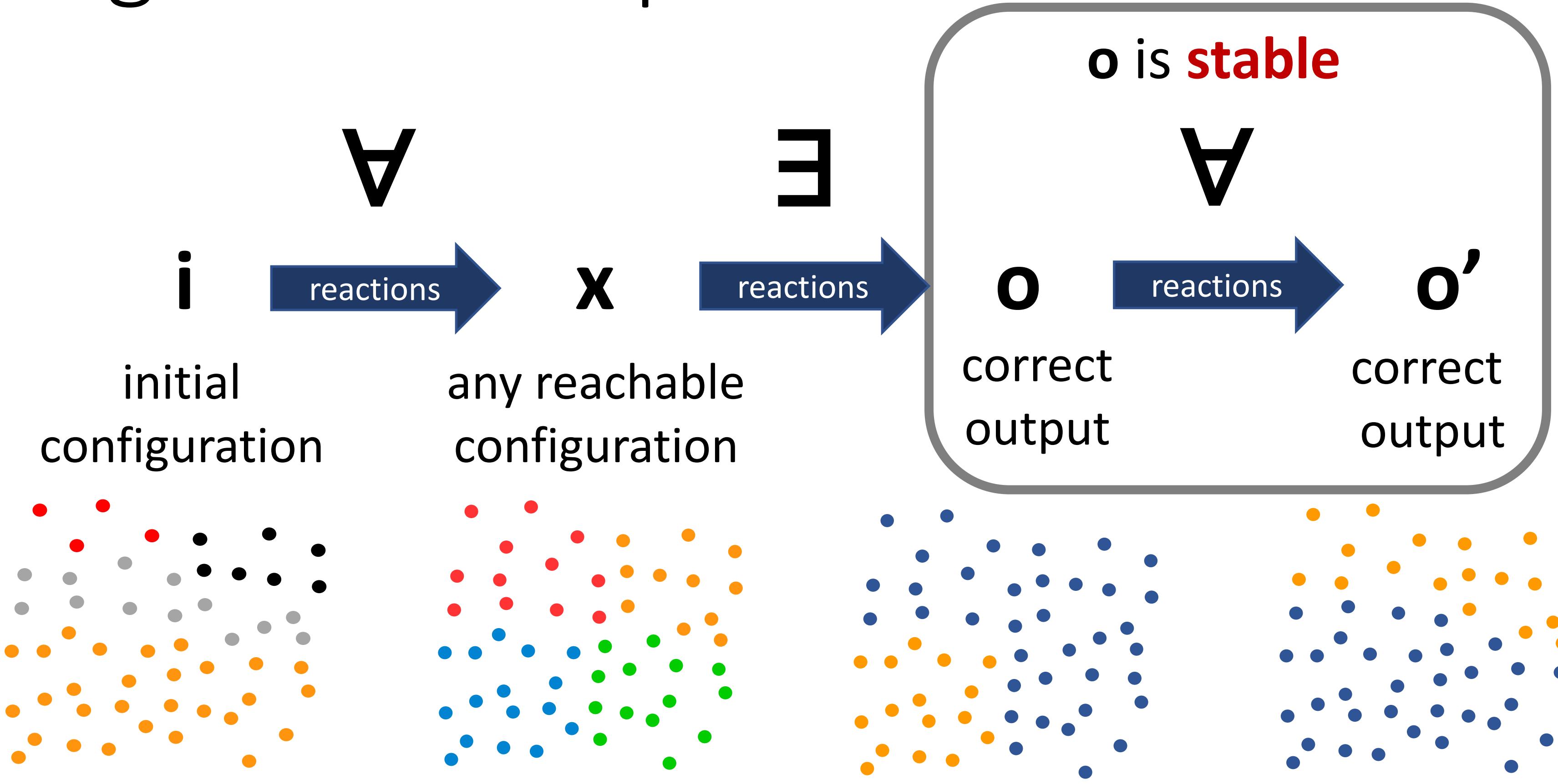
Defining **stable** computation



Defining **stable** computation



Defining **stable** computation



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

Probability-1 correctness can be characterized with only reachability

Probability-1 correctness can be characterized with only reachability

To understand this slide, only need the following fact: if a reaction is applicable, then there is a positive probability it occurs.

Probability-1 correctness can be characterized with only reachability

To understand this slide, only need the following fact: if a reaction is applicable, then there is a positive probability it occurs.

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

Probability-1 correctness can be characterized with only reachability

To understand this slide, only need the following fact: if a reaction is applicable, then there is a positive probability it occurs.

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

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

Probability-1 correctness can be characterized with only reachability

To understand this slide, only need the following fact: if a reaction is applicable, then there is a positive probability it occurs.

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

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

Theorem: Let $\mathbf{i}$ be a configuration where $\mathrm{Reach}(\mathbf{i})$ is finite, and let Y be a set of configurations. Then $(\Pr[\mathbf{i} \Longrightarrow Y] = 1) \Leftrightarrow (\forall \mathbf{x} \in \mathrm{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \mathrm{Reach}(\mathbf{x}))\ \mathbf{o} \in Y.$

Probability-1 correctness can be characterized with only reachability

To understand this slide, only need the following fact: if a reaction is applicable, then there is a positive probability it occurs.

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

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

Theorem: Let **i** be a configuration where Reach(**i**) is finite, and let Y be a set of configurations. Then $(\Pr[\mathbf{i} \Longrightarrow Y] = 1) \Leftrightarrow (\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.

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

Probability-1 correctness can be characterized with only reachability

To understand this slide, only need the following fact: if a reaction is applicable, then there is a positive probability it occurs.

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

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

Theorem: Let **i** be a configuration where Reach(**i**) is finite, and let Y be a set of configurations. Then (Pr[**i** $\Longrightarrow$ Y] = 1) $\Leftrightarrow$ ($\forall$**x**$\in$Reach(**i**)) ($\exists$**o**$\in$Reach(**x**)) **o** $\in Y$.

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

Proof:

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

Deterministic computation ≠ all executions correct

Deterministic computation ≠ all executions correct

False statement: If $\Pr[\mathbf{i} \Longrightarrow Y] = 1$, then every sufficiently long execution starting at $\mathbf{i}$ reaches to some $\mathbf{c} \in Y$.

Deterministic computation ≠ all executions correct

False statement: If Pr[**i** $\Longrightarrow$ *Y*] = 1, then every sufficiently long execution starting at **i** reaches to some **c** ∈ *Y*.

- Counterexample??

Deterministic computation ≠ all executions correct

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

- Counterexample??
- Suppose **i** = {*A*}, with reactions
  - $A \rightleftharpoons B$
  - $B \rightarrow C$

Deterministic computation ≠ all executions correct

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

- Counterexample??
- Suppose **i** = {*A*}, with reactions
  - A $\rightleftharpoons$ B
  - B $\rightarrow$ C
  - Then Pr[{*A*} $\Longrightarrow$ {*C*}] = 1, but the execution {A} $\Longrightarrow$ {B} $\Longrightarrow$ {A} $\Longrightarrow$ {B} $\Longrightarrow$ {A} $\Longrightarrow$ … avoids it forever.

Deterministic computation ≠ all executions correct

False statement: If Pr[**i** ⟹ *Y*] = 1, then every sufficiently long execution starting at **i** reaches to some **c** ∈ *Y*.

- Counterexample??
- Suppose **i** = {*A*}, with reactions
  - A ⇌ B
  - B → C
  - Then Pr[{*A*} ⟹ {*C*}] = 1, but the execution {A} ⟹ {B} ⟹ {A} ⟹ {B} ⟹ {A} ⟹ … avoids it forever.
- **Lesson**: it is too strict to require <u>all</u> sufficiently long executions to reach *Y*.

Fair executions: Alternative characterization of stable computation

Definition: An infinite execution $\mathbf{x}_0$, $\mathbf{x}_1$, … is fair if

Fair executions: Alternative characterization of stable computation

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

Theorem: Let $\mathbf{i}$ be a configuration, and let Y be a finite set of configurations. Then (every fair execution starting at $\mathbf{i}$ reaches some $\mathbf{o} \in Y$) $\Leftrightarrow (\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.

Definition: An infinite execution $\mathbf{x}_0, \mathbf{x}_1, \ldots$ is fair if

Fair executions: Alternative characterization of stable computation

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

Theorem: Let $\mathbf{i}$ be a configuration, and let Y be a finite set of configurations. Then (every fair execution starting at $\mathbf{i}$ reaches some $\mathbf{o} \in Y$) $\Leftrightarrow$ $(\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.

Definition: An infinite execution $\mathbf{x}_0, \mathbf{x}_1, \ldots$ is fair if $(\forall \mathbf{o} \in \mathbb{N}^{\Lambda})\ [(\exists^{\infty} i \in \mathbb{N}\ \mathbf{x}_i \Longrightarrow \mathbf{o})$ implies $(\exists^{\infty} k \in \mathbb{N}\ \mathbf{x}_k = \mathbf{o})]$

Fair executions: Alternative characterization of stable computation

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

Theorem: Let **i** be a configuration, and let Y be a finite set of configurations. Then
(every fair execution starting at **i** reaches some $\mathbf{o} \in Y$)
$\Leftrightarrow (\forall \mathbf{x} \in \mathrm{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \mathrm{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.

"there exist infinitely many"

Definition: An infinite execution $\mathbf{x}_0, \mathbf{x}_1, \ldots$ is fair if
$(\forall \mathbf{o} \in \mathbb{N}^\Lambda)\ [(\exists^\infty i \in \mathbb{N}\ \mathbf{x}_i \Longrightarrow \mathbf{o})$ implies $(\exists^\infty k \in \mathbb{N}\ \mathbf{x}_k = \mathbf{o})]$

Fair executions: Alternative characterization of stable computation

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

Theorem: Let **i** be a configuration, and let Y be a finite set of configurations. Then (every fair execution starting at **i** reaches some $\mathbf{o} \in Y$) $\Leftrightarrow$ $(\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.

"there exist infinitely many"

Definition: An infinite execution $\mathbf{x}_0, \mathbf{x}_1, \ldots$ is fair if $(\forall \mathbf{o} \in \mathbb{N}^\Lambda)\ [(\exists^\infty i \in \mathbb{N}\ \mathbf{x}_i \Longrightarrow \mathbf{o})$ implies $(\exists^\infty k \in \mathbb{N}\ \mathbf{x}_k = \mathbf{o})]$
(*every configuration infinitely often reachable is infinitely often reached*)

Fair executions: Alternative characterization of stable computation

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

Theorem: Let **i** be a configuration, and let Y be a finite set of configurations. Then
(every fair execution starting at **i** reaches some $\mathbf{o} \in Y$)
$\Leftrightarrow (\forall \mathbf{x} \in \mathrm{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \mathrm{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.

"there exist infinitely many"

Definition: An infinite execution $\mathbf{x}_0, \mathbf{x}_1, \ldots$ is fair if
$(\forall \mathbf{o} \in \mathbb{N}^\Lambda)\ [(\exists^\infty i \in \mathbb{N}\ \mathbf{x}_i \Longrightarrow \mathbf{o})$ implies $(\exists^\infty k \in \mathbb{N}\ \mathbf{x}_k = \mathbf{o})]$
(*every configuration infinitely often reachable is infinitely often reached*)

Proof:
1. (⟹): Suppose every fair execution from **i** reaches Y.

Fair executions: Alternative characterization of stable computation

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

Theorem: Let **i** be a configuration, and let Y be a finite set of configurations. Then (every fair execution starting at **i** reaches some $\mathbf{o} \in Y$) $\Leftrightarrow$ $(\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.

"there exist infinitely many"

Definition: An infinite execution $\mathbf{x}_0, \mathbf{x}_1, \ldots$ is fair if $(\forall \mathbf{o} \in \mathbb{N}^\Lambda)\ [(\exists^\infty i \in \mathbb{N}\ \mathbf{x}_i \Longrightarrow \mathbf{o})$ implies $(\exists^\infty k \in \mathbb{N}\ \mathbf{x}_k = \mathbf{o})]$ (*every configuration infinitely often reachable is infinitely often reached*)

Proof:

1. ($\Longrightarrow$): Suppose every fair execution from **i** reaches Y.
2. Any finite execution can be extended to be fair. (why??)

Fair executions: Alternative characterization of stable computation

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

Theorem: Let **i** be a configuration, and let Y be a finite set of configurations. Then
(every fair execution starting at **i** reaches some $\mathbf{o} \in Y$)
$\Leftrightarrow (\forall \mathbf{x} \in \mathrm{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \mathrm{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.

"there exist infinitely many"

Definition: An infinite execution $\mathbf{x}_0, \mathbf{x}_1, \ldots$ is fair if
$(\forall \mathbf{o} \in \mathbb{N}^\Lambda)\ [(\exists^\infty i \in \mathbb{N}\ \mathbf{x}_i \Longrightarrow \mathbf{o})$ implies $(\exists^\infty k \in \mathbb{N}\ \mathbf{x}_k = \mathbf{o})]$
(*every configuration infinitely often reachable is infinitely often reached*)

Proof:
1. $(\Longrightarrow)$: Suppose every fair execution from **i** reaches Y.
2. Any finite execution can be extended to be fair. (why??)
3. Thus $(\forall \mathbf{x} \in \mathrm{Reach}(\mathbf{i}))$, i.e., for all **x** reachable via some finite execution starting at **i**, $(\exists \mathbf{o} \in \mathrm{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$, Y is reachable from **x** by extending with a fair execution.

Fair executions: Alternative characterization of stable computation

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

Theorem: Let **i** be a configuration, and let Y be a finite set of configurations. Then (every fair execution starting at **i** reaches some $\mathbf{o} \in Y$) $\Leftrightarrow$ $(\forall \mathbf{x} \in \mathrm{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \mathrm{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.

"there exist infinitely many"

Definition: An infinite execution $\mathbf{x}_0, \mathbf{x}_1, \ldots$ is fair if $(\forall \mathbf{o} \in \mathbb{N}^\Lambda)\ [(\exists^\infty i \in \mathbb{N}\ \mathbf{x}_i \Longrightarrow \mathbf{o})$ implies $(\exists^\infty k \in \mathbb{N}\ \mathbf{x}_k = \mathbf{o})]$ (*every configuration infinitely often reachable is infinitely often reached*)

Proof:

1. ($\Longrightarrow$): Suppose every fair execution from **i** reaches Y.
2. Any finite execution can be extended to be fair. (why??)
3. Thus $(\forall \mathbf{x} \in \mathrm{Reach}(\mathbf{i}))$, i.e., for all **x** reachable via some finite execution starting at **i**, $(\exists \mathbf{o} \in \mathrm{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$, Y is reachable from **x** by extending with a fair execution.
4. ($\Longleftarrow$): Suppose $(\forall \mathbf{x} \in \mathrm{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \mathrm{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.

Fair executions: Alternative characterization of stable computation

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

Theorem: Let **i** be a configuration, and let Y be a finite set of configurations. Then
(every fair execution starting at **i** reaches some $\mathbf{o} \in Y$)
$\Leftrightarrow (\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.

"there exist infinitely many"

Definition: An infinite execution $\mathbf{x}_0, \mathbf{x}_1, \ldots$ is fair if
$(\forall \mathbf{o} \in \mathbb{N}^\Lambda)\ [(\exists^\infty i \in \mathbb{N}\ \mathbf{x}_i \Longrightarrow \mathbf{o}) \text{ implies } (\exists^\infty k \in \mathbb{N}\ \mathbf{x}_k = \mathbf{o})]$
(*every configuration infinitely often reachable is infinitely often reached*)

Proof:
1. ($\Longrightarrow$): Suppose every fair execution from **i** reaches Y.
2. Any finite execution can be extended to be fair. (why??)
3. Thus $(\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))$, i.e., for all **x** reachable via some finite execution starting at **i**, $(\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$, Y is reachable from **x** by extending with a fair execution.
4. ($\Longleftarrow$): Suppose $(\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.
5. Let $\mathbf{x}_0, \mathbf{x}_1, \ldots$ be a fair execution with $\mathbf{i}=\mathbf{x}_0$.

Fair executions: Alternative characterization of stable computation

Goal of definition of <u>fair</u> is to make this theorem true:

Theorem: Let **i** be a configuration, and let Y be a finite set of configurations. Then
(every fair execution starting at **i** reaches some $\mathbf{o} \in Y$)
$\Leftrightarrow (\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.

"there exist infinitely many"

Definition: An infinite execution $\mathbf{x}_0, \mathbf{x}_1, \ldots$ is <u>fair</u> if
$(\forall \mathbf{o} \in \mathbb{N}^{\Lambda})\ [(\exists^{\infty} i \in \mathbb{N}\ \mathbf{x}_i \Longrightarrow \mathbf{o})$ implies $(\exists^{\infty} k \in \mathbb{N}\ \mathbf{x}_k = \mathbf{o})]$
(*every configuration infinitely often reachable is infinitely often reached*)

Proof:
1. ($\Longrightarrow$): Suppose every fair execution from **i** reaches Y.
2. Any finite execution can be extended to be fair. (why??)
3. Thus $(\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))$, i.e., for all **x** reachable via some finite execution starting at **i**, $(\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$, Y is reachable from **x** by extending with a fair execution.
4. ($\Longleftarrow$): Suppose $(\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.
5. Let $\mathbf{x}_0, \mathbf{x}_1, \ldots$ be a fair execution with $\mathbf{i} = \mathbf{x}_0$.
6. Since all $\mathbf{x}_j \in \text{Reach}(\mathbf{i})$, for each j, by hypothesis $\exists \mathbf{o}_j \in \text{Reach}(\mathbf{x}_j)\ \mathbf{o}_j \in Y$.

Fair executions: Alternative characterization of stable computation

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

Theorem: Let **i** be a configuration, and let Y be a finite set of configurations. Then (every fair execution starting at **i** reaches some $\mathbf{o} \in Y$) $\Leftrightarrow$ $(\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.

"there exist infinitely many"

Definition: An infinite execution $\mathbf{x}_0, \mathbf{x}_1, \ldots$ is fair if $(\forall \mathbf{o} \in \mathbb{N}^\Lambda)\ [(\exists^\infty i \in \mathbb{N}\ \mathbf{x}_i \Rightarrow \mathbf{o})$ implies $(\exists^\infty k \in \mathbb{N}\ \mathbf{x}_k = \mathbf{o})]$
(*every configuration infinitely often reachable is infinitely often reached*)

Proof:

1. $(\Rightarrow)$: Suppose every fair execution from **i** reaches Y.
2. Any finite execution can be extended to be fair. (why??)
3. Thus $(\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))$, i.e., for all **x** reachable via some finite execution starting at **i**, $(\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$, Y is reachable from **x** by extending with a fair execution.
4. $(\Leftarrow)$: Suppose $(\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.
5. Let $\mathbf{x}_0, \mathbf{x}_1, \ldots$ be a fair execution with $\mathbf{i} = \mathbf{x}_0$.
6. Since all $\mathbf{x}_j \in \text{Reach}(\mathbf{i})$, for each j, by hypothesis $\exists \mathbf{o}_j \in \text{Reach}(\mathbf{x}_j)\ \mathbf{o}_j \in Y$.
7. Since Y is finite, some $\mathbf{o} \in Y$ is reachable from infinitely many $\mathbf{x}_j$.

Fair executions: Alternative characterization of stable computation

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

Theorem: Let **i** be a configuration, and let Y be a finite set of configurations. Then
(every fair execution starting at **i** reaches some $\mathbf{o} \in Y$) $\Leftrightarrow$ $(\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.

"there exist infinitely many"

Definition: An infinite execution $\mathbf{x}_0, \mathbf{x}_1, \ldots$ is fair if
$(\forall \mathbf{o} \in \mathbb{N}^\Lambda)\ [(\exists^\infty i \in \mathbb{N}\ \mathbf{x}_i \Longrightarrow \mathbf{o})$ implies $(\exists^\infty k \in \mathbb{N}\ \mathbf{x}_k = \mathbf{o})]$
(*every configuration infinitely often reachable is infinitely often reached*)

Proof:

1. ($\Longrightarrow$): Suppose every fair execution from **i** reaches Y.
2. Any finite execution can be extended to be fair. (why??)
3. Thus $(\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))$, i.e., for all **x** reachable via some finite execution starting at **i**, $(\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$, Y is reachable from **x** by extending with a fair execution.
4. ($\Longleftarrow$): Suppose $(\forall \mathbf{x} \in \text{Reach}(\mathbf{i}))\ (\exists \mathbf{o} \in \text{Reach}(\mathbf{x}))\ \mathbf{o} \in Y$.
5. Let $\mathbf{x}_0, \mathbf{x}_1, \ldots$ be a fair execution with $\mathbf{i} = \mathbf{x}_0$.
6. Since all $\mathbf{x}_j \in \text{Reach}(\mathbf{i})$, for each j, by hypothesis $\exists \mathbf{o}_j \in \text{Reach}(\mathbf{x}_j)\ \mathbf{o}_j \in Y$.
7. Since Y is finite, some $\mathbf{o} \in Y$ is reachable from infinitely many $\mathbf{x}_j$.
8. Since $\mathbf{x}_0, \mathbf{x}_1, \ldots$ is fair and **o** is infinitely often reachable, there is k such that $\mathbf{x}_k = \mathbf{o} \in Y$, i.e., the fair execution reaches Y. **QED**

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: $\mathbb{N}^k \rightarrow \mathbb{N}$, e.g., $f(a,b) = 2a + b/2$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - valid initial configuration **i**: all molecules are from 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 **i**: 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: $\mathbb{N}^k \rightarrow \mathbb{N}$, e.g., $f(a,b) = 2a + b/2$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - valid initial configuration **i**: all molecules are from Σ, e.g., $\{100\ a, 100\ b\}$
- output specification: designate one species $Y \in \Lambda$ whose count is the *output*
- **o** is stable if, for all **o'** reachable from **o**, $\mathbf{o}(Y) = \mathbf{o'}(Y)$

Definition of function computation

- goal: compute function f: $\mathbb{N}^k \to \mathbb{N}$, e.g., $f(a,b) = 2a + b/2$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - valid initial configuration **i**: all molecules are from Σ, e.g., $\{100\ a,\ 100\ b\}$
- output specification: designate one species $Y \in \Lambda$ whose count is the *output*
- **o** is stable if, for all **o'** reachable from **o**, **o**(Y) = **o'**(Y)
- CRN stably computes f if, for all valid initial configurations **i**, and all **x** reachable from **i**, there is a stable **o** reachable from **x** such that **o**(Y) = f(**i**).

Definition of function computation

- goal: compute function $f: \mathbb{N}^k \rightarrow \mathbb{N}$, e.g., $f(a,b) = 2a + b/2$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - valid initial configuration **i**: all molecules are from Σ, e.g., {100 *a*, 100 *b*}
- output specification: designate one species $Y \in \Lambda$ whose count is the *output*
- **o** is stable if, for all **o'** reachable from **o**, **o**(Y) = **o'**(Y)
- CRN stably computes *f* if, for all valid initial configurations **i**, and all **x** reachable from **i**, there is a stable **o** reachable from **x** such that **o**(Y) = f(**i**).
  - **Recall**: this is equivalent to saying that **i** reaches to a correct, stable **o** with probability 1, and equivalent to saying that every fair execution from **i** reaches to a correct, stable **o**.

Definition of *predicate* (decision problem) computation

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

[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: $\mathbb{N}^k \rightarrow$ {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 $\mathbf{i}$ 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: $\mathbb{N}^k \rightarrow \{Y,N\}$, e.g., $\varphi(a,b)=Y \iff a>b$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - in valid initial configurations **i** 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 **i** 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: $\mathbf{o}(S)>0 \implies S \in \Lambda_Y$
  - $\psi(\mathbf{o})$ = N (configuration **o** outputs "no") if vote is unanimously no: $\mathbf{o}(S)>0 \implies S \in \Lambda_N$
  - $\psi(\mathbf{o})$ *undefined* 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: $\mathbb{N}^k \rightarrow \{Y,N\}$, e.g., $\varphi(a,b)=Y \Leftrightarrow a>b$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - in valid initial configurations **i** 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: $\mathbf{o}(S)>0 \Longrightarrow S \in \Lambda_Y$
  - $\psi(\mathbf{o}) = N$ (configuration **o** outputs "no") if vote is unanimously no: $\mathbf{o}(S)>0 \Longrightarrow S \in \Lambda_N$
  - $\psi(\mathbf{o})$ *undefined* otherwise: $(\exists\ S \in \Lambda_N, S' \in \Lambda_Y)$ $\mathbf{o}(S)>0$ and $\mathbf{o}(S')>0$
- **o** is stable if $\psi(\mathbf{o}) = \psi(\mathbf{o'})$ (and is defined) for all **o'** reachable from **o**

[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: $\mathbb{N}^k \rightarrow \{Y,N\}$, e.g., $\varphi(a,b)=Y \iff a>b$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - in valid initial configurations **i** 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: $\mathbf{o}(S)>0 \Longrightarrow S \in \Lambda_Y$
  - $\psi(\mathbf{o})$ = N (configuration **o** outputs "no") if vote is unanimously no: $\mathbf{o}(S)>0 \Longrightarrow S \in \Lambda_N$
  - $\psi(\mathbf{o})$ *undefined* otherwise: $(\exists\ S \in \Lambda_N, S' \in \Lambda_Y)\ \mathbf{o}(S)>0$ and $\mathbf{o}(S')>0$
- **o** is stable if $\psi(\mathbf{o}) = \psi(\mathbf{o'})$ (and is defined) for all **o'** reachable from **o**
- CRN stably computes φ if, for all valid initial configurations **i**, and all **x** reachable from **i**, there is a stable **o** reachable from **x** such that $\psi(\mathbf{o}) = \varphi(\mathbf{i})$ (***o** is correct*).

[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 $\varphi: \mathbb{N}^k \to \{Y,N\}$, e.g., $\varphi(a,b)=Y \iff a>b$
- input specification: designate subset $\Sigma \subseteq \Lambda$ as "input" species
  - in valid initial configurations **i** 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: $\mathbf{o}(S)>0 \implies S\in\Lambda_Y$
  - $\psi(\mathbf{o})$ = N (configuration **o** outputs "no") if vote is unanimously no: $\mathbf{o}(S)>0 \implies S\in\Lambda_N$
  - $\psi(\mathbf{o})$ *undefined* otherwise: $(\exists\ S\in\Lambda_N, S'\in\Lambda_Y)\ \mathbf{o}(S)>0$ and $\mathbf{o}(S')>0$
- **o** is stable if $\psi(\mathbf{o}) = \psi(\mathbf{o'})$ (and is defined) for all **o'** reachable from **o**
- CRN stably computes φ if, for all valid initial configurations **i**, and all **x** reachable from **i**, there is a stable **o** reachable from **x** such that $\psi(\mathbf{o}) = \varphi(\mathbf{i})$ (***o*** *is correct*).
  - We say the CRN stably decides the set $\varphi^{-1}(Y)$ = set of inputs mapping to output Y

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

Feedforward CRNs

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

Stable versus terminal

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

Stable versus terminal

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

Definition: A configuration is <u>terminal</u> if no reaction is applicable to it.

Stable versus terminal

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

Definition: A configuration is <u>terminal</u> if no reaction is applicable to it.

Observation: Every terminal configuration is stable.

Stable versus terminal

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

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

Observation: Every terminal configuration is stable.

Note: A configuration can be *stable without being terminal*. Example?

Feed-forward CRNs

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

Ideas taken from [M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Feed-forward CRNs

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

Example: The max(A,B) CRN:

1. $A \to Y + A_2$ (A doesn't appear below)
2. $B \to Y + B_2$ (B doesn't appear below)
3. $A_2 + B_2 \to K$ (A_2,B_2 don't appear below)
4. $K + Y \to \emptyset$

Ideas taken from [M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Feed-forward CRNs

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

Example: The max(A,B) CRN:

1. $A \rightarrow Y+A_2$ (A doesn't appear below)
2. $B \rightarrow Y+B_2$ (B doesn't appear below)
3. $A_2+B_2 \rightarrow K$ (A_2,B_2 don't appear below)
4. $K+Y \rightarrow \emptyset$

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

Ideas taken from [M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Feed-forward CRNs

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

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

Example: The max(A,B) CRN:

1. $A \rightarrow Y+A_2$ (A doesn't appear below)
2. $B \rightarrow Y+B_2$ (B doesn't appear below)
3. $A_2+B_2 \rightarrow K$ (A_2,B_2 don't appear below)
4. $K+Y \rightarrow \emptyset$

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

Ideas taken from [M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Feed-forward CRNs

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

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

Example: The max(A,B) CRN:

1. $A \rightarrow Y+A_2$ (A doesn't appear below)
2. $B \rightarrow Y+B_2$ (B doesn't appear below)
3. $A_2+B_2 \rightarrow K$ (A_2,B_2 don't appear below)
4. $K+Y \rightarrow \emptyset$

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

$\#(r_k,P)$ = number of times r_k occurs in P

Proof:

1. Let r_k be <u>first</u> reaction in feed-forward order such that $\#(r_k,P) < \#(r_k,Q)$.

Ideas taken from [M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Feed-forward CRNs

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

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

Example: The max(A,B) CRN:

1. $A \to Y + A_2$ (A doesn't appear below)
2. $B \to Y + B_2$ (B doesn't appear below)
3. $A_2 + B_2 \to K$ (A_2,B_2 don't appear below)
4. $K + Y \to \emptyset$

#(r_k,P) = number of times r_k occurs in P

Proof:

1. Let r_k be <u>first</u> reaction in feed-forward order such that $\#(r_k,P) < \#(r_k,Q)$.
2. For ease of exposition, assume r_k has only one reactant A.

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

Ideas taken from [M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Feed-forward CRNs

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

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

Example: The max(A,B) CRN:

1. $A \rightarrow Y+A_2$ (A doesn't appear below)
2. $B \rightarrow Y+B_2$ (B doesn't appear below)
3. $A_2+B_2 \rightarrow K$ (A_2,B_2 don't appear below)
4. $K+Y \rightarrow \emptyset$

#(r_k,P) = number of times r_k occurs in P

Proof:

1. Let r_k be <u>first</u> reaction in feed-forward order such that $\#(r_k,P) < \#(r_k,Q)$.
2. For ease of exposition, assume r_k has only one reactant A.
3. $r_{k+1} \ldots r_n$ do not change $\#A$, by the definition of feed-forward.

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

Ideas taken from [M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Feed-forward CRNs

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

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

Example: The max(A,B) CRN:

1. $A \rightarrow Y + A_2$ (A doesn't appear below)
2. $B \rightarrow Y + B_2$ (B doesn't appear below)
3. $A_2 + B_2 \rightarrow K$ (A_2,B_2 don't appear below)
4. $K + Y \rightarrow \emptyset$

#(r_k,P) = number of times r_k occurs in P

Proof:

1. Let r_k be first reaction in feed-forward order such that $\#(r_k,P) < \#(r_k,Q)$.
2. For ease of exposition, assume r_k has only one reactant A.
3. *3.* $r_{k+1} \ldots r_n$ do not change $\#A$, by the definition of feed-forward.
4. *4.* $r_1 \ldots r_{k-1}$ can produce but not consume A. (*why??*)

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

Ideas taken from [M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Feed-forward CRNs

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

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

Example: The max(A,B) CRN:

1. $A \rightarrow Y + A_2$ (A doesn't appear below)
2. $B \rightarrow Y + B_2$ (B doesn't appear below)
3. $A_2 + B_2 \rightarrow K$ (A_2,B_2 don't appear below)
4. $K + Y \rightarrow \emptyset$

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

$\#(r_k,P)$ = number of times r_k occurs in P

Proof:

1. Let r_k be first reaction in feed-forward order such that $\#(r_k,P) < \#(r_k,Q)$.
2. For ease of exposition, assume r_k has only one reactant A.
3. *3.* $r_{k+1} \ldots r_n$ do not change $\#A$, by the definition of feed-forward.
4. *4.* $r_1 \ldots r_{k-1}$ can produce but not consume A. (*why??*)
5. So only $r_1 \ldots r_k$ can increase $\#A$, and only r_k can decrease $\#A$.

Ideas taken from [M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Feed-forward CRNs

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

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

Example: The max(A,B) CRN:

1. $A \rightarrow Y + A_2$ (A doesn't appear below)
2. $B \rightarrow Y + B_2$ (B doesn't appear below)
3. $A_2 + B_2 \rightarrow K$ (A_2,B_2 don't appear below)
4. $K + Y \rightarrow \emptyset$

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

$\#(r_k,P)$ = number of times r_k occurs in P

Proof:

1. Let r_k be first reaction in feed-forward order such that $\#(r_k,P) < \#(r_k,Q)$.
2. For ease of exposition, assume r_k has only one reactant A.
3. $r_{k+1} \ldots r_n$ do not change $\#A$, by the definition of feed-forward.
4. $r_1 \ldots r_{k-1}$ can produce but not consume A. (*why??*)
5. So only $r_1 \ldots r_k$ can increase $\#A$, and only r_k can decrease $\#A$.
6. Let $m = \#(r_k,P)$; Let Q' be prefix $(\mathbf{i}, \mathbf{x}_1, \ldots, \mathbf{x}_p)$ of Q such that $\mathbf{x}_p \Longrightarrow \mathbf{x}_{p+1}$ by the (m+1)'st execution of reaction r_k.

Ideas taken from [M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Feed-forward CRNs

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

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

Example: The max(A,B) CRN:

1. $A \rightarrow Y+A_2$ (A doesn't appear below)
2. $B \rightarrow Y+B_2$ (B doesn't appear below)
3. $A_2+B_2 \rightarrow K$ (A_2,B_2 don't appear below)
4. $K+Y \rightarrow \emptyset$

#(r_k,P) = number of times r_k occurs in P

Proof:

1. Let r_k be <u>first</u> reaction in feed-forward order such that $\#(r_k,P) < \#(r_k,Q)$.
2. For ease of exposition, assume r_k has only one reactant A.
3. $r_{k+1} \ldots r_n$ do not change $\#A$, by the definition of feed-forward.
4. $r_1 \ldots r_{k-1}$ can produce but not consume A. (*why??*)
5. So only $r_1 \ldots r_k$ can increase $\#A$, and only r_k can decrease $\#A$.
6. Let $m = \#(r_k,P)$; Let Q' be prefix ($\mathbf{i}$, $\mathbf{x}_1$, …, $\mathbf{x}_p$) of Q such that $\mathbf{x}_p \Longrightarrow \mathbf{x}_{p+1}$ by the (m+1)'st execution of reaction r_k.
   - $\mathbf{x}_p$ is the config <u>just before</u> the first time that r_k happens more in Q than P.

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

Ideas taken from [M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Feed-forward CRNs

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

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

Example: The max(A,B) CRN:

1. $A \rightarrow Y + A_2$ (A doesn't appear below)
2. $B \rightarrow Y + B_2$ (B doesn't appear below)
3. $A_2 + B_2 \rightarrow K$ (A_2,B_2 don't appear below)
4. $K + Y \rightarrow \emptyset$

#(r_k,P) = number of times r_k occurs in P

Proof:

1. Let r_k be <u>first</u> reaction in feed-forward order such that $\#(r_k,P) < \#(r_k,Q)$.
2. For ease of exposition, assume r_k has only one reactant A.
3. $r_{k+1} \ldots r_n$ do not change $\#A$, by the definition of feed-forward.
4. $r_1 \ldots r_{k-1}$ can produce but not consume A. (*why??*)
5. So only $r_1 \ldots r_k$ can increase $\#A$, and only r_k can decrease $\#A$.
6. Let $m = \#(r_k,P)$; Let Q' be prefix $(\mathbf{i}, \mathbf{x}_1, \ldots, \mathbf{x}_p)$ of Q such that $\mathbf{x}_p \Longrightarrow \mathbf{x}_{p+1}$ by the $(m+1)$'st execution of reaction r_k.
   - $\mathbf{x}_p$ is the config <u>just before</u> the first time that r_k happens more in Q than P.
7. Note $r_1 \ldots r_{k-1}$ occur least as much in P as in Q. ($\#(r_i,P) \geq \#(r_i,Q)$ for i=1 to k–1)

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

Ideas taken from [M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Feed-forward CRNs

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

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

Example: The max(A,B) CRN:

1. $A \to Y + A_2$ (A doesn't appear below)
2. $B \to Y + B_2$ (B doesn't appear below)
3. $A_2 + B_2 \to K$ (A_2,B_2 don't appear below)
4. $K + Y \to \emptyset$

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

$\#(r_k,P)$ = number of times r_k occurs in P

Proof:

1. Let r_k be <u>first</u> reaction in feed-forward order such that $\#(r_k,P) < \#(r_k,Q)$.
2. For ease of exposition, assume r_k has only one reactant A.
3. $r_{k+1} \ldots r_n$ do not change $\#A$, by the definition of feed-forward.
4. $r_1 \ldots r_{k-1}$ can produce but not consume A. (*why??*)
5. So only $r_1 \ldots r_k$ can increase $\#A$, and only r_k can decrease $\#A$.
6. Let $m = \#(r_k,P)$; Let Q' be prefix $(\mathbf{i}, \mathbf{x}_1, \ldots, \mathbf{x}_p)$ of Q such that $\mathbf{x}_p \Longrightarrow \mathbf{x}_{p+1}$ by the $(m+1)$'st execution of reaction r_k.
   - $\mathbf{x}_p$ is the config <u>just before</u> the first time that r_k happens more in Q than P.
7. Note $r_1 \ldots r_{k-1}$ occur least as much in P as in Q. ($\#(r_i,P) \geq \#(r_i,Q)$ for i=1 to k–1)
8. Thus $r_1 \ldots r_{k-1}$ occur least as much in P as in Q'. (since Q' is prefix of Q)

Ideas taken from [M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Feed-forward CRNs

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

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

Example: The max(A,B) CRN:

1. $A \rightarrow Y+A_2$ (A doesn't appear below)
2. $B \rightarrow Y+B_2$ (B doesn't appear below)
3. $A_2+B_2 \rightarrow K$ (A_2,B_2 don't appear below)
4. $K+Y \rightarrow \emptyset$

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

#(r_k,P) = number of times r_k occurs in P

Proof:

1. Let r_k be <u>first</u> reaction in feed-forward order such that $\#(r_k,P) < \#(r_k,Q)$.
2. For ease of exposition, assume r_k has only one reactant A.
3. $r_{k+1} \ldots r_n$ do not change $\#A$, by the definition of feed-forward.
4. $r_1 \ldots r_{k-1}$ can produce but not consume A. (*why??*)
5. So only $r_1 \ldots r_k$ can increase $\#A$, and only r_k can decrease $\#A$.
6. Let $m = \#(r_k,P)$; Let Q' be prefix $(\mathbf{i}, \mathbf{x}_1, \ldots, \mathbf{x}_p)$ of Q such that $\mathbf{x}_p \Longrightarrow \mathbf{x}_{p+1}$ by the $(m+1)$'st execution of reaction r_k.
   - $\mathbf{x}_p$ is the config <u>just before</u> the first time that r_k happens more in Q than P.
7. Note $r_1 \ldots r_{k-1}$ occur least as much in P as in Q. ($\#(r_i,P) \geq \#(r_i,Q)$ for i=1 to $k-1$)
8. Thus $r_1 \ldots r_{k-1}$ occur least as much in P as in Q'. (since Q' is prefix of Q)
9. Also, $\#(r_k,P) = \#(r_k,Q')$ by our choice of Q'.

Ideas taken from [M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Feed-forward CRNs

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

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

Example: The max(A,B) CRN:

1. $A \rightarrow Y+A_2$ (A doesn't appear below)
2. $B \rightarrow Y+B_2$ (B doesn't appear below)
3. $A_2+B_2 \rightarrow K$ (A_2,B_2 don't appear below)
4. $K+Y \rightarrow \emptyset$

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

$\#(r_k,P)$ = number of times r_k occurs in P

Proof:

1. Let r_k be first reaction in feed-forward order such that $\#(r_k,P) < \#(r_k,Q)$.
2. For ease of exposition, assume r_k has only one reactant A.
3. $r_{k+1} \ldots r_n$ do not change $\#A$, by the definition of feed-forward.
4. $r_1 \ldots r_{k-1}$ can produce but not consume A. (*why??*)
5. So only $r_1 \ldots r_k$ can increase $\#A$, and only r_k can decrease $\#A$.
6. Let $m = \#(r_k,P)$; Let Q' be prefix $(\mathbf{i}, \mathbf{x}_1, \ldots, \mathbf{x}_p)$ of Q such that $\mathbf{x}_p \Longrightarrow \mathbf{x}_{p+1}$ by the $(m+1)$'st execution of reaction r_k.
   - $\mathbf{x}_p$ is the config just before the first time that r_k happens more in Q than P.
7. Note $r_1 \ldots r_{k-1}$ occur least as much in P as in Q. ($\#(r_i,P) \geq \#(r_i,Q)$ for i=1 to $k-1$)
8. Thus $r_1 \ldots r_{k-1}$ occur least as much in P as in Q'. (since Q' is prefix of Q)
9. Also, $\#(r_k,P) = \#(r_k,Q')$ by our choice of Q'.
10. So A is present in $\mathbf{c}$, i.e., $\mathbf{c}(A) > 0$.

Ideas taken from [M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Feed-forward CRNs

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

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

Example: The max(A,B) CRN:

1. $A \rightarrow Y+A_2$ (A doesn't appear below)
2. $B \rightarrow Y+B_2$ (B doesn't appear below)
3. $A_2+B_2 \rightarrow K$ (A_2,B_2 don't appear below)
4. $K+Y \rightarrow \emptyset$

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

$\#(r_k,P)$ = number of times r_k occurs in P

Proof:

1. Let r_k be <u>first</u> reaction in feed-forward order such that $\#(r_k,P) < \#(r_k,Q)$.
2. For ease of exposition, assume r_k has only one reactant A.
3. $r_{k+1} \ldots r_n$ do not change $\#A$, by the definition of feed-forward.
4. $r_1 \ldots r_{k-1}$ can produce but not consume A. (*why??*)
5. So only $r_1 \ldots r_k$ can increase $\#A$, and only r_k can decrease $\#A$.
6. Let $m = \#(r_k,P)$; Let Q' be prefix $(\mathbf{i}, \mathbf{x}_1, \ldots, \mathbf{x}_p)$ of Q such that $\mathbf{x}_p \Longrightarrow \mathbf{x}_{p+1}$ by the $(m+1)$'st execution of reaction r_k.
   - $\mathbf{x}_p$ is the config <u>just before</u> the first time that r_k happens more in Q than P.
7. Note $r_1 \ldots r_{k-1}$ occur least as much in P as in Q. ($\#(r_i,P) \geq \#(r_i,Q)$ for i=1 to $k-1$)
8. Thus $r_1 \ldots r_{k-1}$ occur least as much in P as in Q'. (since Q' is prefix of Q)
9. Also, $\#(r_k,P) = \#(r_k,Q')$ by our choice of Q'.
10. So A is present in $\mathbf{c}$, i.e., $\mathbf{c}(A) > 0$.
11. Thus r_k is applicable at $\mathbf{c}$, so $\mathbf{c}$ is not terminal. **QED**

Ideas taken from [M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Stable function computation by feed-forward CRNs

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

Stable function computation by feed-forward CRNs

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

Corollary 1: A feed-forward CRN has <u>at most one</u> terminal configuration reachable from any initial configuration.

Stable function computation by feed-forward CRNs

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

Corollary 1: A feed-forward CRN has <u>at most one</u> terminal configuration reachable from any initial configuration.

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

Stable function computation by feed-forward CRNs

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

Corollary 1: A feed-forward CRN has <u>at most one</u> terminal configuration reachable from any initial configuration.

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

Question: What's the function?

Stable function computation by feed-forward CRNs

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

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

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

Question: What's the function?

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

Stable function computation by feed-forward CRNs

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

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

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

Question: What's the function?

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

Corollary: The CRN:

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

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

Stable function computation by feed-forward CRNs

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

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

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

Question: What's the function?

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

Corollary: The CRN:

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

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

Proof:

1. Do the following reactions:
   1. #*A* times rxn 1
   2. #*B* times rxn 2
   3. min(#*A*,#*B*) times rxn 3
   4. min(#*A*,#*B*) times rxn 4

Stable function computation by feed-forward CRNs

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

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

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

Question: What's the function?

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

Corollary: The CRN:

1. $A \to Y + A_2$
2. $B \to Y + B_2$
3. $A_2 + B_2 \to K$
4. $K + Y \to \emptyset$

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

Proof:

1. Do the following reactions:
    1. $\#A$ times rxn 1
    2. $\#B$ times rxn 2
    3. $\min(\#A,\#B)$ times rxn 3
    4. $\min(\#A,\#B)$ times rxn 4
2. This removes all A, B, (at least one of A_2 or B_2), and K, so this is terminal. By Corollary 2 it stably computes whatever $\#Y$ is now, which is…

Stable function computation by feed-forward CRNs

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

Corollary 1: A feed-forward CRN has <u>at most one</u> terminal configuration reachable from any initial configuration.

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

Question: What's the function?

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

Corollary: The CRN:

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

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

Proof:

1. Do the following reactions:
    1. #*A* times rxn 1
    2. #*B* times rxn 2
    3. min(#*A*,#*B*) times rxn 3
    4. min(#*A*,#*B*) times rxn 4
2. This removes all *A*, *B*, (at least one of A_2 or B_2), and *K*, so this is terminal. By Corollary 2 it stably computes whatever #*Y* is now, which is…
3. CRN produces #*A*+#*B* count of *Y* by rxns 1 and 2, and consumes min(#*A*,#*B*) *Y*'s by rxn 4, so computes #*A*+#*B*–min(#*A*,#*B*) = max(#*A*,#*B*). **QED**

In feed-forward CRNs, if there is a terminal configuration, any long enough execution reaches it

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

In feed-forward CRNs, if there is a terminal configuration, any long enough execution reaches it

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

Corollary: In a feed-forward CRN, if there is a terminal configuration $\mathbf{c}_\mathbf{i}$ reachable from initial configuration $\mathbf{i}$, then $\mathbf{c}_\mathbf{i}$ is reached by <u>every</u> sufficiently long execution from $\mathbf{i}$. Furthermore, all of these executions are permutations of the same number of each reaction type.

In feed-forward CRNs, if there is a terminal configuration, any long enough execution reaches it

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

Corollary: In a feed-forward CRN, if there is a terminal configuration $\mathbf{c}_i$ reachable from initial configuration $\mathbf{i}$, then $\mathbf{c}_i$ is reached by <u>every</u> sufficiently long execution from $\mathbf{i}$. Furthermore, all of these executions are permutations of the same number of each reaction type.

Proof:

1. Let P be the execution leading from $\mathbf{i}$ to $\mathbf{c}_i$.

In feed-forward CRNs, if there is a terminal configuration, any long enough execution reaches it

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

Corollary: In a feed-forward CRN, if there is a terminal configuration $\mathbf{c}_\mathbf{i}$ reachable from initial configuration $\mathbf{i}$, then $\mathbf{c}_\mathbf{i}$ is reached by <u>every</u> sufficiently long execution from $\mathbf{i}$. Furthermore, all of these executions are permutations of the same number of each reaction type.

Proof:

1. Let P be the execution leading from $\mathbf{i}$ to $\mathbf{c}_\mathbf{i}$.
2. Any execution Q with with $|Q| > |P|$ must have more of some reaction r by the pigeonhole principle.

In feed-forward CRNs, if there is a terminal configuration, any long enough execution reaches it

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

Corollary: In a feed-forward CRN, if there is a terminal configuration $\mathbf{c_i}$ reachable from initial configuration $\mathbf{i}$, then $\mathbf{c_i}$ is reached by <u>every</u> sufficiently long execution from $\mathbf{i}$. Furthermore, all of these executions are permutations of the same number of each reaction type.

Proof:

1. Let P be the execution leading from $\mathbf{i}$ to $\mathbf{c_i}$.
2. Any execution Q with with $|Q| > |P|$ must have more of some reaction r by the pigeonhole principle.
    1. By the Lemma, $\mathbf{c_i}$ is not terminal, a contradiction.

In feed-forward CRNs, if there is a terminal configuration, any long enough execution reaches it

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

Corollary: In a feed-forward CRN, if there is a terminal configuration $\mathbf{c}_\mathbf{i}$ reachable from initial configuration $\mathbf{i}$, then $\mathbf{c}_\mathbf{i}$ is reached by <u>every</u> sufficiently long execution from $\mathbf{i}$. Furthermore, all of these executions are permutations of the same number of each reaction type.

Proof:

1. Let P be the execution leading from $\mathbf{i}$ to $\mathbf{c}_\mathbf{i}$.
2. Any execution Q with with $|Q| > |P|$ must have more of some reaction r by the pigeonhole principle.
    1. By the Lemma, $\mathbf{c}_\mathbf{i}$ is not terminal, a contradiction.
    2. So no execution Q is longer than P.

In feed-forward CRNs, if there is a terminal configuration, any long enough execution reaches it

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

Corollary: In a feed-forward CRN, if there is a terminal configuration $\mathbf{c}_\mathbf{i}$ reachable from initial configuration $\mathbf{i}$, then $\mathbf{c}_\mathbf{i}$ is reached by <u>every</u> sufficiently long execution from $\mathbf{i}$. Furthermore, all of these executions are permutations of the same number of each reaction type.

Proof:

1. Let P be the execution leading from $\mathbf{i}$ to $\mathbf{c}_\mathbf{i}$.
2. Any execution Q with with $|Q| > |P|$ must have more of some reaction r by the pigeonhole principle.
    1. By the Lemma, $\mathbf{c}_\mathbf{i}$ is not terminal, a contradiction.
    2. So no execution Q is longer than P.
3. Any execution Q with $|Q| = |P|$ must be a permutation of P, or else by pigeonhole Q would have more of some reaction, and this would again contradict the terminality of $\mathbf{c}_\mathbf{i}$.

In feed-forward CRNs, if there is a terminal configuration, any long enough execution reaches it

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

Corollary: In a feed-forward CRN, if there is a terminal configuration $\mathbf{c}_i$ reachable from initial configuration **i**, then $\mathbf{c}_i$ is reached by <u>every</u> sufficiently long execution from **i**. Furthermore, all of these executions are permutations of the same number of each reaction type.

Proof:

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

Noncompetitive CRNs

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

[M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Noncompetitive CRNs

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

Example: The max(A,B) CRN:

1. $A \to Y + A_2$ (A isn't a reactant elsewhere)
2. $B \to Y + B_2$ (B isn't a reactant elsewhere)
3. $A_2 + B_2 \to K$ (A_2,B_2 aren't reactants elsewhere)
4. $K + Y \to \emptyset$ (K, Y aren't reactants elsewhere)

[M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Noncompetitive CRNs

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

Example: The max(A,B) CRN:

1. $A \to Y + A_2$ (A isn't a reactant elsewhere)
2. $B \to Y + B_2$ (B isn't a reactant elsewhere)
3. $A_2 + B_2 \to K$ (A_2,B_2 aren't reactants elsewhere)
4. $K + Y \to \emptyset$ (K, Y aren't reactants elsewhere)

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

[M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Noncompetitive CRNs

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

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

Example: The max(A,B) CRN:

1. $A \to Y+A_2$ (A isn't a reactant elsewhere)
2. $B \to Y+B_2$ (B isn't a reactant elsewhere)
3. $A_2+B_2 \to K$ (A_2,B_2 aren't reactants elsewhere)
4. $K+Y \to \emptyset$ (K, Y aren't reactants elsewhere)

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

[M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Noncompetitive CRNs

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

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

Example: The max(A,B) CRN:

1. $A \to Y+A_2$ (A isn't a reactant elsewhere)
2. $B \to Y+B_2$ (B isn't a reactant elsewhere)
3. $A_2+B_2 \to K$ (A_2,B_2 aren't reactants elsewhere)
4. $K+Y \to \emptyset$ (K, Y aren't reactants elsewhere)

#(r,P) = number of times r occurs in P

Proof:

1. Q' = longest prefix ($\mathbf{i}$, $\mathbf{x}_1$, …, $\mathbf{x}_p$) of Q such that $\#(r,P) \geq \#(r,Q)$ for all reactions r.

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

[M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Noncompetitive CRNs

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

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

Example: The max(A,B) CRN:

1. $A \to Y+A_2$ (A isn't a reactant elsewhere)
2. $B \to Y+B_2$ (B isn't a reactant elsewhere)
3. $A_2+B_2 \to K$ (A_2,B_2 aren't reactants elsewhere)
4. $K+Y \to \emptyset$ (K, Y aren't reactants elsewhere)

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

Proof:

#(r,P) = number of times r occurs in P

1. Q' = longest prefix ($\mathbf{i}$, $\mathbf{x}_1$, …, $\mathbf{x}_p$) of Q such that $\#(r,P) \geq \#(r,Q)$ for all reactions r.
   - i.e., $\mathbf{x}_{p+1}$ is the first time in Q that some reaction exceeds its count in P.

[M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Noncompetitive CRNs

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

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

Example: The max(A,B) CRN:

1. $A \to Y+A_2$ (A isn't a reactant elsewhere)
2. $B \to Y+B_2$ (B isn't a reactant elsewhere)
3. $A_2+B_2 \to K$ (A_2,B_2 aren't reactants elsewhere)
4. $K+Y \to \emptyset$ (K, Y aren't reactants elsewhere)

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

#(r,P) = number of times r occurs in P

Proof:

1. Q' = longest prefix ($\mathbf{i}$, $\mathbf{x}_1$, …, $\mathbf{x}_p$) of Q such that #(r,P) ≥ #(r,Q) for all reactions r.
   - i.e., $\mathbf{x}_{p+1}$ is the first time in Q that some reaction exceeds its count in P.
2. Let r be the reaction such that $\mathbf{x}_p \Longrightarrow \mathbf{x}_{p+1}$ via r.

[M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Noncompetitive CRNs

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

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

Example: The max(A,B) CRN:

1. $A \to Y+A_2$ (A isn't a reactant elsewhere)
2. $B \to Y+B_2$ (B isn't a reactant elsewhere)
3. $A_2+B_2 \to K$ (A_2,B_2 aren't reactants elsewhere)
4. $K+Y \to \emptyset$ (K, Y aren't reactants elsewhere)

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

#(r,P) = number of times r occurs in P

Proof:

1. Q' = longest prefix $(\mathbf{i}, \mathbf{x}_1, \ldots, \mathbf{x}_p)$ of Q such that $\#(r,P) \geq \#(r,Q)$ for all reactions r.
   - i.e., $\mathbf{x}_{p+1}$ is the <u>first</u> time in Q that some reaction <u>exceeds</u> its count in P.
2. Let r be the reaction such that $\mathbf{x}_p \Longrightarrow \mathbf{x}_{p+1}$ via r.
3. Note $\#(r,P) = \#(r,Q')$ and $\#(t,P) \geq \#(t,Q')$ for all other reactions $t \neq r$.

[M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Noncompetitive CRNs

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

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

Example: The max(A,B) CRN:

1. $A \to Y+A_2$ (A isn't a reactant elsewhere)
2. $B \to Y+B_2$ (B isn't a reactant elsewhere)
3. $A_2+B_2 \to K$ (A_2,B_2 aren't reactants elsewhere)
4. $K+Y \to \emptyset$ (K, Y aren't reactants elsewhere)

#(r,P) = number of times r occurs in P

Proof:

1. Q' = longest prefix ($\mathbf{i}$, $\mathbf{x}_1$, …, $\mathbf{x}_p$) of Q such that #(r,P) ≥ #(r,Q) for all reactions r.
   - i.e., $\mathbf{x}_{p+1}$ is the <u>first</u> time in Q that some reaction <u>exceeds</u> its count in P.
2. Let r be the reaction such that $\mathbf{x}_p \Longrightarrow \mathbf{x}_{p+1}$ via r.
3. Note #(r,P) = #(r,Q') and #(t,P) ≥ #(t,Q') for all other reactions $t \neq r$.
4. Since CRN is non-competitive, no reactant A of r can be consumed in $t \neq r$.

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

[M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Noncompetitive CRNs

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

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

Example: The max(A,B) CRN:

1. $A \to Y+A_2$ (A isn't a reactant elsewhere)
2. $B \to Y+B_2$ (B isn't a reactant elsewhere)
3. $A_2+B_2 \to K$ (A_2,B_2 aren't reactants elsewhere)
4. $K+Y \to \emptyset$ (K, Y aren't reactants elsewhere)

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

#(r,P) = number of times r occurs in P

Proof:

1. Q' = longest prefix ($\mathbf{i}$, $\mathbf{x}_1$, …, $\mathbf{x}_p$) of Q such that $\#(r,P) \geq \#(r,Q)$ for all reactions r.
   - i.e., $\mathbf{x}_{p+1}$ is the first time in Q that some reaction exceeds its count in P.
2. Let r be the reaction such that $\mathbf{x}_p \Longrightarrow \mathbf{x}_{p+1}$ via r.
3. Note $\#(r,P) = \#(r,Q')$ and $\#(t,P) \geq \#(t,Q')$ for all other reactions $t \neq r$.
4. Since CRN is non-competitive, no reactant A of r can be consumed in $t \neq r$.
   1. Some other reactions t might produce A.

[M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Noncompetitive CRNs

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

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

Example: The max(A,B) CRN:

1. $A \rightarrow Y+A_2$ (A isn't a reactant elsewhere)
2. $B \rightarrow Y+B_2$ (B isn't a reactant elsewhere)
3. $A_2+B_2 \rightarrow K$ (A_2,B_2 aren't reactants elsewhere)
4. $K+Y \rightarrow \emptyset$ (K, Y aren't reactants elsewhere)

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

$\#(r,P)$ = number of times r occurs in P

Proof:

1. Q' = longest prefix $(\mathbf{i}, \mathbf{x}_1, \ldots, \mathbf{x}_p)$ of Q such that $\#(r,P) \geq \#(r,Q)$ for all reactions r.
   - i.e., $\mathbf{x}_{p+1}$ is the first time in Q that some reaction exceeds its count in P.
2. Let r be the reaction such that $\mathbf{x}_p \Longrightarrow \mathbf{x}_{p+1}$ via r.
3. Note $\#(r,P) = \#(r,Q')$ and $\#(t,P) \geq \#(t,Q')$ for all other reactions $t \neq r$.
4. Since CRN is non-competitive, no reactant A of r can be consumed in $t \neq r$.
   1. Some other reactions t might produce A.
   2. Since $\#(t,P) \geq \#(t,Q')$, each $t \neq r$ produces at least as much A in P has in Q'.

[M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Noncompetitive CRNs

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

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

Example: The max(A,B) CRN:

1. $A \to Y+A_2$ (A isn't a reactant elsewhere)
2. $B \to Y+B_2$ (B isn't a reactant elsewhere)
3. $A_2+B_2 \to K$ (A_2,B_2 aren't reactants elsewhere)
4. $K+Y \to \emptyset$ (K, Y aren't reactants elsewhere)

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

#(r,P) = number of times r occurs in P

Proof:

1. Q' = longest prefix ($\mathbf{i}$, $\mathbf{x}_1$, …, $\mathbf{x}_p$) of Q such that $\#(r,P) \geq \#(r,Q)$ for all reactions r.
   - i.e., $\mathbf{x}_{p+1}$ is the first time in Q that some reaction exceeds its count in P.
2. Let r be the reaction such that $\mathbf{x}_p \Longrightarrow \mathbf{x}_{p+1}$ via r.
3. Note $\#(r,P) = \#(r,Q')$ and $\#(t,P) \geq \#(t,Q')$ for all other reactions $t \neq r$.
4. Since CRN is non-competitive, no reactant A of r can be consumed in $t \neq r$.
   1. Some other reactions t might produce A.
   2. Since $\#(t,P) \geq \#(t,Q')$, each $t \neq r$ produces at least as much A in P has in Q'.
   3. Exactly as much A is consumed by r in P as in Q'.

[M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Noncompetitive CRNs

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

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

Example: The max(A,B) CRN:

1. $A \to Y+A_2$ (A isn't a reactant elsewhere)
2. $B \to Y+B_2$ (B isn't a reactant elsewhere)
3. $A_2+B_2 \to K$ (A_2,B_2 aren't reactants elsewhere)
4. $K+Y \to \emptyset$ (K, Y aren't reactants elsewhere)

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

#(r,P) = number of times r occurs in P

Proof:

1. Q' = longest prefix $(\mathbf{i}, \mathbf{x}_1, \ldots, \mathbf{x}_p)$ of Q such that $\#(r,P) \geq \#(r,Q)$ for all reactions r.
   - i.e., $\mathbf{x}_{p+1}$ is the first time in Q that some reaction exceeds its count in P.
2. Let r be the reaction such that $\mathbf{x}_p \Longrightarrow \mathbf{x}_{p+1}$ via r.
3. Note $\#(r,P) = \#(r,Q')$ and $\#(t,P) \geq \#(t,Q')$ for all other reactions $t \neq r$.
4. Since CRN is non-competitive, no reactant A of r can be consumed in $t \neq r$.
   1. Some other reactions t might produce A.
   2. Since $\#(t,P) \geq \#(t,Q')$, each $t \neq r$ produces at least as much A in P has in Q'.
   3. Exactly as much A is consumed by r in P as in Q'.
   4. Thus $\mathbf{x}_p(A) \leq \mathbf{c}(A)$ for all reactants A of r.

[M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Noncompetitive CRNs

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

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

Example: The max(A,B) CRN:

1. $A \to Y+A_2$ (A isn't a reactant elsewhere)
2. $B \to Y+B_2$ (B isn't a reactant elsewhere)
3. $A_2+B_2 \to K$ (A_2,B_2 aren't reactants elsewhere)
4. $K+Y \to \emptyset$ (K, Y aren't reactants elsewhere)

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

#(r,P) = number of times r occurs in P

Proof:

1. Q' = longest prefix $(\mathbf{i}, \mathbf{x}_1, \ldots, \mathbf{x}_p)$ of Q such that $\#(r,P) \geq \#(r,Q)$ for all reactions r.
   - i.e., $\mathbf{x}_{p+1}$ is the first time in Q that some reaction exceeds its count in P.
2. Let r be the reaction such that $\mathbf{x}_p \Longrightarrow \mathbf{x}_{p+1}$ via r.
3. Note $\#(r,P) = \#(r,Q')$ and $\#(t,P) \geq \#(t,Q')$ for all other reactions $t \neq r$.
4. Since CRN is non-competitive, no reactant A of r can be consumed in $t \neq r$.
   1. Some other reactions t might produce A.
   2. Since $\#(t,P) \geq \#(t,Q')$, each $t \neq r$ produces at least as much A in P has in Q'.
   3. Exactly as much A is consumed by r in P as in Q'.
   4. Thus $\mathbf{x}_p(A) \leq \mathbf{c}(A)$ for all reactants A of r.
5. Since r is applicable to $\mathbf{x}_p$, it is applicable to $\mathbf{c}$.

[M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Noncompetitive CRNs

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

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

Example: The max(A,B) CRN:

1. $A \to Y+A_2$ (A isn't a reactant elsewhere)
2. $B \to Y+B_2$ (B isn't a reactant elsewhere)
3. $A_2+B_2 \to K$ (A_2,B_2 aren't reactants elsewhere)
4. $K+Y \to \emptyset$ (K, Y aren't reactants elsewhere)

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

#(r,P) = number of times r occurs in P

Proof:

1. Q' = longest prefix $(\mathbf{i}, \mathbf{x}_1, \ldots, \mathbf{x}_p)$ of Q such that $\#(r,P) \geq \#(r,Q)$ for all reactions r.
   - i.e., $\mathbf{x}_{p+1}$ is the <u>first</u> time in Q that some reaction <u>exceeds</u> its count in P.
2. Let r be the reaction such that $\mathbf{x}_p \Longrightarrow \mathbf{x}_{p+1}$ via r.
3. Note $\#(r,P) = \#(r,Q')$ and $\#(t,P) \geq \#(t,Q')$ for all other reactions $t \neq r$.
4. Since CRN is non-competitive, no reactant A of r can be consumed in $t \neq r$.
   1. Some other reactions t might <u>produce</u> A.
   2. Since $\#(t,P) \geq \#(t,Q')$, each $t \neq r$ produces at least as much A in P has in Q'.
   3. Exactly as much A is consumed by r in P as in Q'.
   4. Thus $\mathbf{x}_p(A) \leq \mathbf{c}(A)$ for all reactants A of r.
5. Since r is applicable to $\mathbf{x}_p$, it is applicable to $\mathbf{c}$.
6. So $\mathbf{c}$ is not terminal. **QED**

[M. Vasić, C. Chalk, A. Luchsinger, S. Khurshid, and D. Soloveichik. Programming and training rate-independent chemical reaction networks. *Proceedings of the National Academy of Sciences*, 119(24):e2111552119, 2022.]

Non-feedforward CRNs

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

Non-feedforward CRNs

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

$f(x) = x/2$

1. $X \rightleftharpoons Y + A$

2. $X + A \rightarrow \emptyset$

Non-feedforward CRNs

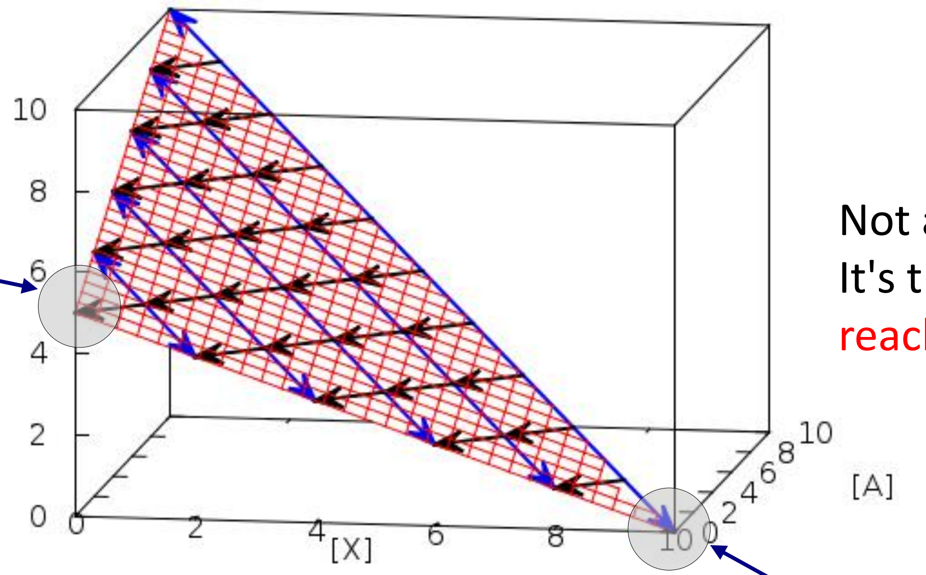
Example of a non-feedforward CRN that stably computes a function?

$f(x) = x/2$

1. $X \rightleftharpoons Y+A$

2. $X+A \rightarrow \emptyset$

final config



initial config

Not a plot of f!
It's the space of reachable states.

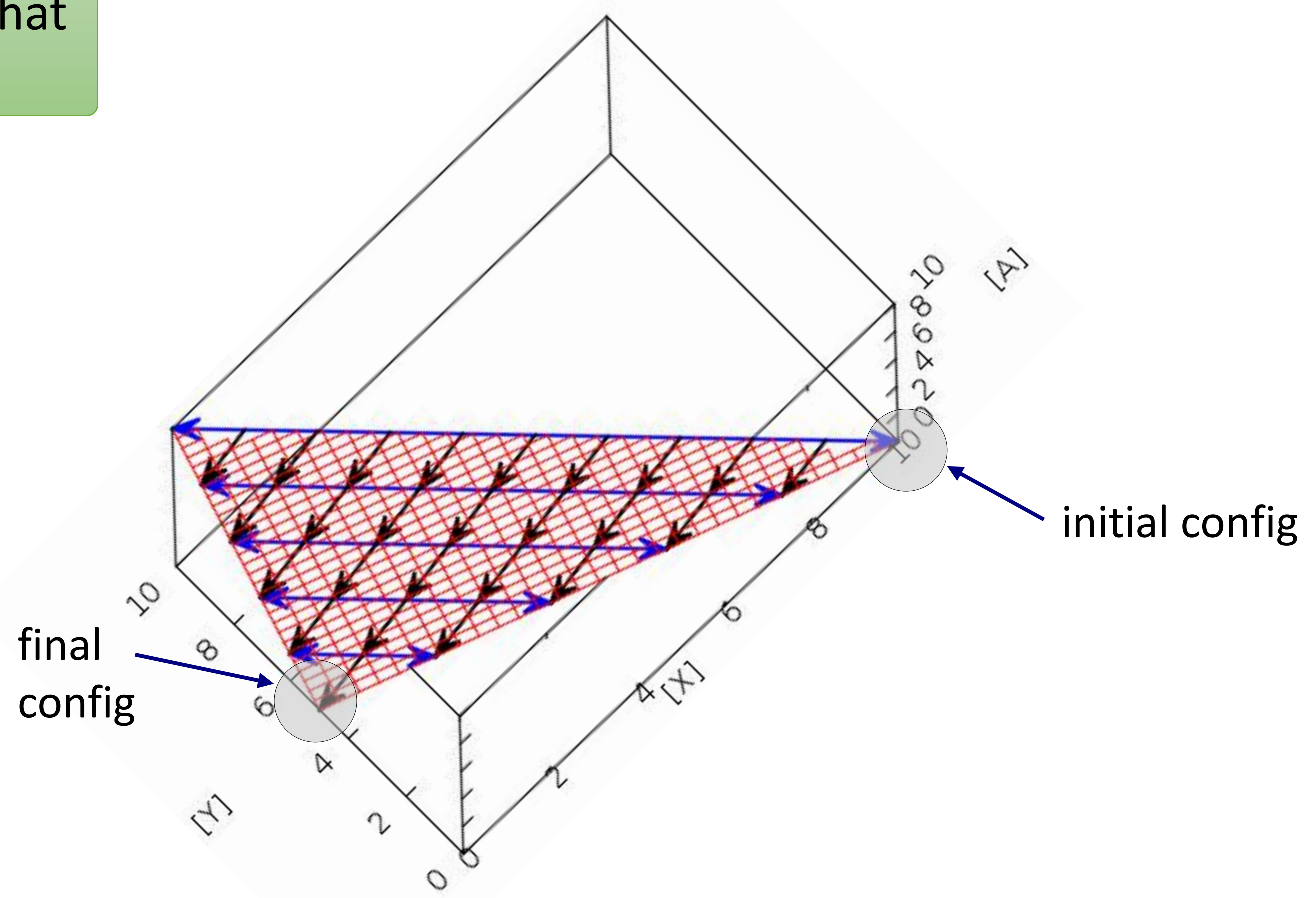
Non-feedforward CRNs

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

$f(x) = x/2$

1. $X \rightleftharpoons Y+A$

2. $X+A \rightarrow \emptyset$



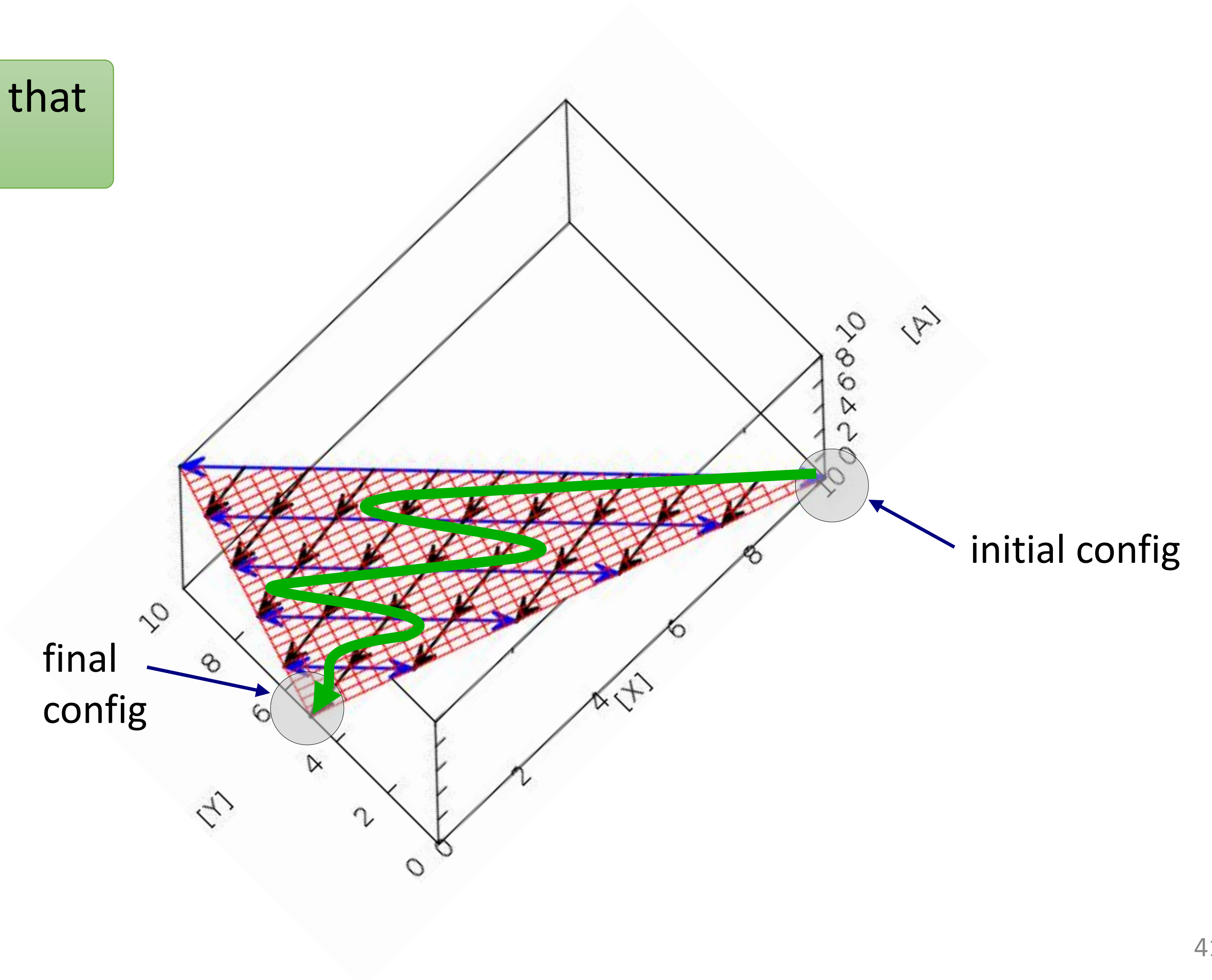
Non-feedforward CRNs

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

$f(x) = x/2$

1. $X \rightleftharpoons Y+A$

2. $X+A \rightarrow \emptyset$



Non-feedforward CRNs

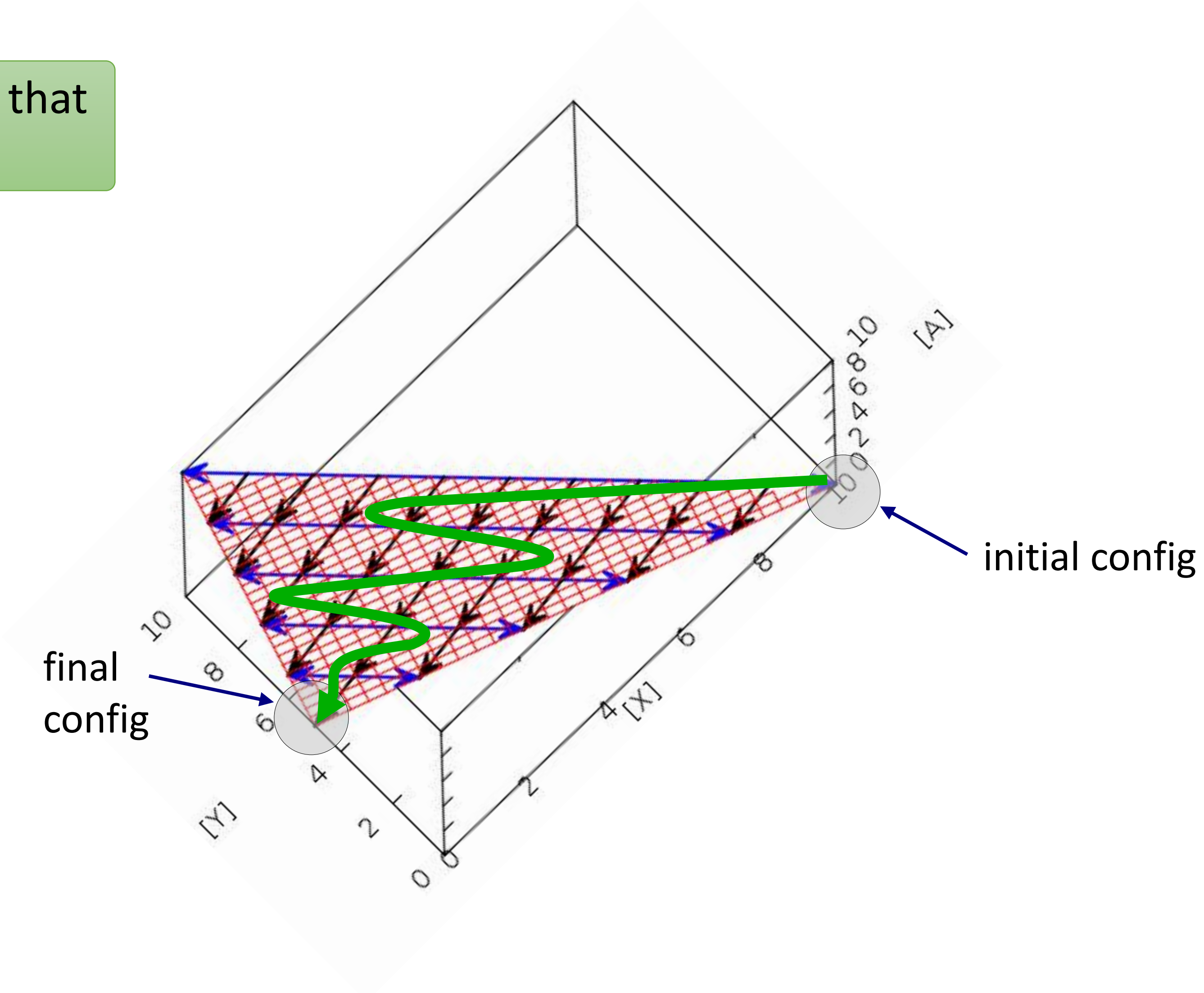
Example of a non-feedforward CRN that stably computes a function?

$f(x) = x/2$

1. $X \rightleftharpoons Y+A$

2. $X+A \rightarrow \emptyset$

It's even non-non-competitive!



Time complexity of CRNs

What is probable:
Stochastic kinetic model of chemical reaction networks

Solution volume *v*

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

[McQuarrie 1967, van Kampen, Gillespie 1977]

What is probable:
Stochastic kinetic model of chemical reaction networks

Solution volume *v*

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

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

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

$x,y \rightarrow x,x$ $a,b \rightarrow c,d$ $a,a \rightarrow a,a$ (*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]: Winner of 2020 Dijkstra Prize in Distributed Computing: https://www.podc.org/dijkstra/2020-dijkstra-prize/

Relationship to distributed computing

population protocol = list of *transitions* such as

$x,y \rightarrow x,x$ $\qquad$ $a,b \rightarrow c,d$ $\qquad$ $a,a \rightarrow a,a$ (*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]: Winner of 2020 Dijkstra Prize in Distributed Computing: https://www.podc.org/dijkstra/2020-dijkstra-prize/

Relationship to distributed computing

population protocol = list of *transitions* such as

$x,y \to x,x$ $a,b \to c,d$ $a,a \to a,a$ (*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]:
Winner of 2020 Dijkstra Prize in Distributed Computing: https://www.podc.org/dijkstra/2020-dijkstra-prize/

Relationship to distributed computing

population protocol = list of *transitions* such as

$x,y \rightarrow x,x$ $\quad\quad$ $a,b \rightarrow c,d$ $\quad\quad$ $a,a \rightarrow a,a$ (*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]:
Winner of 2020 Dijkstra Prize in Distributed Computing: https://www.podc.org/dijkstra/2020-dijkstra-prize/

Relationship to distributed computing

population protocol = list of *transitions* such as

$x,y \to x,x$ $\quad\quad$ $a,b \to c,d$ $\quad\quad$ $a,a \to a,a$ (*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]: Winner of 2020 Dijkstra Prize in Distributed Computing: https://www.podc.org/dijkstra/2020-dijkstra-prize/

Time complexity in population protocols

- pair of agents picked uniformly at random to interact (possibly null interaction)
- *parallel time* = number of interactions / n

  i.e., each agent has $O(1)$ interactions per "unit time"

Speed of computation

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
reasonable requirement on volume: $v = O(n)$
i.e., require bounded concentration (finite density constraint)

Full CRN time model (*Gillespie kinetics*)

Full CRN time model (*Gillespie kinetics*)

- What *should* influence total rate λ (a.k.a., propensity) of bimolecular reaction $A+B \xrightarrow{k} C$?

Full CRN time model (*Gillespie kinetics*)

- What *should* influence total rate λ (a.k.a., propensity) of bimolecular reaction $A+B \xrightarrow{k} C$?
  - molecular counts of reactants: $\lambda \propto \#A \cdot \#B$ (*the more there are, the faster collisions happen*)

Full CRN time model (*Gillespie kinetics*)

- What *should* influence total rate λ (a.k.a., propensity) of bimolecular reaction $A+B \xrightarrow{k} C$?
  - molecular counts of reactants: $\lambda \propto \#A \cdot \#B$ (*the more there are, the faster collisions happen*)
  - volume v: $\lambda \propto 1/v$ (*the bigger the volume, the slower collisions happen*)

Full CRN time model (*Gillespie kinetics*)

- What *should* influence total rate λ (a.k.a., propensity) of bimolecular reaction $A+B \xrightarrow{k} C$?
  - molecular counts of reactants: $\lambda \propto \#A \cdot \#B$ (*the more there are, the faster collisions happen*)
  - volume v: $\lambda \propto 1/v$ (*the bigger the volume, the slower collisions happen*)
  - rate constant k: $\lambda \propto k$ (*captures things not directly modeled, e.g., diffusion rates, probability that a collision results in a reaction*)

Full CRN time model (*Gillespie kinetics*)

- What *should* influence total rate λ (a.k.a., propensity) of bimolecular reaction $A+B \xrightarrow{k} C$?
  - molecular counts of reactants: $\lambda \propto \#A \cdot \#B$ (*the more there are, the faster collisions happen*)
  - volume v: $\lambda \propto 1/v$ (*the bigger the volume, the slower collisions happen*)
  - rate constant k: $\lambda \propto k$ (*captures things not directly modeled, e.g., diffusion rates, probability that a collision results in a reaction*)
- For this example reaction $A+B \xrightarrow{k} C$, combining these we get $\lambda = k \cdot \#A \cdot \#B \,/\, v$

Full CRN time model (*Gillespie kinetics*)

- What *should* influence total rate λ (a.k.a., propensity) of bimolecular reaction $A+B \xrightarrow{k} C$?
  - molecular counts of reactants: $\lambda \propto \#A \cdot \#B$ (*the more there are, the faster collisions happen*)
  - volume v: $\lambda \propto 1/v$ (*the bigger the volume, the slower collisions happen*)
  - rate constant k: $\lambda \propto k$ (*captures things not directly modeled, e.g., diffusion rates, probability that a collision results in a reaction*)
- For this example reaction $A+B \xrightarrow{k} C$, combining these we get $\lambda = k \cdot \#A \cdot \#B \,/\, v$
- Other reaction types:
  - $A+A \xrightarrow{k} \ldots$ $\lambda = k \cdot \#A \cdot (\#A - 1) \,/\, v$ (*symmetric bimolecular reaction*)

Full CRN time model (*Gillespie kinetics*)

- What *should* influence total rate λ (a.k.a., propensity) of bimolecular reaction $A+B \xrightarrow{k} C$?
  - molecular counts of reactants: $\lambda \propto \#A \cdot \#B$ (*the more there are, the faster collisions happen*)
  - volume v: $\lambda \propto 1/v$ (*the bigger the volume, the slower collisions happen*)
  - rate constant k: $\lambda \propto k$ (*captures things not directly modeled, e.g., diffusion rates, probability that a collision results in a reaction*)
- For this example reaction $A+B \xrightarrow{k} C$, combining these we get $\lambda = k \cdot \#A \cdot \#B \,/\, v$
- Other reaction types:
  - $A+A \xrightarrow{k} \ldots$ $\lambda = k \cdot \#A \cdot (\#A-1) \,/\, v$ (*symmetric bimolecular reaction*)
    - $\#A \cdot (\#A-1)/2$ = # ways to pick two A's to react; factor ½ by convention is put into rate constant k

Full CRN time model (*Gillespie kinetics*)

- What *should* influence total rate λ (a.k.a., propensity) of bimolecular reaction $A+B \xrightarrow{k} C$?
  - molecular counts of reactants: $\lambda \propto \#A \cdot \#B$ (*the more there are, the faster collisions happen*)
  - volume v: $\lambda \propto 1/v$ (*the bigger the volume, the slower collisions happen*)
  - rate constant k: $\lambda \propto k$ (*captures things not directly modeled, e.g., diffusion rates, probability that a collision results in a reaction*)
- For this example reaction $A+B \xrightarrow{k} C$, combining these we get $\lambda = k \cdot \#A \cdot \#B / v$
- Other reaction types:
  - $A+A \xrightarrow{k} \ldots$ $\lambda = k \cdot \#A \cdot (\#A - 1) / v$ (*symmetric bimolecular reaction*)
    - $\#A \cdot (\#A - 1)/2$ = # ways to pick two A's to react; factor ½ by convention is put into rate constant k
  - $A \xrightarrow{k} \ldots$ $\lambda = k \cdot \#A$ (*unimolecular reaction*)

Full CRN time model (*Gillespie kinetics*)

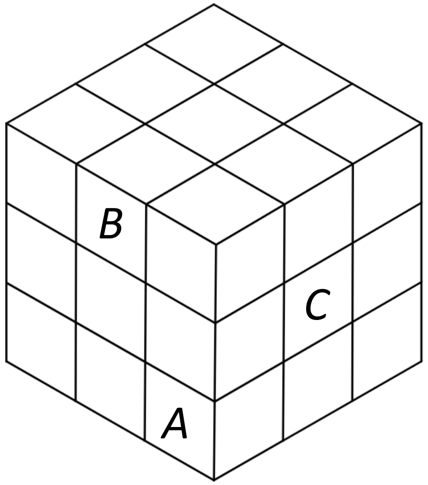
- What *should* influence total rate λ (a.k.a., propensity) of bimolecular reaction $A+B \xrightarrow{k} C$?
  - molecular counts of reactants: $\lambda \propto \#A \cdot \#B$ (*the more there are, the faster collisions happen*)
  - volume v: $\lambda \propto 1/v$ (*the bigger the volume, the slower collisions happen*)
  - rate constant k: $\lambda \propto k$ (*captures things not directly modeled, e.g., diffusion rates, probability that a collision results in a reaction*)
- For this example reaction $A+B \xrightarrow{k} C$, combining these we get $\lambda = k \cdot \#A \cdot \#B \,/\, v$
- Other reaction types:
  - $A+A \xrightarrow{k} \ldots$ $\lambda = k \cdot \#A \cdot (\#A - 1) \,/\, v$ (*symmetric bimolecular reaction*)
    - $\#A \cdot (\#A-1)/2$ = # ways to pick two A's to react; factor ½ by convention is put into rate constant k
  - $A \xrightarrow{k} \ldots$ $\lambda = k \cdot \#A$ (*unimolecular reaction*)
    - no volume term since no collision required

Full CRN time model (*Gillespie kinetics*)

- What *should* influence total rate λ (a.k.a., propensity) of bimolecular reaction $A+B \xrightarrow{k} C$?
  - molecular counts of reactants: $\lambda \propto \#A \cdot \#B$ (*the more there are, the faster collisions happen*)
  - volume v: $\lambda \propto 1/v$ (*the bigger the volume, the slower collisions happen*)
  - rate constant k: $\lambda \propto k$ (*captures things not directly modeled, e.g., diffusion rates, probability that a collision results in a reaction*)
- For this example reaction $A+B \xrightarrow{k} C$, combining these we get $\lambda = k \cdot \#A \cdot \#B / v$
- Other reaction types:
  - $A+A \xrightarrow{k} \ldots$ $\lambda = k \cdot \#A \cdot (\#A-1) / v$ (*symmetric bimolecular reaction*)
    - $\#A \cdot (\#A-1)/2$ = # ways to pick two A's to react; factor ½ by convention is put into rate constant k
  - $A \xrightarrow{k} \ldots$ $\lambda = k \cdot \#A$ (*unimolecular reaction*)
    - no volume term since no collision required
  - $A+B+C \xrightarrow{k} \ldots$ $\lambda = k \cdot \#A \cdot \#B \cdot \#C / v^2$ (*trimolecular reaction*)

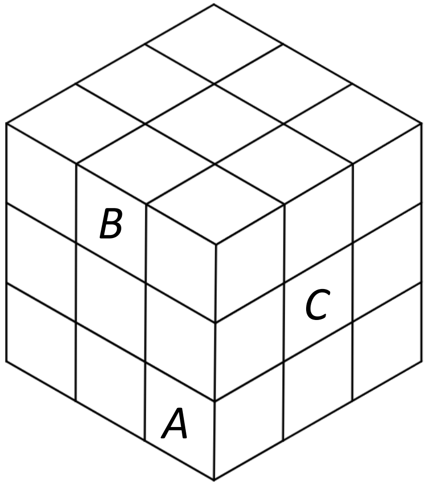
Full CRN time model (*Gillespie kinetics*)

- What *should* influence total rate λ (a.k.a., propensity) of bimolecular reaction $A+B \xrightarrow{k} C$?
  - molecular counts of reactants: $\lambda \propto \#A \cdot \#B$ (*the more there are, the faster collisions happen*)
  - volume v: $\lambda \propto 1/v$ (*the bigger the volume, the slower collisions happen*)
  - rate constant k: $\lambda \propto k$ (*captures things not directly modeled, e.g., diffusion rates, probability that a collision results in a reaction*)
- For this example reaction $A+B \xrightarrow{k} C$, combining these we get $\lambda = k \cdot \#A \cdot \#B \,/\, v$
- Other reaction types:
  - $A+A \xrightarrow{k} \ldots$ $\quad \lambda = k \cdot \#A \cdot (\#A - 1) \,/\, v$ (*symmetric bimolecular reaction*)
    - $\#A \cdot (\#A-1)/2$ = # ways to pick two A's to react; factor ½ by convention is put into rate constant k
  - $A \xrightarrow{k} \ldots$ $\quad \lambda = k \cdot \#A$ (*unimolecular reaction*)
    - no volume term since no collision required
  - $A+B+C \xrightarrow{k} \ldots$ $\quad \lambda = k \cdot \#A \cdot \#B \cdot \#C \,/\, v^2$ (*trimolecular reaction*)
    - The volume term is squared because (roughly) if we define coordinate system so position of A is always at the origin, then B and C are randomly moving around through v volume "cells", and it takes v^2 expected time for them both to occupy the origin, to cause a three-way A-B-C collision



Full CRN time model (*Gillespie kinetics*)

- What *should* influence total rate λ (a.k.a., propensity) of bimolecular reaction $A+B \xrightarrow{k} C$?
  - molecular counts of reactants: $\lambda \propto \#A \cdot \#B$ (*the more there are, the faster collisions happen*)
  - volume v: $\lambda \propto 1/v$ (*the bigger the volume, the slower collisions happen*)
  - rate constant k: $\lambda \propto k$ (*captures things not directly modeled, e.g., diffusion rates, probability that a collision results in a reaction*)
- For this example reaction $A+B \xrightarrow{k} C$, combining these we get $\lambda = k \cdot \#A \cdot \#B / v$
- Other reaction types:
  - $A+A \xrightarrow{k} \ldots$ $\quad \lambda = k \cdot \#A \cdot (\#A - 1) / v$ (*symmetric bimolecular reaction*)
    - $\#A \cdot (\#A - 1)/2$ = # ways to pick two A's to react; factor ½ by convention is put into rate constant k
  - $A \xrightarrow{k} \ldots$ $\quad \lambda = k \cdot \#A$ (*unimolecular reaction*)
    - no volume term since no collision required
  - $A+B+C \xrightarrow{k} \ldots$ $\quad \lambda = k \cdot \#A \cdot \#B \cdot \#C / v^2$ (*trimolecular reaction*)
    - The volume term is squared because (roughly) if we define coordinate system so position of A is always at the origin, then B and C are randomly moving around through v volume "cells", and it takes v^2 expected time for them both to occupy the origin, to cause a three-way A-B-C collision
  - In general, with r reactants, propensity is number of ways to pick reactants, times k, divided by v^{r-1}



Discrete versus continuous time

- Time between interactions in CRN model is <u>exponential</u> random variable **T**
- Time between interactions in PP model is <u>geometric</u> random variable **T**
    - **T** = # of coin flips until a heads, with Pr[heads] = p, E[time] = $1/p$
    - (essentially the discrete version of an exponential random variable)

Discrete versus continuous time

- Time between interactions in CRN model is <u>exponential</u> random variable **T**
- Time between interactions in PP model is <u>geometric</u> random variable **T**
  - **T** = # of coin flips until a heads, with Pr[heads] = p, E[time] = $1/p$
  - (essentially the discrete version of an exponential random variable)

both are *memoryless*: $\forall\ s,r > 0$
Pr[**T** > $s + r$ | **T** > s] = Pr[**T** > r]

Discrete versus continuous time

- Time between interactions in CRN model is <u>exponential</u> random variable **T**
- Time between interactions in PP model is <u>geometric</u> random variable **T**
  - **T** = # of coin flips until a heads, with Pr[heads] = p, E[time] = 1/p
  - (essentially the discrete version of an exponential random variable)

both are *memoryless*: $\forall\ s,r > 0$

$$\Pr[\ \mathbf{T} > s + r \mid \underbrace{\mathbf{T} > s}_{\text{no rxn in first } s \text{ secs.}}\ ] = \Pr[\ \mathbf{T} > r\ ]$$

Discrete versus continuous time

- Time between interactions in CRN model is exponential random variable **T**
- Time between interactions in PP model is geometric random variable **T**
  - **T** = # of coin flips until a heads, with Pr[heads] = p, E[time] = $1/p$
  - (essentially the discrete version of an exponential random variable)

both are *memoryless*: $\forall\ s,r > 0$

$$\Pr[\ \underbrace{\mathbf{T} > s + r}_{\text{no rxn after } r \text{ additional secs.}} \mid \underbrace{\mathbf{T} > s}_{\text{no rxn in first } s \text{ secs.}}\ ] = \Pr[\ \mathbf{T} > r\ ]$$

Discrete versus continuous time

- Time between interactions in CRN model is <u>exponential</u> random variable **T**
- Time between interactions in PP model is <u>geometric</u> random variable **T**
  - **T** = # of coin flips until a heads, with Pr[heads] = p, E[time] = $1/p$
  - (essentially the discrete version of an exponential random variable)

both are *memoryless*: $\forall\ s,r > 0$

$$\Pr[\ \underbrace{\mathbf{T} > s + r}_{\text{no rxn after } r \text{ additional secs.}}\ |\ \underbrace{\mathbf{T} > s}_{\text{no rxn in first } s \text{ secs.}}\ ] = \Pr[\ \underbrace{\mathbf{T} > r}_{\text{no rxn in first } r \text{ secs.}}\ ]$$

Discrete versus continuous time

- Time between interactions in CRN model is <u>exponential</u> random variable **T**
- Time between interactions in PP model is <u>geometric</u> random variable **T**
  - **T** = # of coin flips until a heads, with Pr[heads] = p, E[time] = $1/p$
  - (essentially the discrete version of an exponential random variable)
  - in population protocol, heads event = non-null interaction

both are *memoryless*: $\forall\ s,r > 0$

$$\Pr[\ \underbrace{\mathbf{T} > s + r}_{\text{no rxn after } r \text{ additional secs.}}\ |\ \underbrace{\mathbf{T} > s}_{\text{no rxn in first } s \text{ secs.}}\ ] = \Pr[\ \underbrace{\mathbf{T} > r}_{\text{no rxn in first } r \text{ secs.}}\ ]$$

Discrete versus continuous time

- Time between interactions in CRN model is exponential random variable **T**
- Time between interactions in PP model is geometric random variable **T**
  - **T** = # of coin flips until a heads, with Pr[heads] = p, E[time] = $1/p$
  - (essentially the discrete version of an exponential random variable)
  - in population protocol, heads event = non-null interaction
- CRN model: time = sum of exponential random variables

both are *memoryless*: $\forall\ s,r > 0$

$$\Pr[\ \underbrace{\mathbf{T} > s + r}_{\text{no rxn after } r \text{ additional secs.}}\ |\ \underbrace{\mathbf{T} > s}_{\text{no rxn in first } s \text{ secs.}}\ ] = \Pr[\ \underbrace{\mathbf{T} > r}_{\text{no rxn in first } r \text{ secs.}}\ ]$$

Discrete versus continuous time

- Time between interactions in CRN model is <u>exponential</u> random variable **T**
- Time between interactions in PP model is <u>geometric</u> random variable **T**
  - **T** = # of coin flips until a heads, with Pr[heads] = p, E[time] = $1/p$
  - (essentially the discrete version of an exponential random variable)
  - in population protocol, heads event = non-null interaction
- CRN model: time = sum of exponential random variables
  - define volume = n, and rate of interaction $a,b \rightarrow \ldots$ as $\#a \cdot \#b \,/\, n$, i.e., expected time $n \,/\, (\#a \cdot \#b)$

both are *memoryless*: $\forall\, s,r > 0$

$$\Pr[\,\underbrace{\mathbf{T} > s + r}_{\text{no rxn after } r \text{ additional secs.}} \mid \underbrace{\mathbf{T} > s}_{\text{no rxn in first } s \text{ secs.}}\,] = \Pr[\,\underbrace{\mathbf{T} > r}_{\text{no rxn in first } r \text{ secs.}}\,]$$

Discrete versus continuous time

- Time between interactions in CRN model is <u>exponential</u> random variable **T**
- Time between interactions in PP model is <u>geometric</u> random variable **T**
  - **T** = # of coin flips until a heads, with Pr[heads] = p, E[time] = $1/p$
  - (essentially the discrete version of an exponential random variable)
  - in population protocol, heads event = non-null interaction

both are *memoryless*: $\forall\ s,r > 0$

$$\Pr[\ \underbrace{\mathbf{T} > s + r}_{\text{no rxn after } r \text{ additional secs.}}\ |\ \underbrace{\mathbf{T} > s}_{\text{no rxn in first } s \text{ secs.}}\ ] = \Pr[\ \underbrace{\mathbf{T} > r}_{\text{no rxn in first } r \text{ secs.}}\ ]$$

- CRN model: time = sum of exponential random variables
  - define volume = n, and rate of interaction $a,b \rightarrow \ldots$ as $\#a \cdot \#b\ /\ n$, i.e., expected time $n\ /\ (\#a \cdot \#b)$
- PP model (time = #interactions / n)

Discrete versus continuous time

- Time between interactions in CRN model is <u>exponential</u> random variable **T**
- Time between interactions in PP model is <u>geometric</u> random variable **T**
  - **T** = # of coin flips until a heads, with Pr[heads] = p, E[time] = $1/p$
  - (essentially the discrete version of an exponential random variable)
  - in population protocol, heads event = non-null interaction

both are *memoryless*: $\forall\, s,r > 0$

$$\Pr[\, \underbrace{\mathbf{T} > s + r}_{\text{no rxn after } r \text{ \underline{additional} secs.}} \mid \underbrace{\mathbf{T} > s}_{\text{no rxn in first } s \text{ secs.}} \,] = \Pr[\, \underbrace{\mathbf{T} > r}_{\text{no rxn in first } r \text{ secs.}} \,]$$

- CRN model: time = sum of exponential random variables
  - define volume = n, and rate of interaction $a,b \rightarrow \ldots$ as $\#a \cdot \#b \,/\, n$, i.e., expected time $n \,/\, (\#a \cdot \#b)$
- PP model (time = #interactions / n)
  - probability that next interaction is $a,b \rightarrow \ldots$ is $\#a \cdot \#b \,/\, (n \text{ choose } 2) = 2 \cdot \#a \cdot \#b \,/\, (n(n-1))$

Discrete versus continuous time

- Time between interactions in CRN model is <u>exponential</u> random variable **T**
- Time between interactions in PP model is <u>geometric</u> random variable **T**
  - **T** = # of coin flips until a heads, with Pr[heads] = p, E[time] = $1/p$
  - (essentially the discrete version of an exponential random variable)
  - in population protocol, heads event = non-null interaction

both are *memoryless*: $\forall\, s,r > 0$

$$\Pr[\, \underbrace{\mathbf{T} > s + r}_{\text{no rxn after } r \text{ additional secs.}} \mid \underbrace{\mathbf{T} > s}_{\text{no rxn in first } s \text{ secs.}} \,] = \Pr[\, \underbrace{\mathbf{T} > r}_{\text{no rxn in first } r \text{ secs.}} \,]$$

- CRN model: time = sum of exponential random variables
  - define volume = n, and rate of interaction $a,b \rightarrow \ldots$ as $\#a \cdot \#b \,/\, n$, i.e., expected time $n \,/\, (\#a \cdot \#b)$
- PP model (time = #interactions / n)
  - probability that next interaction is $a,b \rightarrow \ldots$ is $\#a \cdot \#b \,/\, (n \text{ choose } 2) = 2 \cdot \#a \cdot \#b \,/\, (n(n-1))$
  - expected interactions until next $a,b \rightarrow \ldots$ interaction = $n(n-1) \,/\, (2 \cdot \#a \cdot \#b)$, i.e., time $(n-1) \,/\, (2 \cdot \#a \cdot \#b)$

Discrete versus continuous time

- Time between interactions in CRN model is <u>exponential</u> random variable **T**
- Time between interactions in PP model is <u>geometric</u> random variable **T**
  - **T** = # of coin flips until a heads, with Pr[heads] = p, E[time] = $1/p$
  - (essentially the discrete version of an exponential random variable)
  - in population protocol, heads event = non-null interaction

both are *memoryless*: $\forall\, s,r > 0$

$$\Pr[\,\underbrace{\mathbf{T} > s + r}_{\text{no rxn after } r \text{ additional secs.}} \mid \underbrace{\mathbf{T} > s}_{\text{no rxn in first } s \text{ secs.}}\,] = \Pr[\,\underbrace{\mathbf{T} > r}_{\text{no rxn in first } r \text{ secs.}}\,]$$

- CRN model: time = sum of exponential random variables
  - define volume = n, and rate of interaction $a,b \to \ldots$ as $\#a \cdot \#b / n$, i.e., expected time $\boxed{n / (\#a \cdot \#b)}$
- PP model (time = #interactions / n)
  - probability that next interaction is $a,b \to \ldots$ is $\#a \cdot \#b$ / (n choose 2) = $2 \cdot \#a \cdot \#b / (n(n-1))$
  - expected interactions until next $a,b \to \ldots$ interaction = $n(n-1) / (2 \cdot \#a \cdot \#b)$, i.e., time $(n-1) / (2 \cdot \#a \cdot \#b)$
  - If we treat interactions symmetrically, (*i.e., $a,b \to c,d$ is an interaction if and only if $b,a \to d,c$ is an interaction*), then we have twice the probability, i.e., expected time becomes $(n-1) / \#a \cdot \#b \sim \boxed{n / (\#a \cdot \#b)}$, essentially the same as the CRN model

Discrete versus continuous time

- Time between interactions in CRN model is exponential random variable **T**
- Time between interactions in PP model is geometric random variable **T**
  - **T** = # of coin flips until a heads, with Pr[heads] = p, E[time] = $1/p$
  - (essentially the discrete version of an exponential random variable)
  - in population protocol, heads event = non-null interaction

both are *memoryless*: $\forall\ s,r > 0$

$$\Pr[\ \underbrace{\mathbf{T} > s + r}_{\text{no rxn after } r \text{ additional secs.}}\ |\ \underbrace{\mathbf{T} > s}_{\text{no rxn in first } s \text{ secs.}}\ ] = \Pr[\ \underbrace{\mathbf{T} > r}_{\text{no rxn in first } r \text{ secs.}}\ ]$$

- CRN model: time = sum of exponential random variables
  - define volume = n, and rate of interaction $a,b \rightarrow \ldots$ as $\#a \cdot \#b\ /\ n$, i.e., expected time $\boxed{n\ /\ (\#a \cdot \#b)}$
- PP model (time = #interactions / n)
  - probability that next interaction is $a,b \rightarrow \ldots$ is $\#a \cdot \#b\ /\ (n \text{ choose } 2) = 2 \cdot \#a \cdot \#b\ /\ (n(n-1))$
  - expected interactions until next $a,b \rightarrow \ldots$ interaction = $n(n-1)\ /\ (2 \cdot \#a \cdot \#b)$, i.e., time $(n-1)\ /\ (2 \cdot \#a \cdot \#b)$
  - If we treat interactions symmetrically, (*i.e., $a,b \rightarrow c,d$ is an interaction if and only if $b,a \rightarrow d,c$ is an interaction*), then we have twice the probability, i.e., expected time becomes $(n-1)\ /\ \#a \cdot \#b \sim \boxed{n\ /\ (\#a \cdot \#b)}$, essentially the same as the CRN model
  - one possible convention to avoid symmetric interactions is simply define time = $2 \cdot$#interactions/n

Discrete versus continuous time

- Time between interactions in CRN model is <u>exponential</u> random variable **T**
- Time between interactions in PP model is <u>geometric</u> random variable **T**
  - **T** = # of coin flips until a heads, with Pr[heads] = p, E[time] = $1/p$
  - (essentially the discrete version of an exponential random variable)
  - in population protocol, heads event = non-null interaction

both are *memoryless*: $\forall\, s,r > 0$

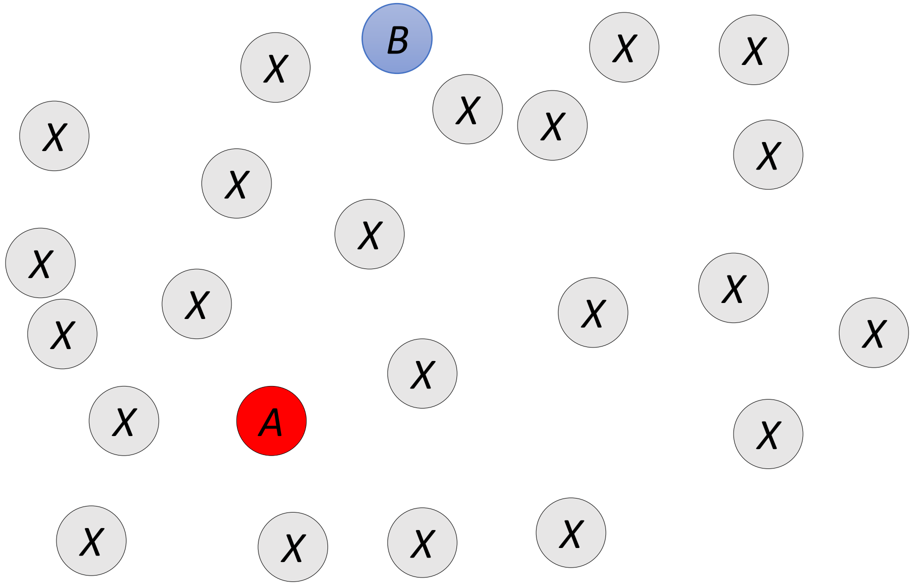
$$\Pr[\, \underbrace{\mathbf{T} > s + r}_{\text{no rxn after } r \text{ additional secs.}} \mid \underbrace{\mathbf{T} > s}_{\text{no rxn in first } s \text{ secs.}} \,] = \Pr[\, \underbrace{\mathbf{T} > r}_{\text{no rxn in first } r \text{ secs.}} \,]$$

- CRN model: time = sum of exponential random variables
  - define volume = n, and rate of interaction $a,b \rightarrow \ldots$ as $\#a \cdot \#b \,/\, n$, i.e., expected time $\boxed{n \,/\, (\#a \cdot \#b)}$
- PP model (time = #interactions / n)
  - probability that next interaction is $a,b \rightarrow \ldots$ is $\#a \cdot \#b \,/\, (n \text{ choose } 2) = 2 \cdot \#a \cdot \#b \,/\, (n(n-1))$
  - expected interactions until next $a,b \rightarrow \ldots$ interaction = $n(n-1) \,/\, (2 \cdot \#a \cdot \#b)$, i.e., time $(n-1) \,/\, (2 \cdot \#a \cdot \#b)$
  - If we treat interactions symmetrically, (*i.e., $a,b \rightarrow c,d$ is an interaction if and only if $b,a \rightarrow d,c$ is an interaction*), then we have twice the probability, i.e., expected time becomes $(n-1) \,/\, \#a \cdot \#b \sim \boxed{n \,/\, (\#a \cdot \#b)}$, essentially the same as the CRN model
  - one possible convention to avoid symmetric interactions is simply define time = $2 \cdot \#\text{interactions}/n$

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

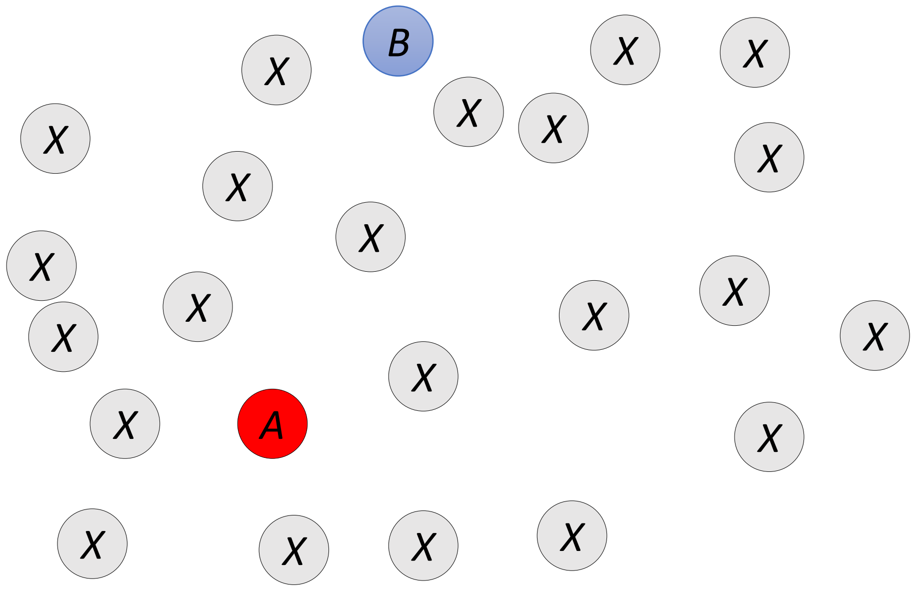
An exponential time difference

n molecules
volume $v = O(n)$



An exponential time difference

n molecules
volume $v = O(n)$



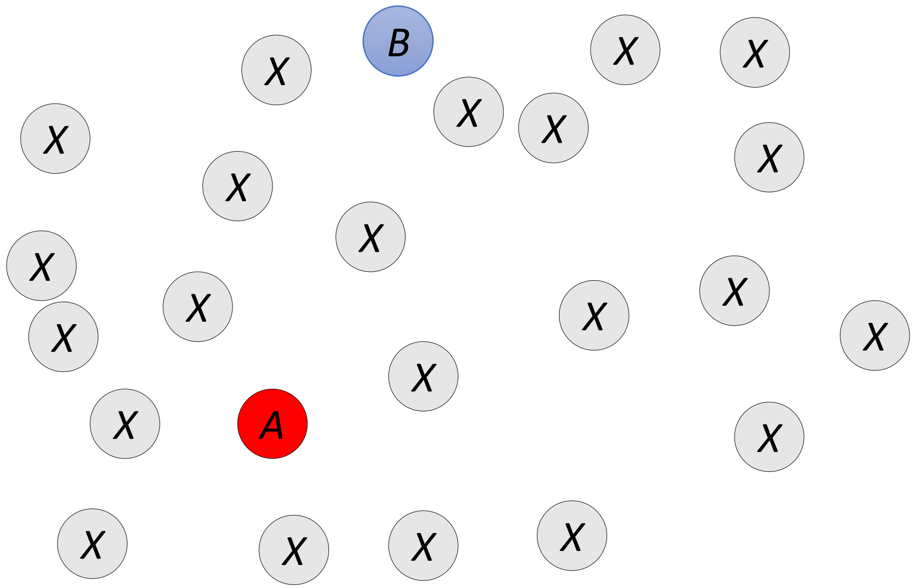
$$\boxed{A+B \rightarrow Y+B}$$

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

expected time to produce Y: $O(n)$

An exponential time difference

n molecules
volume $v = O(n)$



$$\boxed{A+B \rightarrow Y+B}$$

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

$$\boxed{\begin{array}{c} B+X \rightarrow B+B \\ A+B \rightarrow Y+B \end{array}}$$

distributed computing terms:

- *epidemic*
- *rumor/gossip spreading*

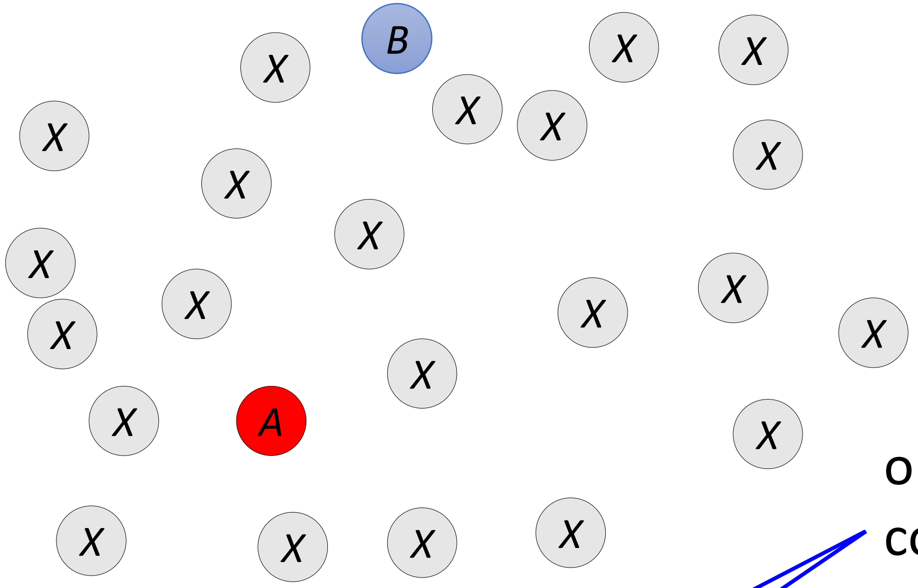
chemical term:

- *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 ≥ $n/2$

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

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

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

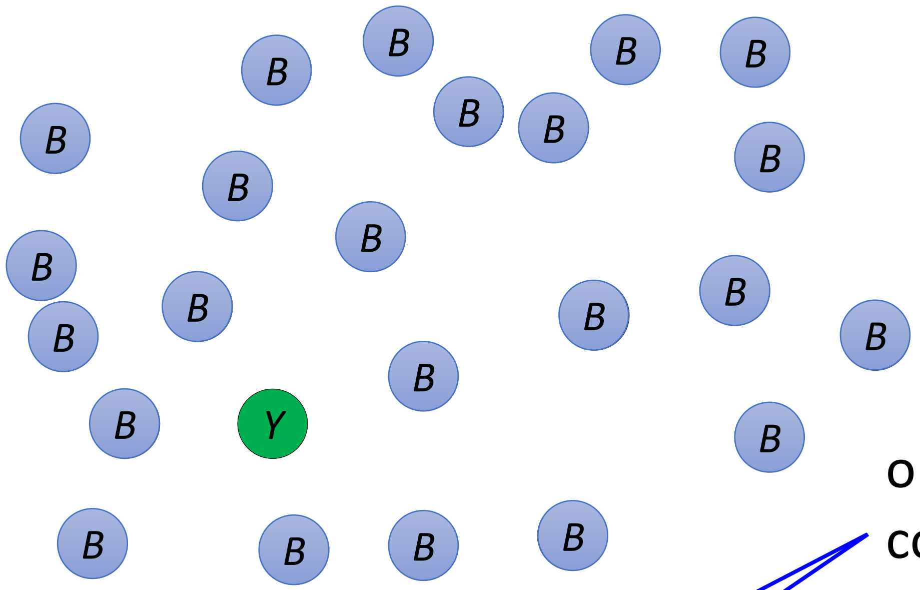
distributed computing terms:
- *epidemic*
- *rumor/gossip spreading*

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 $\geq n/2$

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

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

Time complexity analysis (*basic motifs*)

"direct communication"

$\boxed{A+B \rightarrow Y+W}$ $\#A=\#B=1,\ \#X=n\text{-}2$

Time complexity analysis (*basic motifs*)

"direct communication"

$\boxed{A+B \rightarrow Y+W}$ $\#A=\#B=1,\ \#X=n\text{-}2$

population protocol time complexity:
time until non-null interaction is geometric random variable with success probability $p = 1\ /\ (n \text{ choose } 2) = 2\ /\ (n(n–1))$

Time complexity analysis (*basic motifs*)

"direct communication"

$\boxed{A+B \rightarrow Y+W}$ $\#A=\#B=1, \ \#X=n\text{-}2$

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

Time complexity analysis (*basic motifs*)

"direct communication"

$\boxed{A + B \rightarrow Y + W}$ $\#A = \#B = 1$, $\#X = n\text{-}2$

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

Time complexity analysis (*basic motifs*)

"direct communication"

$\boxed{A+B \rightarrow Y+W}$ $\#A=\#B=1,\ \#X=n\text{-}2$

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

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

Time complexity analysis (*basic motifs*)

"direct communication"

$\boxed{A+B \rightarrow Y+W}$ $\#A=\#B=1,\ \#X=n\text{-}2$

population protocol time complexity:
time until non-null interaction is geometric random variable with success probability
p = 1 / (n choose 2) = 2 / (n(n–1))
E[# interactions] = 1/p = (n(n–1)) / 2
E[time] = E[# interactions]/n = (n–1) / 2 = O(n)

CRN time complexity:
time until reaction is exponential random variable with
rate $\lambda = \#A \cdot \#B / n = 1 / n$
E[time] = $1/\lambda = n$

"epidemic", "gossip", "rumor spreading"

$\boxed{B+X \rightarrow B+B}$

Time complexity analysis (*basic motifs*)

"direct communication"

$\boxed{A+B \rightarrow Y+W}$ $\#A=\#B=1,\ \#X=n\text{-}2$

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

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

"epidemic", "gossip", "rumor spreading"

$\boxed{B+X \rightarrow B+B}$

population protocol time complexity:

Time complexity analysis (*basic motifs*)

"direct communication"

$\boxed{A+B \rightarrow Y+W}$ $\#A=\#B=1,\ \#X=n\text{-}2$

population protocol time complexity:
time until non-null interaction is geometric random variable with success probability
p = 1 / (n choose 2) = 2 / (n(n–1))
E[# interactions] = 1/p = (n(n–1)) / 2
E[time] = E[# interactions]/n = (n–1) / 2 = O(n)

CRN time complexity:
time until reaction is exponential random variable with
rate $\lambda = \#A \cdot \#B / n = 1 / n$
E[time] = $1/\lambda = n$

"epidemic", "gossip", "rumor spreading"

$\boxed{B+X \rightarrow B+B}$

population protocol time complexity:
when $\#B = k$, we have $\#X = n–k$

Time complexity analysis (*basic motifs*)

"direct communication"

$\boxed{A+B \rightarrow Y+W}$ $\#A=\#B=1,\ \#X=n\text{-}2$

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

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

"epidemic", "gossip", "rumor spreading"

$\boxed{B+X \rightarrow B+B}$

population protocol time complexity:
when $\#B = k$, we have $\#X = n-k$
Pr[$B+X \rightarrow B+B$ is next interaction | $\#B=k$] = $k(n-k) / (n \text{ choose } 2)$

Time complexity analysis (*basic motifs*)

"direct communication"

$\boxed{A+B \to Y+W}$ $\#A=\#B=1, \ \#X=n\text{-}2$

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

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

"epidemic", "gossip", "rumor spreading"

$\boxed{B+X \to B+B}$

population protocol time complexity:
when $\#B = k$, we have $\#X = n–k$
$\Pr[\, B+X \to B+B \text{ is next interaction} \mid \#B=k \,] = k(n–k) / (n \text{ choose } 2)$
$= 2k(n–k) / ((n(n–1))$

Time complexity analysis (*basic motifs*)

"direct communication"

$A+B \to Y+W$ $\#A=\#B=1,\ \#X=n\text{-}2$

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

CRN time complexity:
time until reaction is exponential random variable with
rate $\lambda = \#A \cdot \#B / n = 1 / n$
E[time] $= 1/\lambda = n$

"epidemic", "gossip", "rumor spreading"

$B+X \to B+B$

population protocol time complexity:
when $\#B = k$, we have $\#X = n–k$
Pr[$B+X \to B+B$ is next interaction | $\#B=k$] $= k(n–k) / (n \text{ choose } 2)$
$= 2k(n–k) / ((n(n–1))$
expected time until <u>one</u> X converted to $B = 1/(n \cdot \text{probability})$
$= (n–1) / (2k(n–k))$

Time complexity analysis (*basic motifs*)

"direct communication"

$\boxed{A+B \rightarrow Y+W}$ $\#A=\#B=1,\ \#X=n\text{-}2$

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

CRN time complexity:
time until reaction is exponential random variable with
rate $\lambda = \#A \cdot \#B / n = 1 / n$
E[time] $= 1/\lambda = n$

"epidemic", "gossip", "rumor spreading"

$\boxed{B+X \rightarrow B+B}$

population protocol time complexity:
when $\#B = k$, we have $\#X = n–k$
Pr[$B+X \rightarrow B+B$ is next interaction | $\#B=k$] $= k(n–k) / (n \text{ choose } 2)$
$= 2k(n–k) / ((n(n–1))$
expected time until <u>one</u> X converted to $B = 1/(n\cdot\text{probability})$
$= (n–1) / (2k(n–k))$
expected time until <u>all</u> X converted to B =

Time complexity analysis (*basic motifs*)

"direct communication"

$\boxed{A+B \rightarrow Y+W}$ $\#A=\#B=1,\ \#X=n\text{-}2$

population protocol time complexity:
time until non-null interaction is geometric random variable with success probability
p = 1 / (n choose 2) = 2 / (n(n–1))
E[# interactions] = 1/p = (n(n–1)) / 2
E[time] = E[# interactions]/n = (n–1) / 2 = O(n)

CRN time complexity:
time until reaction is exponential random variable with
rate $\lambda = \#A \cdot \#B / n = 1 / n$
E[time] = $1/\lambda = n$

"epidemic", "gossip", "rumor spreading"

$\boxed{B+X \rightarrow B+B}$

population protocol time complexity:
when #B = k, we have #X = n–k
Pr[B+X→B+B is next interaction | #B=k] = k(n–k) / (n choose 2)
= 2k(n–k) / ((n(n–1))
expected time until one X converted to B = 1/(n·probability)
= (n–1) / (2k(n–k))
expected time until all X converted to B =

$$\frac{n-1}{2}\sum_{k=1}^{n-1}\frac{1}{k(n-k)}$$

Time complexity analysis (*basic motifs*)

"direct communication"

$\boxed{A+B \to Y+W}$ $\#A=\#B=1,\ \#X=n\text{-}2$

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

CRN time complexity:
time until reaction is exponential random variable with
rate $\lambda = \#A \cdot \#B / n = 1 / n$
E[time] = $1/\lambda = n$

"epidemic", "gossip", "rumor spreading"

$\boxed{B+X \to B+B}$

population protocol time complexity:
when $\#B = k$, we have $\#X = n–k$
Pr[$B+X \rightarrow B+B$ is next interaction | $\#B=k$] = $k(n–k) / (n \text{ choose } 2)$
= $2k(n–k) / ((n(n–1))$
expected time until <u>one</u> X converted to B = $1/(n \cdot \text{probability})$
= $(n–1) / (2k(n–k))$
expected time until <u>all</u> X converted to B =

$$\frac{n-1}{2}\sum_{k=1}^{n-1}\frac{1}{k(n-k)} = \frac{n-1}{2}\sum_{k=1}^{n-1}\frac{1}{n}\left(\frac{1}{k}+\frac{1}{n-k}\right)$$

Time complexity analysis (*basic motifs*)

"direct communication"

$\boxed{A+B \to Y+W}$ $\#A=\#B=1,\ \#X=n\text{-}2$

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

CRN time complexity:
time until reaction is exponential random variable with
rate $\lambda = \#A \cdot \#B / n = 1 / n$
E[time] $= 1/\lambda = n$

"epidemic", "gossip", "rumor spreading"

$\boxed{B+X \to B+B}$

population protocol time complexity:
when $\#B = k$, we have $\#X = n-k$
Pr[$B+X \rightarrow B+B$ is next interaction | $\#B=k$] $= k(n-k) / (n \text{ choose } 2)$
$= 2k(n-k) / ((n(n-1))$
expected time until <u>one</u> X converted to $B = 1/(n\cdot\text{probability})$
$= (n-1) / (2k(n-k))$
expected time until <u>all</u> X converted to B =

$$\frac{n-1}{2}\sum_{k=1}^{n-1}\frac{1}{k(n-k)} = \frac{n-1}{2}\sum_{k=1}^{n-1}\frac{1}{n}\left(\frac{1}{k}+\frac{1}{n-k}\right)$$

$$\approx \frac{1}{2}\left(\sum_{k=1}^{n}\frac{1}{k}+\sum_{k=n}^{1}\frac{1}{k}\right) = \sum_{k=1}^{n}\frac{1}{k}$$

Time complexity analysis (*basic motifs*)

"direct communication"

$\boxed{A+B \rightarrow Y+W}$ $\#A=\#B=1$, $\#X=n\text{-}2$

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

CRN time complexity:
time until reaction is exponential random variable with
rate $\lambda = \#A \cdot \#B / n = 1 / n$
E[time] $= 1/\lambda = n$

"epidemic", "gossip", "rumor spreading"

$\boxed{B+X \rightarrow B+B}$

population protocol time complexity:
when $\#B = k$, we have $\#X = n–k$
Pr[$B+X \rightarrow B+B$ is next interaction | $\#B=k$] $= k(n–k) / (n$ choose $2)$
$= 2k(n–k) / ((n(n–1))$
expected time until <u>one</u> X converted to $B = 1/(n\cdot$probability$)$
$= (n–1) / (2k(n–k))$
expected time until <u>all</u> X converted to B =

$$\frac{n-1}{2}\sum_{k=1}^{n-1}\frac{1}{k(n-k)} = \frac{n-1}{2}\sum_{k=1}^{n-1}\frac{1}{n}\left(\frac{1}{k}+\frac{1}{n-k}\right)$$

$$\approx \frac{1}{2}\left(\sum_{k=1}^{n}\frac{1}{k}+\sum_{k=n}^{1}\frac{1}{k}\right) = \sum_{k=1}^{n}\frac{1}{k} \approx \ln n$$

Time complexity analysis (*basic motifs*)

"no communication"
? here means "every species" (including A)

$\boxed{A+? \rightarrow B+?}$ $\#A=n$, $\#B=0$

Time complexity analysis (*basic motifs*)

"no communication"
? here means "every species" (including *A*)

$\boxed{A + ? \rightarrow B + ?}$ $\#A = n$, $\#B = 0$

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

Time complexity analysis (*basic motifs*)

"no communication"
? here means "every species" (including *A*)

$\boxed{A+? \rightarrow B+?}$ #A=n, #B=0

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

"no communication/ unimolecular decay"
(unimolecular CRN version)

$\boxed{A \rightarrow B}$ #A=n, #B=0

Time complexity analysis (*basic motifs*)

"no communication"
? here means "every species" (including *A*)

$\boxed{A + ? \rightarrow B + ?}$ #*A*=*n*, #*B*=0

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

"no communication/ unimolecular decay"
(unimolecular CRN version)

$\boxed{A \rightarrow B}$ #*A*=*n*, #*B*=0

CRN time complexity:

Time complexity analysis (*basic motifs*)

"no communication"
? here means "every species" (including A)

$\boxed{A + ? \rightarrow B + ?}$ #A=n, #B=0

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

"no communication/ unimolecular decay"
(unimolecular CRN version)

$\boxed{A \rightarrow B}$ #A=n, #B=0

CRN time complexity:
When #A=k, time until next reaction is exponential random variable with rate $\lambda = k$

Time complexity analysis (*basic motifs*)

"no communication"
? here means "every species" (including *A*)

$\boxed{A + ? \rightarrow B + ?}$ #*A*=*n*, #*B*=0

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

"no communication/ unimolecular decay"
(unimolecular CRN version)

$\boxed{A \rightarrow B}$ #*A*=*n*, #*B*=0

CRN time complexity:
When #*A*=*k*, time until next reaction is exponential random variable with rate λ = *k*
E[time until next reaction] = 1/λ = 1/*k*

Time complexity analysis (*basic motifs*)

"no communication"
? here means "every species" (including *A*)

$\boxed{A + ? \rightarrow B + ?}$ $\#A = n$, $\#B = 0$

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

"no communication/ unimolecular decay"
(unimolecular CRN version)

$\boxed{A \rightarrow B}$ $\#A = n$, $\#B = 0$

CRN time complexity:
When $\#A = k$, time until next reaction is exponential random variable with rate $\lambda = k$
E[time until next reaction] = $1/\lambda = 1/k$
E[time for all n reactions] = $\sum_{k=1}^{n}\frac{1}{k} \approx \ln n$

Time complexity analysis (*basic motifs*)

"pairing off"

$\boxed{A+B \rightarrow C}$ $\#A=n$, $\#B=n$, total volume = O(total count) = n

Time complexity analysis (*basic motifs*)

"pairing off"

$\boxed{A+B \rightarrow C}$

$\#A=n$, $\#B=n$, total volume = O(total count) = n

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

Time complexity analysis (*basic motifs*)

"pairing off"

$\boxed{A+B \rightarrow C}$

$\#A=n$, $\#B=n$, total volume = O(total count) = n

CRN time complexity:
When $\#A=\#B=k$, next reaction has rate $\lambda = k^2/n$
E[time until next reaction] = $1/\lambda = n/k^2$

Time complexity analysis (*basic motifs*)

"pairing off"

$\boxed{A+B \rightarrow C}$

$\#A=n$, $\#B=n$, total volume = O(total count) = n

CRN time complexity:
When $\#A=\#B=k$, next reaction has rate $\lambda = k^2/n$
E[time until next reaction] = $1/\lambda = n/k^2$
E[time for all n reactions] = $\sum_{k=1}^{n} \frac{n}{k^2}$

Time complexity analysis (*basic motifs*)

"pairing off"

$$\boxed{A + B \to C}$$

$\#A=n$, $\#B=n$, total volume = O(total count) = n

CRN time complexity:
When $\#A=\#B=k$, next reaction has rate $\lambda = k^2/n$
E[time until next reaction] = $1/\lambda = n/k^2$
E[time for all n reactions] = $\sum_{k=1}^{n} \frac{n}{k^2}$
$< n \sum_{k=1}^{\infty} \frac{1}{k^2}$

Time complexity analysis (*basic motifs*)

"pairing off"

$A+B \rightarrow C$

$\#A=n$, $\#B=n$, total volume = O(total count) = n

CRN time complexity:
When $\#A=\#B=k$, next reaction has rate $\lambda = k^2/n$
E[time until next reaction] = $1/\lambda = n/k^2$
E[time for all n reactions] = $\sum_{k=1}^{n} \frac{n}{k^2}$
$< n \sum_{k=1}^{\infty} \frac{1}{k^2}$
$= n \cdot \pi^2/6 = \Theta(n)$

Time complexity analysis (*basic motifs*)

"pairing off"

$\boxed{A+B \to C}$

$\#A=n$, $\#B=n$, total volume = O(total count) = n

CRN time complexity:
When $\#A=\#B=k$, next reaction has rate $\lambda = k^2/n$
E[time until next reaction] = $1/\lambda = n/k^2$
E[time for all n reactions] = $\sum_{k=1}^{n} \frac{n}{k^2}$
$< n \sum_{k=1}^{\infty} \frac{1}{k^2}$
$= n \cdot \pi^2/6 = \Theta(n)$

"pairing off" (symmetric version)

$\boxed{A+A \to C}$ similar analysis

Time complexity analysis (*basic motifs*)

"coupon collecting"

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

Time complexity analysis (*basic motifs*)

"coupon collecting"

$\boxed{L + A \to L + B}$ #L=1, #A=n, #B=0, total volume = O(total count) = n

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

Time complexity analysis of stably computing CRNs

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \to 2Y$

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \rightarrow 2Y$

$O(\log n)$ "*unimolecular decay*"

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \rightarrow 2Y$

$O(\log n)$ *"unimolecular decay"*

division by 2: $f(a) = a/2$
$2A \rightarrow Y$

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \rightarrow 2Y$

$O(\log n)$ *"unimolecular decay"*

division by 2: $f(a) = a/2$
$2A \rightarrow Y$

$O(n)$ *"pairing off"*

Time complexity analysis of stably computing CRNs

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

$A \rightarrow 2Y$

$O(\log n)$ *"unimolecular decay"*

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

$2A \rightarrow Y$

$O(n)$ *"pairing off"*

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

$A \rightarrow Y$

$B \rightarrow Y$

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \rightarrow 2Y$

$O(\log n)$ *"unimolecular decay"*

division by 2: $f(a) = a/2$
$2A \rightarrow Y$

$O(n)$ *"pairing off"*

addition: $f(a,b) = a+b$
$A \rightarrow Y$
$B \rightarrow Y$

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

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \to 2Y$

$O(\log n)$ *"unimolecular decay"*

minimum: $f(a,b) = \min(a,b)$
$A+B \to Y$

division by 2: $f(a) = a/2$
$2A \to Y$

$O(n)$ *"pairing off"*

addition: $f(a,b) = a+b$
$A \to Y$
$B \to Y$

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

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \rightarrow 2Y$

$O(\log n)$ *"unimolecular decay"*

division by 2: $f(a) = a/2$
$2A \rightarrow Y$

$O(n)$ *"pairing off"*

addition: $f(a,b) = a+b$
$A \rightarrow Y$
$B \rightarrow Y$

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

minimum: $f(a,b) = \min(a,b)$
$A+B \rightarrow Y$

$O(n)$: *"pairing off"*
… worst case if $a = b$

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \rightarrow 2Y$

$O(\log n)$ *"unimolecular decay"*

division by 2: $f(a) = a/2$
$2A \rightarrow Y$

$O(n)$ *"pairing off"*

addition: $f(a,b) = a+b$
$A \rightarrow Y$
$B \rightarrow Y$

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

minimum: $f(a,b) = \min(a,b)$
$A+B \rightarrow Y$

$O(n)$: *"pairing off"*
… worst case if $a = b$

Suppose $a > b$.

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \to 2Y$

$O(\log n)$ *"unimolecular decay"*

division by 2: $f(a) = a/2$
$2A \to Y$

$O(n)$ *"pairing off"*

addition: $f(a,b) = a+b$
$A \to Y$
$B \to Y$

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

minimum: $f(a,b) = \min(a,b)$
$A+B \to Y$

$O(n)$: *"pairing off"*
… worst case if $a = b$

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

Time complexity analysis of stably computing CRNs

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

$A \to 2Y$

$O(\log n)$ *"unimolecular decay"*

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

$2A \to Y$

$O(n)$ *"pairing off"*

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

$A \to Y$

$B \to Y$

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

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

$A+B \to Y$

$O(n)$: *"pairing off"*

… worst case if $a = b$

Suppose $a > b$.

E[time] =

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

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

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \rightarrow 2Y$

$O(\log n)$ *"unimolecular decay"*

division by 2: $f(a) = a/2$
$2A \rightarrow Y$

$O(n)$ *"pairing off"*

addition: $f(a,b) = a+b$
$A \rightarrow Y$
$B \rightarrow Y$

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

minimum: $f(a,b) = \min(a,b)$
$A+B \rightarrow Y$

$O(n)$: *"pairing off"*
… worst case if $a = b$

Suppose $a > b$.
E[time] =

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

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

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

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \to 2Y$

$O(\log n)$ *"unimolecular decay"*

division by 2: $f(a) = a/2$
$2A \to Y$

$O(n)$ *"pairing off"*

addition: $f(a,b) = a+b$
$A \to Y$
$B \to Y$

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

minimum: $f(a,b) = \min(a,b)$
$A+B \to Y$

$O(n)$: *"pairing off"*
… worst case if $a = b$

Suppose $a > b$.
E[time] =

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

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

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

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

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \to 2Y$

$O(\log n)$ *"unimolecular decay"*

division by 2: $f(a) = a/2$
$2A \to Y$

$O(n)$ *"pairing off"*

addition: $f(a,b) = a+b$
$A \to Y$
$B \to Y$

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

minimum: $f(a,b) = \min(a,b)$
$A+B \to Y$

$O(n)$: *"pairing off"*
… worst case if $a = b$

Suppose $a > b$.
E[time] =
$$\sum_{i=0}^{b-1} \frac{n}{(a-i)(b-i)}$$
$$= n \sum_{i=0}^{b-1} \frac{1}{(a-i)(b-i)}$$
$$< n \sum_{i=0}^{b-1} \frac{1}{(b-i)^2}$$
$$= n \sum_{i=1}^{b} \frac{1}{i^2}$$
$= O(n)$
So it's no slower… can it be *faster* in some cases?

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \to 2Y$

$O(\log n)$ *"unimolecular decay"*

division by 2: $f(a) = a/2$
$2A \to Y$

$O(n)$ *"pairing off"*

addition: $f(a,b) = a+b$
$A \to Y$
$B \to Y$

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

minimum: $f(a,b) = \min(a,b)$
$A+B \to Y$

$O(n)$: *"pairing off"*
… worst case if $a = b$

Suppose $a > b$.
E[time] =

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

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

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

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

$= O(n)$
So it's no slower… can it be *faster* in some cases?

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

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \to 2Y$

$O(\log n)$ "*unimolecular decay*"

division by 2: $f(a) = a/2$
$2A \to Y$

$O(n)$ "*pairing off*"

addition: $f(a,b) = a+b$
$A \to Y$
$B \to Y$

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

minimum: $f(a,b) = \min(a,b)$
$A+B \to Y$

$O(n)$: "*pairing off*"
… worst case if $a = b$

Suppose $a > b$.
E[time] =

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

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

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

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

$= O(n)$

So it's no slower… can it be *faster* in some cases?

Suppose $a > 2b$, so $a > 2n/3$.
E[time] =

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

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \rightarrow 2Y$

$O(\log n)$ *"unimolecular decay"*

division by 2: $f(a) = a/2$
$2A \rightarrow Y$

$O(n)$ *"pairing off"*

addition: $f(a,b) = a+b$
$A \rightarrow Y$
$B \rightarrow Y$

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

minimum: $f(a,b) = \min(a,b)$
$A+B \rightarrow Y$

$O(n)$: *"pairing off"*
… worst case if $a = b$

Suppose $a > b$.
E[time] =
$$\sum_{i=0}^{b-1} \frac{n}{(a-i)(b-i)}$$
$$= n \sum_{i=0}^{b-1} \frac{1}{(a-i)(b-i)}$$
$$< n \sum_{i=0}^{b-1} \frac{1}{(b-i)^2}$$
$$= n \sum_{i=1}^{b} \frac{1}{i^2}$$
$= O(n)$
So it's no slower… can it be *faster* in some cases?

Suppose $a > 2b$, so $a > 2n/3$.
E[time] =
$$n \sum_{i=0}^{b-1} \frac{1}{(a-i)(b-i)}$$
$$< n \sum_{i=0}^{b-1} \frac{1}{(a-b)(b-i)}$$

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \to 2Y$

$O(\log n)$ *"unimolecular decay"*

division by 2: $f(a) = a/2$
$2A \to Y$

$O(n)$ *"pairing off"*

addition: $f(a,b) = a+b$
$A \to Y$
$B \to Y$

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

minimum: $f(a,b) = \min(a,b)$
$A+B \to Y$

$O(n)$: *"pairing off"*
… worst case if $a = b$

Suppose $a > b$.
E[time] =
$\sum_{i=0}^{b-1} \frac{n}{(a-i)(b-i)}$
$= n \sum_{i=0}^{b-1} \frac{1}{(a-i)(b-i)}$
$< n \sum_{i=0}^{b-1} \frac{1}{(b-i)^2}$
$= n \sum_{i=1}^{b} \frac{1}{i^2}$
$= O(n)$
So it's no slower… can it be *faster* in some cases?

Suppose $a > 2b$, so $a > 2n/3$.
E[time] =
$n \sum_{i=0}^{b-1} \frac{1}{(a-i)(b-i)}$
$< n \sum_{i=0}^{b-1} \frac{1}{(a-b)(b-i)}$
$< n \sum_{i=0}^{b-1} \frac{1}{(a/2)(b-i)}$

Time complexity analysis of stably computing CRNs

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

$A \to 2Y$

$O(\log n)$ *"unimolecular decay"*

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

$2A \to Y$

$O(n)$ *"pairing off"*

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

$A \to Y$

$B \to Y$

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

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

$A+B \to Y$

$O(n)$: *"pairing off"*

… worst case if $a = b$

Suppose $a > b$.

E[time] =

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

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

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

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

$= O(n)$

So it's no slower… can it be *faster* in some cases?

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

E[time] =

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

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

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

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

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \to 2Y$

$O(\log n)$ *"unimolecular decay"*

division by 2: $f(a) = a/2$
$2A \to Y$

$O(n)$ *"pairing off"*

addition: $f(a,b) = a+b$
$A \to Y$
$B \to Y$

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

minimum: $f(a,b) = \min(a,b)$
$A+B \to Y$

$O(n)$: *"pairing off"*
… worst case if $a = b$

Suppose $a > b$.
E[time] =

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

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

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

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

$= O(n)$

So it's no slower… can it be *faster* in some cases?

Suppose $a > 2b$, so $a > 2n/3$.
E[time] =

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

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

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

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

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

Time complexity analysis of stably computing CRNs

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

$A \to 2Y$

$O(\log n)$ *"unimolecular decay"*

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

$2A \to Y$

$O(n)$ *"pairing off"*

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

$A \to Y$

$B \to Y$

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

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

$A+B \to Y$

$O(n)$: *"pairing off"*

… worst case if $a = b$

Suppose $a > b$.

E[time] =

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

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

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

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

$= O(n)$

So it's no slower… can it be *faster* in some cases?

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

E[time] =

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

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

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

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

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

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

Time complexity analysis of stably computing CRNs

multiplication by 2: $f(a) = 2a$
$A \to 2Y$

$O(\log n)$ *"unimolecular decay"*

division by 2: $f(a) = a/2$
$2A \to Y$

$O(n)$ *"pairing off"*

addition: $f(a,b) = a+b$
$A \to Y$
$B \to Y$

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

minimum: $f(a,b) = \min(a,b)$
$A+B \to Y$

$O(n)$: *"pairing off"*
… worst case if $a = b$

Suppose $a > b$.
E[time] =

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

So it's no slower… can it be *faster* in some cases?

Suppose $a > 2b$, so $a > 2n/3$.
E[time] =

$$n \sum_{i=0}^{b-1} \frac{1}{(a-i)(b-i)}$$
$$< n \sum_{i=0}^{b-1} \frac{1}{(a-b)(b-i)}$$
$$< n \sum_{i=0}^{b-1} \frac{1}{(a/2)(b-i)}$$
$$= \frac{2n}{a} \sum_{i=0}^{b-1} \frac{1}{(b-i)}$$
$$= \frac{2n}{a} \sum_{i=1}^{b} \frac{1}{i} \approx \frac{2n}{a} \ln b$$
$$\leq \frac{2n}{\frac{2}{3}n} \ln b = 3 \ln b$$

Intuitively, there's always a large $\Omega(n)$ excess of A, so "acts like" unimolecular decay of B.

Time complexity analysis of stably computing CRNs

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

$A \to Y$

$B + Y \to \emptyset$

Time complexity analysis of stably computing CRNs

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

$A \rightarrow Y$

$B+Y \rightarrow \emptyset$

- Unlike addition, this is a nontrivial combination of reactions: rate of second reaction depends how many times first has happened.

Time complexity analysis of stably computing CRNs

subtraction: $f(a,b) = a-b$
$A \rightarrow Y$
$B+Y \rightarrow \emptyset$

- Unlike addition, this is a nontrivial combination of reactions: rate of second reaction depends how many times first has happened.
- To simplify, we assume second reaction cannot happen until first has finished.

Time complexity analysis of stably computing CRNs

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

$A \to Y$

$B+Y \to \emptyset$

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

Time complexity analysis of stably computing CRNs

subtraction: $f(a,b) = a-b$
$A \rightarrow Y$
$B+Y \rightarrow \emptyset$

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

E[time] = E[time for first to finish] + E[time for second to finish]

Time complexity analysis of stably computing CRNs

subtraction: $f(a,b) = a - b$
$A \to Y$
$B + Y \to \emptyset$

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

E[time] = E[time for first to finish] + E[time for second to finish]

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

Time complexity analysis of stably computing CRNs

subtraction: $f(a,b) = a-b$
$A \rightarrow Y$
$B+Y \rightarrow \emptyset$

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

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

Time complexity analysis of stably computing CRNs

subtraction: $f(a,b) = a-b$
$A \rightarrow Y$
$B+Y \rightarrow \emptyset$

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

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

Time complexity analysis of stably computing CRNs

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

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

Time complexity analysis of stably computing CRNs

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

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

- Assume reaction 3 waits for reactions 1 and 2 before starting, and reaction 4 waits for reaction 3.

Time complexity analysis of stably computing CRNs

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

1. $A \to Y+A_2$
2. $B \to Y+B_2$
3. $A_2+B_2 \to K$
4. $K+Y \to \emptyset$

- Assume reaction 3 waits for reactions 1 and 2 before starting, and reaction 4 waits for reaction 3.
- E[time for 1 and 2] = $O(\log n)$

Time complexity analysis of stably computing CRNs

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

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

- Assume reaction 3 waits for reactions 1 and 2 before starting, and reaction 4 waits for reaction 3.
- E[time for 1 and 2] = $O(\log n)$
- E[time for 3] = $O(n)$

Time complexity analysis of stably computing CRNs

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

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

- Assume reaction 3 waits for reactions 1 and 2 before starting, and reaction 4 waits for reaction 3.
- E[time for 1 and 2] = $O(\log n)$
- E[time for 3] = $O(n)$
- E[time for 4] = $O(n)$

Time complexity analysis of stably computing CRNs

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

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

- Assume reaction 3 waits for reactions 1 and 2 before starting, and reaction 4 waits for reaction 3.
- E[time for 1 and 2] = $O(\log n)$
- E[time for 3] = $O(n)$
- E[time for 4] = $O(n)$
- So E[time] = $O(\log n) + O(n) + O(n) = O(n)$

Possibilities of stable computation

What can be stably computed?

Summary: Possibilities and limits of stable computation

Summary: Possibilities and limits of stable computation

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]

Summary: Possibilities and limits of stable computation

Predicates

- φ is stably computable if and only if φ is *semilinear*.
- semilinear = Boolean combination of threshold and mod predicates: take weighted sum $s = w_1 \cdot \mathbf{a}_1 + \ldots\ w_d \cdot \mathbf{a}_d$ of inputs and ask if

  $s > t$? (threshold)

  $s \equiv c \bmod m$? (mod)

[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]

Summary: Possibilities and limits of stable computation

Predicates

- φ is stably computable if and only if φ is *semilinear*.
- semilinear = Boolean combination of threshold and mod predicates: take weighted sum $s = w_1 \cdot \mathbf{a}_1 + \dots w_d \cdot \mathbf{a}_d$ of inputs and ask if

  $s > t$? (threshold)

  $s \equiv c \bmod m$? (mod)

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

[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]

Summary: Possibilities and limits of stable computation

Predicates

- φ is stably computable if and only if φ is *semilinear*.
- semilinear = Boolean combination of <u>threshold</u> and <u>mod</u> predicates: take weighted sum $s = w_1 \cdot \mathbf{a}_1 + \ldots w_d \cdot \mathbf{a}_d$ of inputs and ask if

  $s > t$? (threshold)

  $s \equiv c \bmod m$? (mod)

$a>b$?	$a=b$?	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]

Summary: Possibilities and limits of stable computation

Predicates

- φ is stably computable if and only if φ is *semilinear*.
- semilinear = Boolean combination of threshold and mod predicates: take weighted sum $s = w_1 \cdot \mathbf{a}_1 + \ldots w_d \cdot \mathbf{a}_d$ of inputs and ask if

  $s > t$? (threshold)

  $s \equiv c \bmod m$? (mod)

$a>b$? $a=b$? 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) = $\{ (\boldsymbol{a},y) \mid f(\boldsymbol{a})=y \}$ is semilinear.

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

Summary: Possibilities and limits of stable computation

Predicates

- φ is stably computable if and only if φ is *semilinear*.
- semilinear = Boolean combination of <u>threshold</u> and <u>mod</u> predicates: take weighted sum $s = w_1 \cdot \mathbf{a}_1 + \ldots w_d \cdot \mathbf{a}_d$ of inputs and ask if

  $s > t$? (threshold)

  $s \equiv c \bmod m$? (mod)

$a>b$? $a=b$? 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) = $\{ (\boldsymbol{a},y) \mid f(\boldsymbol{a})=y \}$ is semilinear.
- <u>piecewise affine</u>, with semilinear predicate to determine which piece.

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

Summary: Possibilities and limits of stable computation

Predicates

- φ is stably computable if and only if φ is *semilinear*.
- semilinear = Boolean combination of threshold and mod predicates: take weighted sum $s = w_1 \cdot \mathbf{a}_1 + \ldots\ w_d \cdot \mathbf{a}_d$ of inputs and ask if

  $s > t$? (threshold)

  $s \equiv c \bmod m$? (mod)

$a>b$? $a=b$? 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) = $\{ (\boldsymbol{a},y) \mid f(\boldsymbol{a})=y \}$ is semilinear.
- piecewise affine, 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 $f(a) = a/4+5b$

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

Summary: Possibilities and limits of stable computation

Predicates

- φ is stably computable if and only if φ is *semilinear*.
- semilinear = Boolean combination of threshold and mod predicates: take weighted sum $s = w_1 \cdot \mathbf{a}_1 + \ldots w_d \cdot \mathbf{a}_d$ of inputs and ask if
  - $s > t$? (threshold)
  - $s \equiv c \bmod m$? (mod)

$a>b$? $a=b$? 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) = $\{ (\boldsymbol{a},y) \mid f(\boldsymbol{a})=y \}$ is semilinear.
- piecewise affine, 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]

Summary: Possibilities and limits of stable computation

Predicates

- φ is stably computable if and only if φ is *semilinear*.
- semilinear = Boolean combination of threshold and mod predicates: take weighted sum $s = w_1 \cdot \mathbf{a}_1 + \ldots w_d \cdot \mathbf{a}_d$ of inputs and ask if
  - $s > t$? (threshold)
  - $s \equiv c \bmod m$? (mod)

$a>b$? $a=b$? 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 affine, 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]

Linear sets

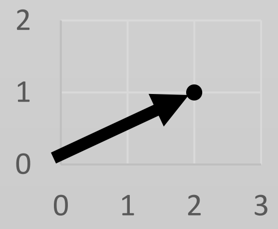
Definition: A set $X \subseteq \mathbb{N}^d$ is <u>linear</u> if there are vectors $\mathbf{b}, \mathbf{u}_1, \ldots, \mathbf{u}_p \in \mathbb{N}^d$ such that
$X = \{\, \mathbf{b} + n_1 \cdot \mathbf{u}_1 + \ldots + n_p \cdot \mathbf{u}_p \mid n_1, \ldots, n_p \in \mathbb{N} \,\}$

multi-dimensional generalization of *eventually periodic*

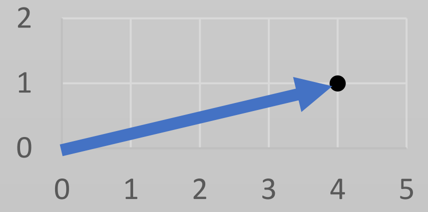
Linear sets

Example in dimension *d*=2:

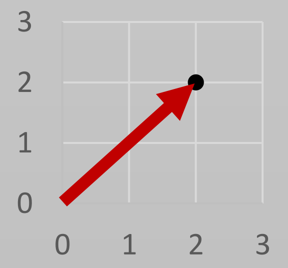
$\mathbf{b} = (2,1)$



$\mathbf{u}_1 = (4,1)$



$\mathbf{u}_2 = (2,2)$



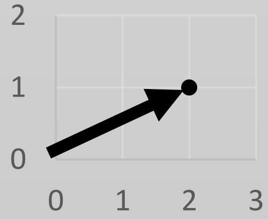
Definition: A set $X \subseteq \mathbb{N}^d$ is <u>linear</u> if there are vectors $\mathbf{b}, \mathbf{u}_1, \ldots, \mathbf{u}_p \in \mathbb{N}^d$ such that
$X = \{\, \mathbf{b} + n_1 \cdot \mathbf{u}_1 + \ldots + n_p \cdot \mathbf{u}_p \mid n_1, \ldots, n_p \in \mathbb{N} \,\}$

multi-dimensional generalization of *eventually periodic*

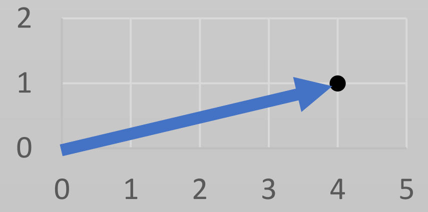
Linear sets

Example in dimension d=2:

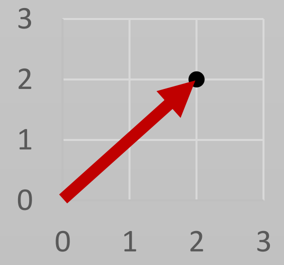
$\mathbf{b} = (2,1)$



$\mathbf{u}_1 = (4,1)$

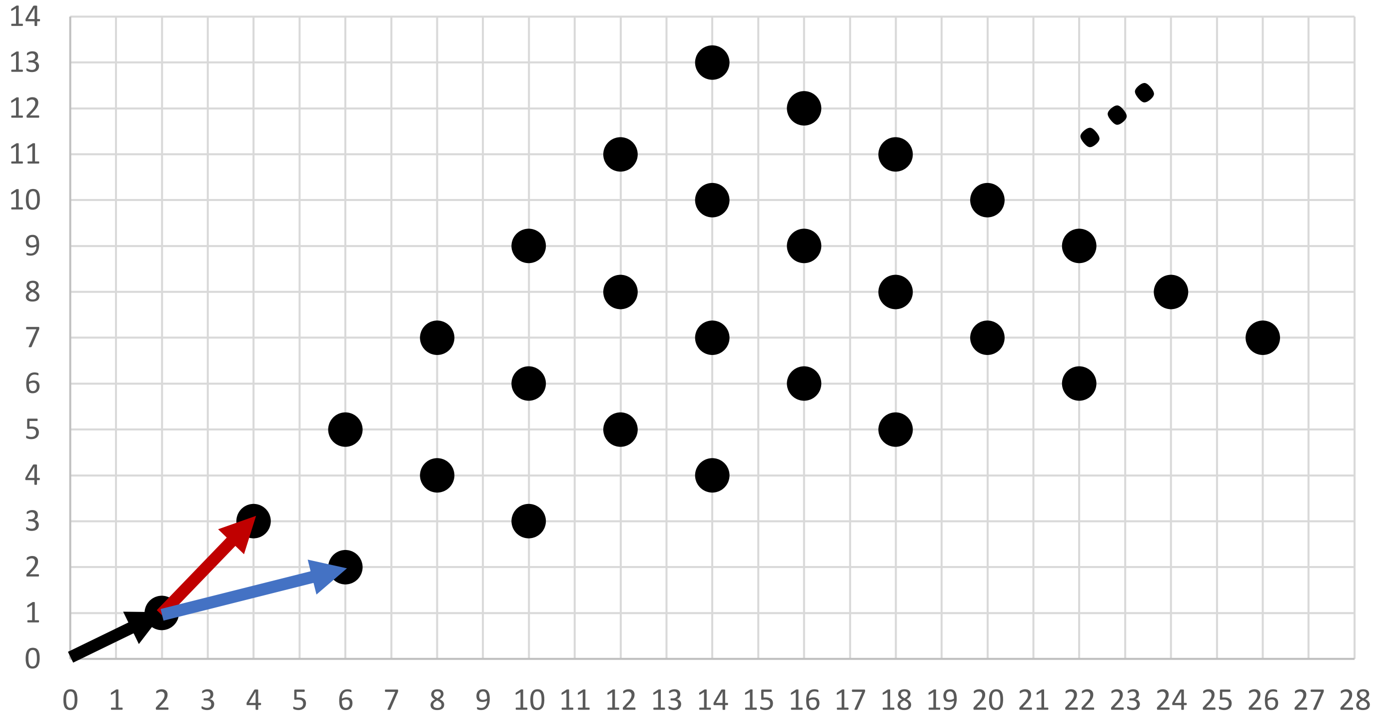


$\mathbf{u}_2 = (2,2)$



Definition: A set $X \subseteq \mathbb{N}^d$ is <u>linear</u> if there are vectors $\mathbf{b}, \mathbf{u}_1, \ldots, \mathbf{u}_p \in \mathbb{N}^d$ such that
$X = \{ \mathbf{b} + n_1 \cdot \mathbf{u}_1 + \ldots + n_p \cdot \mathbf{u}_p \mid n_1, \ldots, n_p \in \mathbb{N} \}$

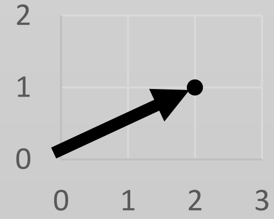
multi-dimensional generalization of *eventually periodic*



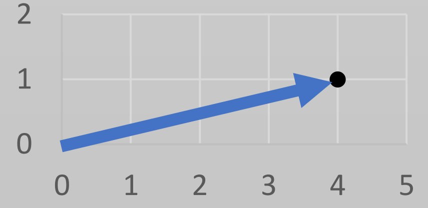
Linear sets

Example in dimension *d*=2:

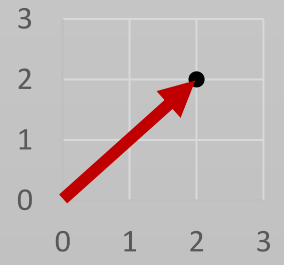
$\mathbf{b} = (2,1)$



$\mathbf{u}_1 = (4,1)$

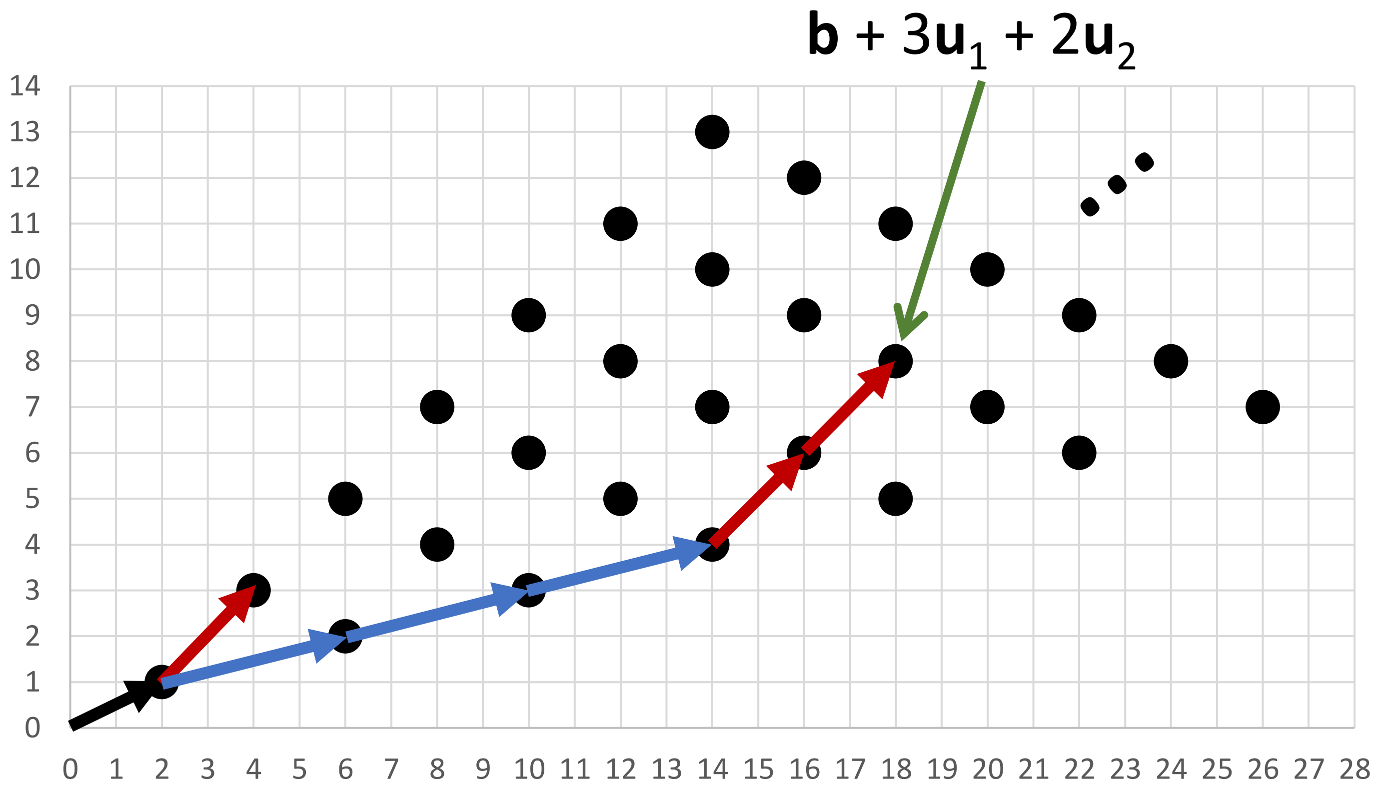


$\mathbf{u}_2 = (2,2)$



Definition: A set $X \subseteq \mathbb{N}^d$ is <u>linear</u> if there are vectors $\mathbf{b}, \mathbf{u}_1, \ldots, \mathbf{u}_p \in \mathbb{N}^d$ such that
$X = \{\ \mathbf{b} + n_1 \cdot \mathbf{u}_1 + \ldots + n_p \cdot \mathbf{u}_p \mid n_1, \ldots, n_p \in \mathbb{N}\ \}$

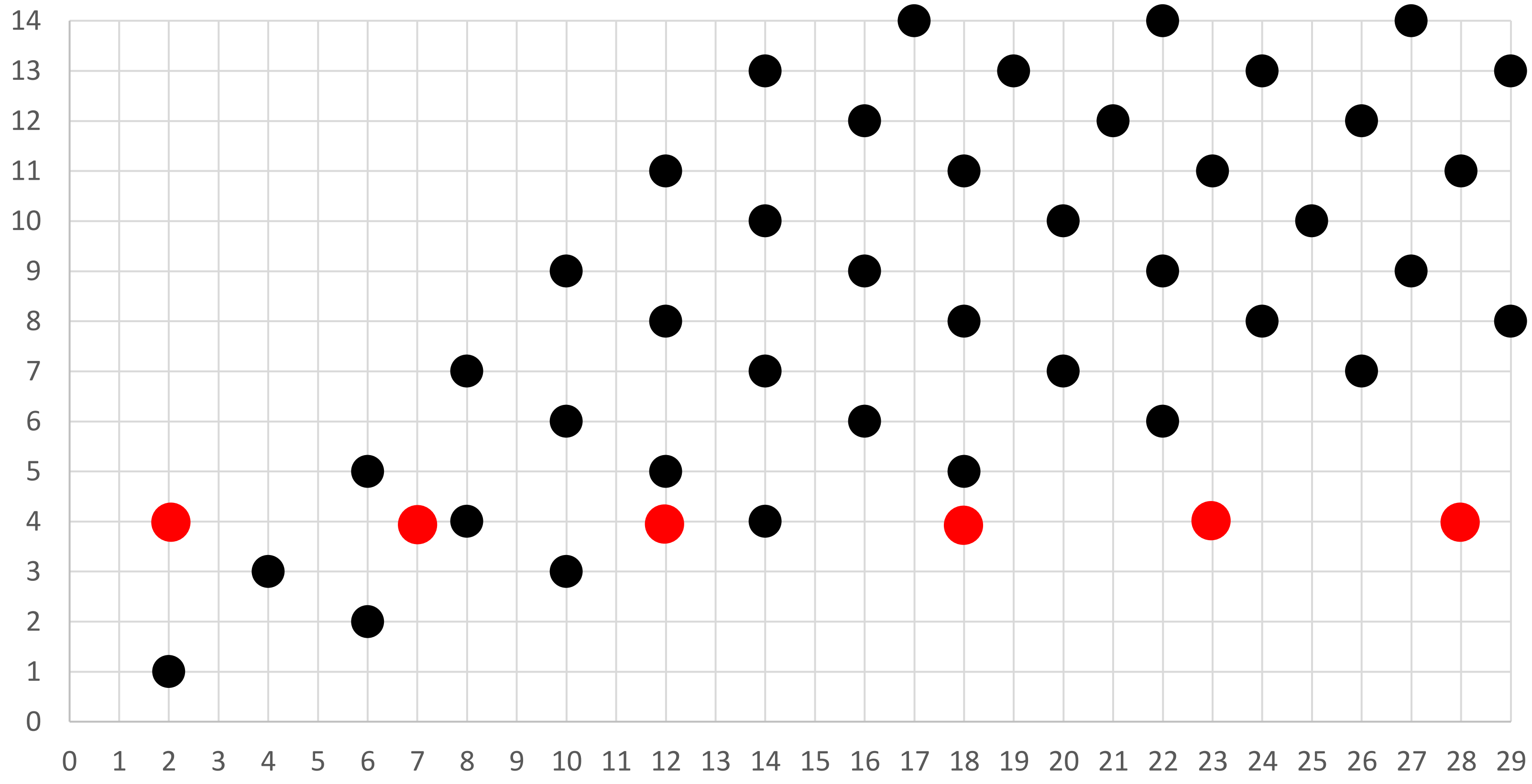
multi-dimensional generalization of *eventually periodic*



Semilinear sets

Definition: A set $X \subseteq \mathbb{N}^d$ is <u>semilinear</u> if it is a finite union of linear sets.

union of two linear sets:



Equivalent definitions of semilinear

Definition 2: A set $X \subseteq \mathbb{N}^d$ is semilinear if it is a finite union of linear sets.

Equivalent definitions of semilinear

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

Definition 2: A set $X \subseteq \mathbb{N}^d$ is semilinear if it is a finite union of linear sets.

Equivalent definitions of semilinear

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

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

Definition 2: A set $X \subseteq \mathbb{N}^d$ is semilinear if it is a finite union of linear sets.

Equivalent definitions of semilinear

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

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

examples:

Definition 2: A set $X \subseteq \mathbb{N}^d$ is semilinear if it is a finite union of linear sets.

Equivalent definitions of semilinear

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

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

examples:
is $x_1 > x_2$?

Definition 2: A set $X \subseteq \mathbb{N}^d$ is semilinear if it is a finite union of linear sets.

Equivalent definitions of semilinear

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

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

examples:
is $x_1 > x_2$?
is $x_1 - 3x_2 > x_2 + 5$?

Definition 2: A set $X \subseteq \mathbb{N}^d$ is <u>semilinear</u> if it is a finite union of linear sets.

Equivalent definitions of semilinear

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

Definition 2: A set $X \subseteq \mathbb{N}^d$ is semilinear if it is a finite union of linear sets.

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

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

examples:
is $x_1 > x_2$?
is $x_1 - 3x_2 > x_2 + 5$?

Equivalent definitions of semilinear

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

Definition 2: A set $X \subseteq \mathbb{N}^d$ is semilinear if it is a finite union of linear sets.

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

examples:
is $x_1 > x_2$?
is $x_1 - 3x_2 > x_2 + 5$?

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

examples:

Equivalent definitions of semilinear

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

Definition 2: A set $X \subseteq \mathbb{N}^d$ is semilinear if it is a finite union of linear sets.

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

examples:
is $x_1 > x_2$?
is $x_1 - 3x_2 > x_2 + 5$?

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

examples:
is x odd?

Equivalent definitions of semilinear

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

Definition 2: A set $X \subseteq \mathbb{N}^d$ is semilinear if it is a finite union of linear sets.

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

examples:
is $x_1 > x_2$?
is $x_1 - 3x_2 > x_2 + 5$?

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

examples:
is x odd?
is x 2 more than a multiple of 3? = $\{2, 5, 8, 11, 14, \dots\}$

Equivalent definitions of semilinear

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

Definition 2: A set $X \subseteq \mathbb{N}^d$ is semilinear if it is a finite union of linear sets.

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

examples:
is $x_1 > x_2$?
is $x_1 - 3x_2 > x_2 + 5$?

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

examples:
is x odd?
is x 2 more than a multiple of 3? = {2, 5, 8, 11, 14, …}
is $x_1 - 3x_2$ odd?

Equivalent definitions of semilinear

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

Definition 2: A set $X \subseteq \mathbb{N}^d$ is semilinear if it is a finite union of linear sets.

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

examples:
is $x_1 > x_2$?
is $x_1 - 3x_2 > x_2 + 5$?

example semilinear set:
is $x_1 > x_2$ and $x_1 + x_2$ is odd?

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

examples:
is x odd?
is x 2 more than a multiple of 3? = {2, 5, 8, 11, 14, …}
is $x_1 - 3x_2$ odd?

Equivalent definitions of semilinear

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

Definition 2: A set $X \subseteq \mathbb{N}^d$ is <u>semilinear</u> if it is a finite union of linear sets.

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

examples:
is $x_1 > x_2$?
is $x_1 - 3x_2 > x_2 + 5$?

example semilinear set:
is $x_1 > x_2$ and $x_1 + x_2$ is odd?

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

examples:
is x odd?
is x 2 more than a multiple of 3? = {2, 5, 8, 11, 14, …}
is $x_1 - 3x_2$ odd?

example semilinear set:
is $x_1 + x_2$ is <u>not</u> a multiple of 3?

Equivalent definitions of semilinear

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

Equivalent definitions of semilinear

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

Other places semilinear sets show up in computer science:

- Sets decidable by *reversal-bounded counter machines*.
- In 2D, they are conjectured to be the sets weakly self-assembled by temperature τ=1 tile systems.

Limits of stable computation

Theorem 1: A set $X \subseteq \mathbb{N}^d$ is stably decided by some CRN if and only if it is semilinear.

Limits of stable computation

Theorem 1: A set $X \subseteq \mathbb{N}^d$ is stably decided by some CRN if and only if it is semilinear.

Full proof is too complex to do in this course. But we'll show:

1. All semilinear sets <u>can</u> be stably decided.
2. The non-semilinear "squaring" set X = $\{ (a,y) \in \mathbb{N}^2 \mid a^2 = y \}$ <u>cannot</u> be stably decided.

Limits of stable computation

Theorem 1: A set $X \subseteq \mathbb{N}^d$ is stably decided by some CRN if and only if it is semilinear.

Full proof is too complex to do in this course. But we'll show:

1. All semilinear sets <u>can</u> be stably decided.
2. The non-semilinear "squaring" set $X = \{ (a,y) \in \mathbb{N}^2 \mid a^2 = y \}$ <u>cannot</u> be stably decided.

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

Limits of stable computation

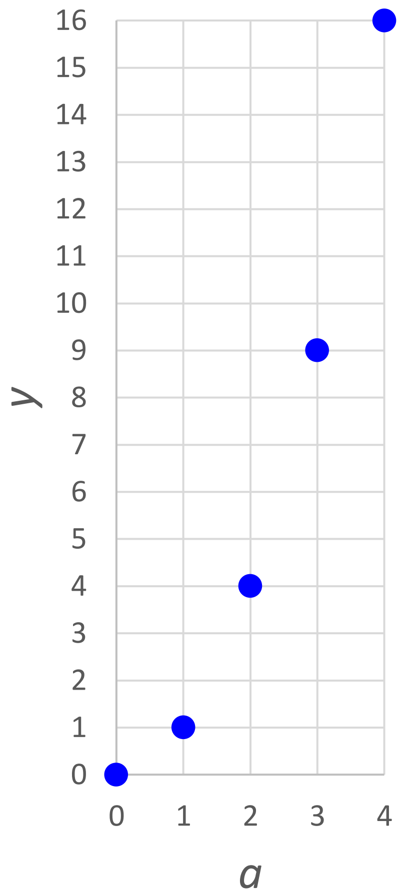
Theorem 1: A set $X \subseteq \mathbb{N}^d$ is stably decided by some CRN if and only if it is semilinear.

Full proof is too complex to do in this course. But we'll show:

1. All semilinear sets can be stably decided.
2. The non-semilinear "squaring" set $X = \{ (a,y) \in \mathbb{N}^2 \mid a^2 = y \}$ cannot be stably decided.

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

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



X = graph(f), where $f(a) = a^2$

Limits of stable computation

Theorem 1: A set $X \subseteq \mathbb{N}^d$ is stably decided by some CRN if and only if it is semilinear.

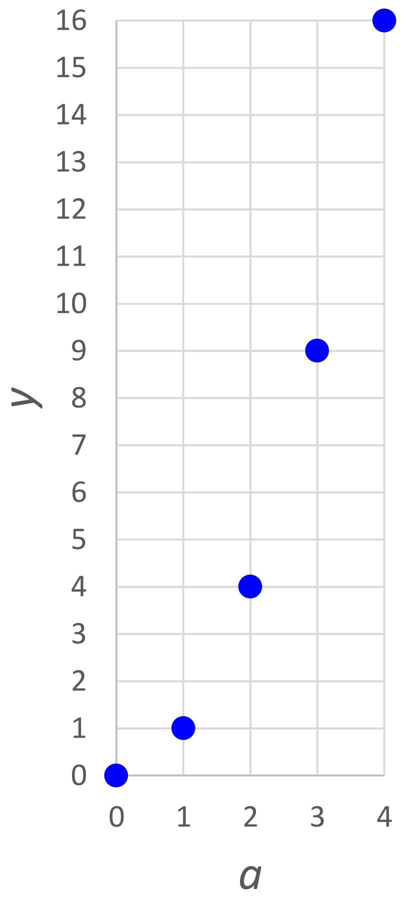
Full proof is too complex to do in this course. But we'll show:

1. All semilinear sets <u>can</u> be stably decided.
2. The non-semilinear "squaring" set $X = \{ (a,y) \in \mathbb{N}^2 \mid a^2 = y \}$ <u>cannot</u> be stably decided.

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

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

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



X = graph(f), where $f(a) = a^2$

Possibilities of stable computation

All semilinear functions/predicates can be stably computed by CRNs

Stably decidable sets are closed under Boolean operations

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

Stably decidable sets are closed under Boolean operations

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

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

Proof:

Stably decidable sets are closed under Boolean operations

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

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

Proof:

1. To stably decide $\overline{X_1}$, swap the yes and no voters.

Stably decidable sets are closed under Boolean operations

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

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

Proof:

1. To stably decide $\overline{X_1}$, swap the yes and no voters.
2. For ∪ and ∩, let C_1 and C_2 stably decide X_1 and X_2.

Stably decidable sets are closed under Boolean operations

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

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

Proof:

1. To stably decide $\overline{X_1}$, swap the yes and no voters.
2. For $\cup$ and $\cap$, let C_1 and C_2 stably decide X_1 and X_2.
3. Add the reaction $A \rightarrow A_1 + A_2$ for each input species A, and let A_i be the input species for C_i.

Stably decidable sets are closed under Boolean operations

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

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

Proof:
1. To stably decide $\overline{X_1}$, swap the yes and no voters.
2. For ∪ and ∩, let C_1 and C_2 stably decide X_1 and X_2.
3. Add the reaction $A \rightarrow A_1 + A_2$ for each input species A, and let A_i be the input species for C_i.
4. Add four new species V_{NN}, V_{NY}, V_{YN}, and V_{YY}.

Stably decidable sets are closed under Boolean operations

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

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

Proof:

1. To stably decide $\overline{X_1}$, swap the yes and no voters.
2. For ∪ and ∩, let C_1 and C_2 stably decide X_1 and X_2.
3. Add the reaction $A \rightarrow A_1 + A_2$ for each input species A, and let A_i be the input species for C_i.
4. Add four new species V_{NN}, V_{NY}, V_{YN}, and V_{YY}.
5. To "record" the votes of C_1 and C_2:

Stably decidable sets are closed under Boolean operations

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

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

Proof:

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

Stably decidable sets are closed under Boolean operations

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

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

Proof:

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

Stably decidable sets are closed under Boolean operations

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

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

Proof:

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

Stably decidable sets are closed under Boolean operations

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

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

Proof:

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

Stably decidable sets are closed under Boolean operations

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

What if all species are required to vote??

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

Proof:

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

Mod and threshold sets are stably decidable

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

Mod and threshold sets are stably decidable

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

Proof:

1. Start with 1 L_0 leader.
   The leader will "*count the (weighted) input mod m*."

Mod and threshold sets are stably decidable

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

Proof:

1. Start with 1 L_0 leader.
   The leader will "*count the (weighted) input mod m*."
2. For each $1 \leq i \leq d$ and $0 \leq j < m$, add the reaction $X_i + L_j \rightarrow L_{j+wi \bmod m}$

Mod and threshold sets are stably decidable

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

Proof:

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

Mod and threshold sets are stably decidable

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

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

Proof:

1. Start with 1 L_0 leader.
   The leader will "*count the (weighted) input mod m*."
2. For each $1 \le i \le d$ and $0 \le j < m$, add the reaction $X_i + L_j \rightarrow L_{j+wi \bmod m}$
3. Let L_c vote yes and all others vote no.

Mod and threshold sets are stably decidable

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

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

Proof:

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

Proof:

1. If $w_i > 0$, add reaction $X_i \rightarrow w_i\, P$

Mod and threshold sets are stably decidable

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

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

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

Proof:
1. If $w_i > 0$, add reaction $X_i \to w_i$ P
2. If $w_i < 0$, add reaction $X_i \to (-w_i)$ N

Mod and threshold sets are stably decidable

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

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

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

Proof:
1. If $w_i > 0$, add reaction $X_i \rightarrow w_i\ P$
2. If $w_i < 0$, add reaction $X_i \rightarrow (-w_i)\ N$
3. Need to decide if (#P produced) > (#N produced) + t

Mod and threshold sets are stably decidable

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

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

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

Proof:
1. If $w_i > 0$, add reaction $X_i \rightarrow w_i\ P$
2. If $w_i < 0$, add reaction $X_i \rightarrow (-w_i)\ N$
3. Need to decide if (#P produced) > (#N produced) + t
4. Start with 1 L_N leader and

Mod and threshold sets are stably decidable

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

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

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

Proof:
1. If $w_i > 0$, add reaction $X_i \to w_i\, P$
2. If $w_i < 0$, add reaction $X_i \to (-w_i)\, N$
3. Need to decide if (#P produced) > (#N produced) + t
4. Start with 1 L_N leader and
   1. t N if $t > 0$.
   2. $(-t)$ P if $t < 0$.

Mod and threshold sets are stably decidable

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

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

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

Proof:
1. If $w_i > 0$, add reaction $X_i \rightarrow w_i\ P$
2. If $w_i < 0$, add reaction $X_i \rightarrow (-w_i)\ N$
3. Need to decide if (#P produced) > (#N produced) + t
4. Start with 1 L_N leader and
   1. t N if $t > 0$.
   2. $(-t)$ P if $t < 0$.
5. Now need to decide if #P > #N (*including those present initially*)

Mod and threshold sets are stably decidable

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

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

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

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

Mod and threshold sets are stably decidable

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

Proof:

1. Start with 1 L_0 leader.
   The leader will "*count the (weighted) input mod m*."
2. For each $1 \le i \le d$ and $0 \le j < m$, add the reaction $X_i + L_j \to L_{j+wi \bmod m}$
3. Let L_c vote yes and all others vote no.

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

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

Proof:

1. If $w_i > 0$, add reaction $X_i \to w_i\, P$
2. If $w_i < 0$, add reaction $X_i \to (-w_i)\, N$
3. Need to decide if (#P produced) > (#N produced) + t
4. Start with 1 L_N leader and
   1. t N if $t > 0$.
   2. $(-t)$ P if $t < 0$.
5. Now need to decide if #P > #N (*including those present initially*)
6. Add reactions
   1. $L_Y + N \to L_N$
   2. $L_N + P \to L_Y$

Also true for leaderless CRNs.

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

Semilinear functions are stably computable

Lemma: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then it is piecewise affine: a finite union of partial affine functions g_i: $\mathbb{N}^d \dashrightarrow \mathbb{N}$.

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

Furthermore, each "piece" dom g_i is a linear set.

We won't prove this; see [Chen, Doty, Soloveichik, *Deterministic function computation with chemical reaction networks*. DNA 2012]

Semilinear functions are stably computable

Lemma: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then it is piecewise affine: a finite union of partial affine functions g_i: $\mathbb{N}^d \dashrightarrow \mathbb{N}$.

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

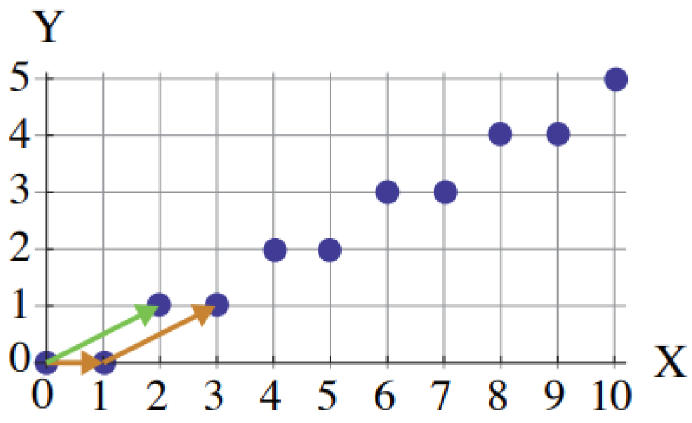
Furthermore, each "piece" dom g_i is a linear set.

$f(x) = \lfloor x/2 \rfloor$

start with: (input) X

output: Y

$X + X \to Y$



$\{n_1 \cdot (2,1) \mid n_1 \in \mathbb{N}\} \cup$

$\{(1,0) + n_1 \cdot (2,1) \mid n_1 \in \mathbb{N}\}$

We won't prove this; see [Chen, Doty, Soloveichik, *Deterministic function computation with chemical reaction networks*. DNA 2012]

Semilinear functions are stably computable

Lemma: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then it is piecewise affine: a finite union of partial affine functions g_i: $\mathbb{N}^d \dashrightarrow \mathbb{N}$.

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

Furthermore, each "piece" dom g_i is a linear set.

$f(x) = \lfloor x/2 \rfloor$

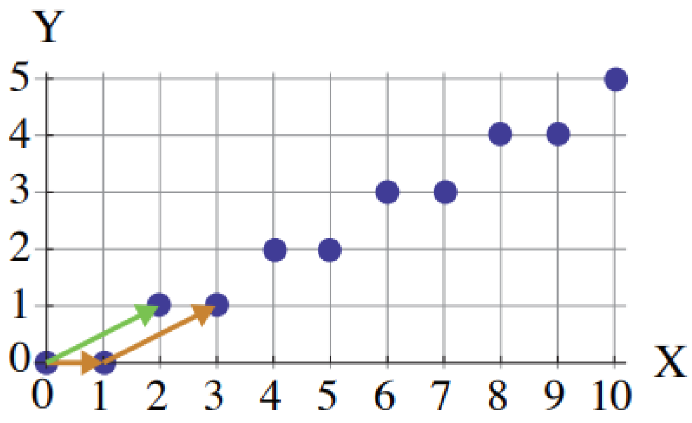
start with: (input) X

output: Y

$g_1(x) = ½ \cdot x$

$g_2(x) = ½ \cdot (x - 1)$

$X + X \to Y$



$\{n_1 \cdot (2, 1) \mid n_1 \in \mathbb{N}\} \cup$

$\{(1, 0) + n_1 \cdot (2, 1) \mid n_1 \in \mathbb{N}\}$

We won't prove this; see [Chen, Doty, Soloveichik, *Deterministic function computation with chemical reaction networks*. DNA 2012]

Semilinear functions are stably computable

Lemma: If $f: \mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then it is piecewise affine: a finite union of partial affine functions $g_i: \mathbb{N}^d \dashrightarrow \mathbb{N}$.

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

Furthermore, each "piece" dom g_i is a linear set.

We won't prove this; see [Chen, Doty, Soloveichik, *Deterministic function computation with chemical reaction networks*. DNA 2012]

$f(x) = \lfloor x/2 \rfloor$

start with: (input) X

output: Y

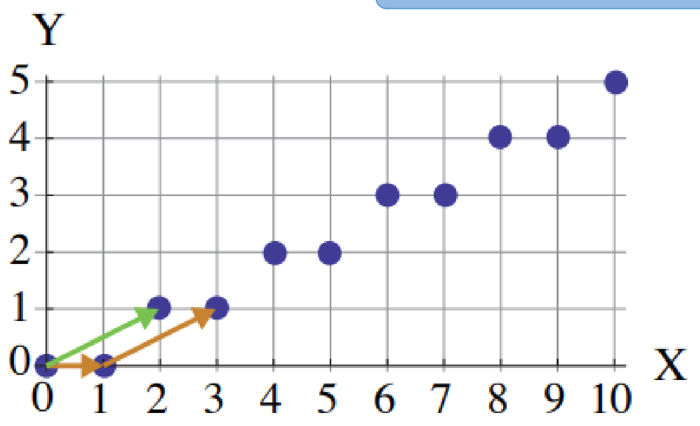
$g_1(x) = ½ \cdot x$

$g_2(x) = ½ \cdot (x-1)$

$X + X \to Y$

dom $g_1 = \{x \equiv 0 \bmod 2\}$

dom $g_2 = \{x \equiv 1 \bmod 2\}$



$\{n_1 \cdot (2,1) \mid n_1 \in \mathbb{N}\} \cup$

$\{(1,0) + n_1 \cdot (2,1) \mid n_1 \in \mathbb{N}\}$

Semilinear function examples

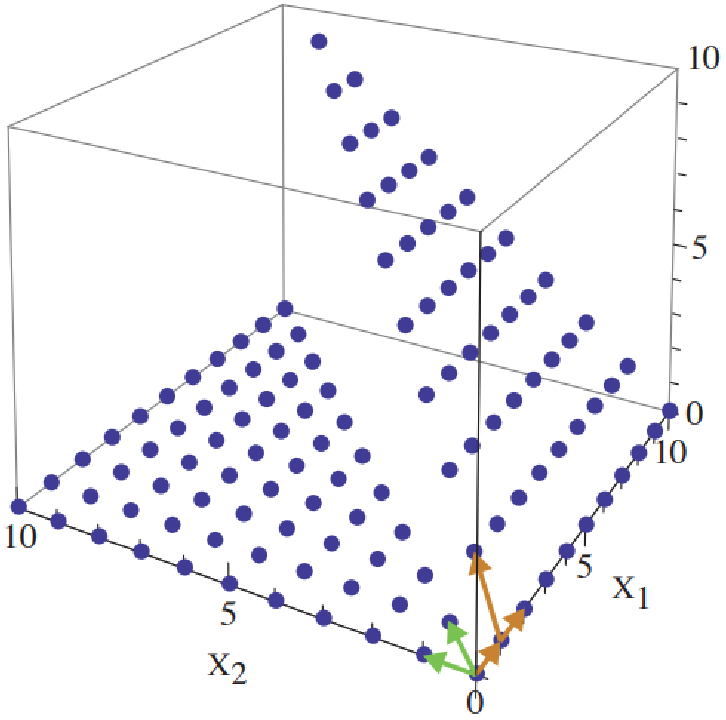
$g_1(x) = x_1$
$g_2(x) = x_2$
dom $g_1 = \{x_1 > x_2\}$

b) $f(x_1, x_2) = \begin{cases} x_2 & \text{if } x_1 > x_2 \\ 0 & \text{otherwise} \end{cases}$

start with: (input) X_1, X_2

output: Y

$X_1 + X_2 \rightarrow B$
$X_1 + B \rightarrow X_1 + Y$
$B + Y \rightarrow B + B$



$\{n_1 \cdot (1,1,0) + n_2 \cdot (0,1,0) \mid n_1, n_2 \in \mathbb{N}\} \cup$
$\{(1,0,0) + n_1 \cdot (1,1,1) + n_2 \cdot (1,0,0) \mid n_1, n_2 \in \mathbb{N}\}$

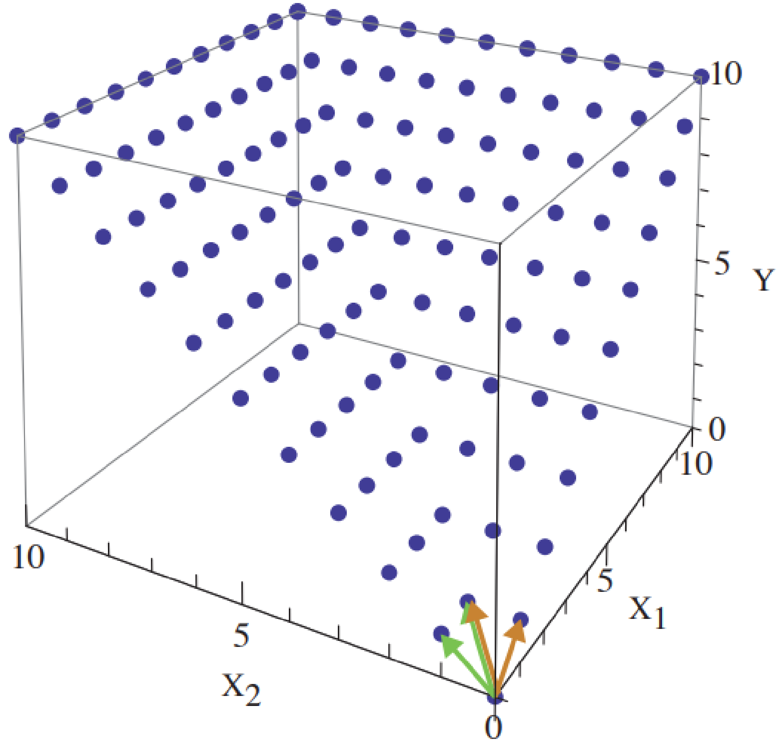
$g_1(x) = x_2$
$g_2(x) = 0$
dom $g_1 = \{x_1 > x_2\}$

c) $f(x_1, x_2) = \max(x_1, x_2)$

start with: (input) X_1, X_2

output: Y

$X_1 \rightarrow Z_1 + Y$
$X_2 \rightarrow Z_2 + Y$
$Z_1 + Z_2 \rightarrow K$
$K + Y \rightarrow Ø$



$\{n_1 \cdot (1,1,1) + n_2 \cdot (1,1,0) \mid n_1, n_2 \in \mathbb{N}\} \cup$
$\{n_1 \cdot (1,1,1) + n_2 \cdot (1,0,1) \mid n_1, n_2 \in \mathbb{N}\}$

Computing affine functions (*by example*)

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

Computing affine functions (*by example*)

linear:
$f(a,b,c) = 2a + (4/3)b - (5/6)c$

General form: $w_1 \dots w_d \in \mathbb{Q}$ and $b, c_1, \dots, c_d \in \mathbb{N}$
$g_i(x_1, \dots, x_d) = w_1 \cdot (x_1 - c_1) + \dots + w_d \cdot (x_d - c_d) + b.$

Computing affine functions (*by example*)

linear:

$f(a,b,c) = 2a + (4/3)b - (5/6)c$

$A \rightarrow 2Y$

General form: $w_1 \ldots w_d \in \mathbb{Q}$ and $b, c_1, \ldots, c_d \in \mathbb{N}$

$g_i(x_1, \ldots, x_d) = w_1 \cdot (x_1 - c_1) + \ldots + w_d \cdot (x_d - c_d) + b.$

Computing affine functions (*by example*)

linear:

$f(a,b,c) = 2a + (4/3)b - (5/6)c$

$A \to 2Y$

$3B \to 4Y$

General form: $w_1 \ldots w_d \in \mathbb{Q}$ and $b, c_1, \ldots, c_d \in \mathbb{N}$

$g_i(x_1, \ldots, x_d) = w_1 \cdot (x_1 - c_1) + \ldots + w_d \cdot (x_d - c_d) + b.$

Computing affine functions (*by example*)

<u>linear</u>:
$f(a,b,c) = 2a + (4/3)b - (5/6)c$

$A \rightarrow 2Y$

$3B \rightarrow 4Y$

$6C + 5Y \rightarrow \emptyset$

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

Computing affine functions (*by example*)

linear:

$f(a,b,c) = 2a + (4/3)b - (5/6)c$

$A \rightarrow 2Y$

$3B \rightarrow 4Y$

$6C+5Y \rightarrow \emptyset$

add constant offset:

start with 1 L, a A's, b B's

$f(a,b) = 2a + 3b + 4$

General form: $w_1 \ldots w_d \in \mathbb{Q}$ and $b, c_1, \ldots, c_d \in \mathbb{N}$

$g_i(x_1, \ldots, x_d) = w_1 \cdot (x_1 - c_1) + \ldots + w_d \cdot (x_d - c_d) + b.$

Computing affine functions (*by example*)

<u>linear</u>:
$f(a,b,c) = 2a + (4/3)b - (5/6)c$

$A \rightarrow 2Y$

$3B \rightarrow 4Y$

$6C + 5Y \rightarrow \emptyset$

<u>add constant offset</u>:
start with 1 L, a A's, b B's
$f(a,b) = 2a + 3b + 4$

$L \rightarrow 4Y$

General form: $w_1 \dots w_d \in \mathbb{Q}$ and $b, c_1, \dots, c_d \in \mathbb{N}$
$g_i(x_1, \dots, x_d) = w_1 \cdot (x_1 - c_1) + \dots + w_d \cdot (x_d - c_d) + b$.

Computing affine functions (*by example*)

<u>linear</u>:
$f(a,b,c) = 2a + (4/3)b - (5/6)c$

$A \rightarrow 2Y$

$3B \rightarrow 4Y$

$6C + 5Y \rightarrow \emptyset$

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

<u>add constant offset</u>:
start with 1 L, a A's, b B's
$f(a,b) = 2a + 3b + 4$

$L \rightarrow 4Y$

$A \rightarrow 2Y$

Computing affine functions (*by example*)

<u>linear</u>:
$f(a,b,c) = 2a + (4/3)b - (5/6)c$

$A \to 2Y$

$3B \to 4Y$

$6C + 5Y \to \emptyset$

<u>add constant offset</u>:
start with 1 L, a A's, b B's
$f(a,b) = 2a + 3b + 4$

$L \to 4Y$

$A \to 2Y$

$B \to 3Y$

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

Computing affine functions (*by example*)

linear:

$f(a,b,c) = 2a + (4/3)b - (5/6)c$

$A \rightarrow 2Y$

$3B \rightarrow 4Y$

$6C + 5Y \rightarrow \emptyset$

add constant offset:

start with 1 L, a A's, b B's

$f(a,b) = 2a + 3b + 4$

$L \rightarrow 4Y$

$A \rightarrow 2Y$

$B \rightarrow 3Y$

General form: $w_1 \ldots w_d \in \mathbb{Q}$ and $b, c_1, \ldots, c_d \in \mathbb{N}$

$g_i(x_1, \ldots, x_d) = w_1 \cdot (x_1 - c_1) + \ldots + w_d \cdot (x_d - c_d) + b.$

subtract constant offset c_i from input x_i:

start with 1 L, a A's, b B's

$f(a,b) = 2(a-3) - (5/4)(b-1) + 6$

Computing affine functions (*by example*)

__linear__:
$f(a,b,c) = 2a + (4/3)b - (5/6)c$
$A \to 2Y$
$3B \to 4Y$
$6C + 5Y \to \emptyset$

__add constant offset__:
start with 1 L, a A's, b B's
$f(a,b) = 2a + 3b + 4$
$L \to 4Y$
$A \to 2Y$
$B \to 3Y$

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

__subtract constant offset c_i from input x_i__:
start with 1 L, a A's, b B's
$f(a,b) = 2(a-3) - (5/4)(b-1) + 6$
$L \to 6Y + L_{a0} + L_{b0}$ *create d offset, __and__ one leader for each input*

Computing affine functions (*by example*)

<u>linear</u>:
$f(a,b,c) = 2a + (4/3)b - (5/6)c$

$A \to 2Y$

$3B \to 4Y$

$6C + 5Y \to \emptyset$

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

<u>add constant offset</u>:
start with 1 L, a A's, b B's
$f(a,b) = 2a + 3b + 4$

$L \to 4Y$

$A \to 2Y$

$B \to 3Y$

<u>subtract constant offset c_i from input x_i</u>:
start with 1 L, a A's, b B's
$f(a,b) = 2(a-3) - (5/4)(b-1) + 6$

$L \to 6Y + L_{a0} + L_{b0}$ *create d offset, <u>and</u> one leader for each input*

$L_{a0} + A \to L_{a1}$ *remove 3 copies of A*

$L_{a1} + A \to L_{a2}$

$L_{a2} + A \to L_{a3}$

Computing affine functions (*by example*)

linear:

$f(a,b,c) = 2a + (4/3)b - (5/6)c$

$A \to 2Y$

$3B \to 4Y$

$6C + 5Y \to \emptyset$

General form: $w_1 \ldots w_d \in \mathbb{Q}$ and $b, c_1, \ldots, c_d \in \mathbb{N}$

$g_i(x_1, \ldots, x_d) = w_1 \cdot (x_1 - c_1) + \ldots + w_d \cdot (x_d - c_d) + b.$

add constant offset:

start with 1 L, a A's, b B's

$f(a,b) = 2a + 3b + 4$

$L \to 4Y$

$A \to 2Y$

$B \to 3Y$

subtract constant offset c_i from input x_i:

start with 1 L, a A's, b B's

$f(a,b) = 2(a-3) - (5/4)(b-1) + 6$

$L \to 6Y + L_{a0} + L_{b0}$ *create d offset, and one leader for each input*

$L_{a0} + A \to L_{a1}$ *remove 3 copies of A*

$L_{a1} + A \to L_{a2}$

$L_{a2} + A \to L_{a3}$

$L_{a3} + A \to L_{a3} + A'$ *convert remaining A to A'*

Computing affine functions (*by example*)

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

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

<u>add constant offset</u>:
start with 1 L, a A's, b B's
$f(a,b) = 2a + 3b + 4$
$L \rightarrow 4Y$
$A \rightarrow 2Y$
$B \rightarrow 3Y$

<u>subtract constant offset c_i from input x_i</u>:
start with 1 L, a A's, b B's
$f(a,b) = 2(a-3) - (5/4)(b-1) + 6$

$L \rightarrow 6Y + L_{a0} + L_{b0}$	*create d offset, <u>and</u> one leader for each input*
$L_{a0} + A \rightarrow L_{a1}$	*remove 3 copies of A*
$L_{a1} + A \rightarrow L_{a2}$	
$L_{a2} + A \rightarrow L_{a3}$	
$L_{a3} + A \rightarrow L_{a3} + A'$	*convert remaining A to A'*
$A' \rightarrow 2Y$	*compute 2(a–3) by doubling A'*

Computing affine functions (*by example*)

linear:

$f(a,b,c) = 2a + (4/3)b - (5/6)c$

$A \to 2Y$

$3B \to 4Y$

$6C + 5Y \to \emptyset$

add constant offset:

start with 1 L, a A's, b B's

$f(a,b) = 2a + 3b + 4$

$L \to 4Y$

$A \to 2Y$

$B \to 3Y$

General form: $w_1 \ldots w_d \in \mathbb{Q}$ and $b, c_1, \ldots, c_d \in \mathbb{N}$

$g_i(x_1, \ldots, x_d) = w_1 \cdot (x_1 - c_1) + \ldots + w_d \cdot (x_d - c_d) + b.$

subtract constant offset c_i from input x_i:

start with 1 L, a A's, b B's

$f(a,b) = 2(a-3) - (5/4)(b-1) + 6$

$L \to 6Y + L_{a0} + L_{b0}$	*create d offset, and one leader for each input*
$L_{a0} + A \to L_{a1}$	*remove 3 copies of A*
$L_{a1} + A \to L_{a2}$	
$L_{a2} + A \to L_{a3}$	
$L_{a3} + A \to L_{a3} + A'$	*convert remaining A to A'*
$A' \to 2Y$	*compute 2(a–3) by doubling A'*
$L_{b0} + B \to L_{b1}$	*remove 1 copy of B*
$L_{b1} + B \to L_{b1} + B'$	*convert remaining B to B'*

Computing affine functions (*by example*)

<u>linear</u>:

$f(a,b,c) = 2a + (4/3)b - (5/6)c$

$A \to 2Y$

$3B \to 4Y$

$6C + 5Y \to \emptyset$

General form: $w_1 \ldots w_d \in \mathbb{Q}$ and $b, c_1, \ldots, c_d \in \mathbb{N}$

$g_i(x_1, \ldots, x_d) = w_1 \cdot (x_1 - c_1) + \ldots + w_d \cdot (x_d - c_d) + b.$

<u>add constant offset</u>:

start with 1 L, a A's, b B's

$f(a,b) = 2a + 3b + 4$

$L \to 4Y$

$A \to 2Y$

$B \to 3Y$

<u>subtract constant offset c_i from input x_i</u>:

start with 1 L, a A's, b B's

$f(a,b) = 2(a-3) - (5/4)(b-1) + 6$

$L \to 6Y + L_{a0} + L_{b0}$	*create d offset, <u>and</u> one leader for each input*
$L_{a0} + A \to L_{a1}$	*remove 3 copies of A*
$L_{a1} + A \to L_{a2}$	
$L_{a2} + A \to L_{a3}$	
$L_{a3} + A \to L_{a3} + A'$	*convert remaining A to A'*
$A' \to 2Y$	*compute $2(a-3)$ by doubling A'*
$L_{b0} + B \to L_{b1}$	*remove 1 copy of B*
$L_{b1} + B \to L_{b1} + B'$	*convert remaining B to B'*
$4B' + 5Y \to \emptyset$	*compute $(-5/4)(b-1)$ on B'*

Combining all affine function computations

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

$\Leftarrow$

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

Furthermore, each "piece" dom g_i is a linear set.

Combining all affine function computations

Lemma: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then it is <u>piecewise affine</u>: a finite union of partial affine functions g_i: $\mathbb{N}^d \dashrightarrow \mathbb{N}$.

Furthermore, each "piece" dom g_i is a linear set.

$\Leftarrow$

Theorem: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then some CRN stably computes f.

Proof sketch:

1. To compute whole semilinear function f, compute all these affine functions g_i in parallel, storing output of g_i in species Y_i.

Combining all affine function computations

Lemma: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then it is piecewise affine: a finite union of partial affine functions g_i: $\mathbb{N}^d \dashrightarrow \mathbb{N}$.

Furthermore, each "piece" dom g_i is a linear set.

$\Leftarrow$

Theorem: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then some CRN stably computes f.

Proof sketch:

1. To compute whole semilinear function f, compute all these affine functions g_i in parallel, storing output of g_i in species Y_i.
   - *in parallel* means: split each input species A via reaction $A \to A_1 + A_2 + \ldots$, where A_i is used as input for computing g_i.

Combining all affine function computations

Lemma: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then it is <u>piecewise affine</u>: a finite union of partial affine functions g_i: $\mathbb{N}^d \dashrightarrow \mathbb{N}$.

Furthermore, each "piece" dom g_i is a linear set.

$\Leftarrow$

Theorem: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then some CRN stably computes f.

Proof sketch:

1. To compute whole semilinear function f, compute all these affine functions g_i in parallel, storing output of g_i in species Y_i.
   - *in parallel* means: split each input species A via reaction $A \to A_1 + A_2 + \ldots$, where A_i is used as input for computing g_i.
2. Also in parallel, for each domain dom g_i, compute the predicate [$\mathbf{x} \in$ dom g_i?].

Combining all affine function computations

Lemma: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then it is <u>piecewise affine</u>: a finite union of partial affine functions g_i: $\mathbb{N}^d \dashrightarrow \mathbb{N}$.

Furthermore, each "piece" dom g_i is a linear set.

$\Leftarrow$

Theorem: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then some CRN stably computes f.

Proof sketch:

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

Combining all affine function computations

Theorem: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then some CRN stably computes f.

$\Leftarrow$

Lemma: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then it is piecewise affine: a finite union of partial affine functions g_i: $\mathbb{N}^d \dashrightarrow \mathbb{N}$.

Furthermore, each "piece" dom g_i is a linear set.

Proof sketch:

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

Combining all affine function computations

Theorem: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then some CRN stably computes f.

$\Leftarrow$

Lemma: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then it is <u>piecewise affine</u>: a finite union of partial affine functions g_i: $\mathbb{N}^d \dashrightarrow \mathbb{N}$.

Furthermore, each "piece" dom g_i is a linear set.

Proof sketch:

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

Combining all affine function computations

Theorem: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then some CRN stably computes f.

$\Leftarrow$

Lemma: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then it is <u>piecewise affine</u>: a finite union of partial affine functions g_i: $\mathbb{N}^d \dashrightarrow \mathbb{N}$.

Furthermore, each "piece" dom g_i is a linear set.

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

Question 1: what's the point of species $\hat{Y}_i$?

Combining all affine function computations

Theorem: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then some CRN stably computes f.

$\Leftarrow$

Lemma: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then it is <u>piecewise affine</u>: a finite union of partial affine functions g_i: $\mathbb{N}^d \dashrightarrow \mathbb{N}$.

Furthermore, each "piece" dom g_i is a linear set.

Proof sketch:

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

Question 1: what's the point of species $\hat{Y}_i$?

Question 2: Something else doesn't work as described… what is it?

Combining all affine function computations

Lemma: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then it is piecewise affine: a finite union of partial affine functions g_i: $\mathbb{N}^d \dashrightarrow \mathbb{N}$.

Furthermore, each "piece" dom g_i is a linear set.

$\Leftarrow$

Theorem: If f: $\mathbb{N}^d \to \mathbb{N}$ is a semilinear function, then some CRN stably computes f.

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

Question 1: what's the point of species $\hat{Y}_i$?

Question 2: Something else doesn't work as described… what is it?

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

Limits of stable computation

Non-semilinear functions/predicates cannot be stably computed by CRNs

Impossibility of stably deciding non-semilinear sets

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

Impossibility of stably deciding non-semilinear sets

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

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

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

Impossibility of stably deciding non-semilinear sets

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

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

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

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

Impossibility of stably computing non-semilinear functions

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

Impossibility of stably computing non-semilinear functions

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

Proof:

1. Let C be a CRN stably computing f.
2. We convert C to a CRN D stably deciding graph(f) = $\{ (x_1,x_2,\ldots,x_k,y) \in \mathbb{N}^{k+1} \mid f(x_1,x_2,\ldots,x_k) = y \}$.

Impossibility of stably computing non-semilinear functions

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

Proof:

1. Let C be a CRN stably computing f.
2. We convert C to a CRN D stably deciding graph(f) = $\{ (x_1,x_2,\ldots,x_k,y) \in \mathbb{N}^{k+1} \mid f(x_1,x_2,\ldots,x_k) = y \}$.
3. Then graph(f) must be semilinear, since a CRN stably decides it. (*By first theorem on previous slide.*)

Impossibility of stably computing non-semilinear functions

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

Proof:

1. Let C be a CRN stably computing f.
2. We convert C to a CRN D stably deciding graph(f) = { $(x_1,x_2,\ldots,x_k,y) \in \mathbb{N}^{k+1} \mid f(x_1,x_2,\ldots,x_k) = y$ }.
3. Then graph(f) must be semilinear, since a CRN stably decides it. (*By first theorem on previous slide.*)
   1. **Key challenge**: D will run C with D's first k inputs. But D has to test the count of C's output Y to compare it to D's last input, without consuming Y, since consuming Y could disrupt the correctness of C.

Impossibility of stably computing non-semilinear functions

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

Proof:

1. Let C be a CRN stably computing f.
2. We convert C to a CRN D stably deciding graph(f) = $\{ (x_1,x_2,\ldots,x_k,y) \in \mathbb{N}^{k+1} \mid f(x_1,x_2,\ldots,x_k) = y \}$.
3. Then graph(f) must be semilinear, since a CRN stably decides it. (*By first theorem on previous slide.*)
   1. **Key challenge**: D will run C with D's first k inputs. But D has to test the count of C's output Y to compare it to D's last input, <u>without consuming Y</u>, since consuming Y could disrupt the correctness of C.
   2. **Solution**: we introduce two new species Y_P and Y_C representing #Y that are <u>only produced</u> by C, <u>never consumed</u>, so we are free to add reactions consuming them. This is called *dual-rail encoding*: $\#Y = \#Y_P - \#Y_C$

Impossibility of stably computing non-semilinear functions

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

Proof:
1. Let C be a CRN stably computing f.
2. We convert C to a CRN D stably deciding graph(f) = $\{ (x_1,x_2,\ldots,x_k,y) \in \mathbb{N}^{k+1} \mid f(x_1,x_2,\ldots,x_k) = y \}$.
3. Then graph(f) must be semilinear, since a CRN stably decides it. (*By first theorem on previous slide.*)
   1. **Key challenge**: D will run C with D's first k inputs. But D has to test the count of C's output Y to compare it to D's last input, <u>without consuming Y</u>, since consuming Y could disrupt the correctness of C.
   2. **Solution**: we introduce two new species Y_P and Y_C representing #Y that are <u>only produced</u> by C, <u>never consumed</u>, so we are free to add reactions consuming them. This is called *dual-rail encoding*: $\#Y = \#Y_P - \#Y_C$
4. For each reaction in C changing the count of output Y, add Y_P or Y_C as products to track the change:

Impossibility of stably computing non-semilinear functions

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

Proof:

1. Let C be a CRN stably computing f.
2. We convert C to a CRN D stably deciding graph(f) = $\{ (x_1,x_2,\ldots,x_k,y) \in \mathbb{N}^{k+1} \mid f(x_1,x_2,\ldots,x_k) = y \}$.
3. Then graph(f) must be semilinear, since a CRN stably decides it. (*By first theorem on previous slide.*)
   1. **Key challenge**: D will run C with D's first k inputs. But D has to test the count of C's output Y to compare it to D's last input, without consuming Y, since consuming Y could disrupt the correctness of C.
   2. **Solution**: we introduce two new species Y_P and Y_C representing #Y that are only produced by C, never consumed, so we are free to add reactions consuming them. This is called *dual-rail encoding*: $\#Y = \#Y_P - \#Y_C$
4. For each reaction in C changing the count of output Y, add Y_P or Y_C as products to track the change:
   - $X \to Z + 2Y$ becomes $X \to Z + 2Y + 2Y_P$ since there are net 2 Y's **produced**

Impossibility of stably computing non-semilinear functions

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

Proof:

1. Let C be a CRN stably computing f.
2. We convert C to a CRN D stably deciding graph(f) = $\{ (x_1,x_2,\ldots,x_k,y) \in \mathbb{N}^{k+1} \mid f(x_1,x_2,\ldots,x_k) = y \}$.
3. Then graph(f) must be semilinear, since a CRN stably decides it. (*By first theorem on previous slide.*)
   1. **Key challenge**: D will run C with D's first k inputs. But D has to test the count of C's output Y to compare it to D's last input, without consuming Y, since consuming Y could disrupt the correctness of C.
   2. **Solution**: we introduce two new species Y_P and Y_C representing #Y that are only produced by C, never consumed, so we are free to add reactions consuming them. This is called *dual-rail encoding*: $\#Y = \#Y_P - \#Y_C$
4. For each reaction in C changing the count of output Y, add Y_P or Y_C as products to track the change:
   - $X \to Z + 2Y$ becomes $X \to Z + 2Y + 2Y_P$ since there are net 2 Y's **produced**
   - $X + 6Y \to 2Y$ becomes $X + 6Y \to 2Y + 4Y_C$ since there are net 4 Y's **consumed**

Impossibility of stably computing non-semilinear functions

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

Proof:

1. Let C be a CRN stably computing f.
2. We convert C to a CRN D stably deciding graph(f) = { $(x_1,x_2,\ldots,x_k,y) \in \mathbb{N}^{k+1} \mid f(x_1,x_2,\ldots,x_k) = y$ }.
3. Then graph(f) must be semilinear, since a CRN stably decides it. (*By first theorem on previous slide.*)
   1. **Key challenge**: D will run C with D's first k inputs. But D has to test the count of C's output Y to compare it to D's last input, <u>without consuming Y</u>, since consuming Y could disrupt the correctness of C.
   2. **Solution**: we introduce two new species Y_P and Y_C representing $\#Y$ that are <u>only produced</u> by C, <u>never consumed</u>, so we are free to add reactions consuming them. This is called *dual-rail encoding*: $\#Y = \#Y_P - \#Y_C$
4. For each reaction in C changing the count of output Y, add Y_P or Y_C as products to track the change:
   - $X \to Z + 2Y$ becomes $X \to Z + 2Y + 2Y_P$ since there are net 2 Y's **produced**
   - $X + 6Y \to 2Y$ becomes $X + 6Y \to 2Y + 4Y_C$ since there are net 4 Y's **consumed**
5. For concreteness, assume k=1.
   - CRN D deciding graph(f) has 2 input species. The first is A. Let the second input species be Y_C.

Impossibility of stably computing non-semilinear functions

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

Proof:
1. Let C be a CRN stably computing f.
2. We convert C to a CRN D stably deciding graph(f) = { $(x_1,x_2,\ldots,x_k,y) \in \mathbb{N}^{k+1}$ | $f(x_1,x_2,\ldots,x_k) = y$ }.
3. Then graph(f) must be semilinear, since a CRN stably decides it. (*By first theorem on previous slide.*)
   1. **Key challenge**: D will run C with D's first k inputs. But D has to test the count of C's output Y to compare it to D's last input, <u>without consuming Y</u>, since consuming Y could disrupt the correctness of C.
   2. **Solution**: we introduce two new species Y_P and Y_C representing #Y that are <u>only produced</u> by C, <u>never consumed</u>, so we are free to add reactions consuming them. This is called *dual-rail encoding*: $\#Y = \#Y_P - \#Y_C$
4. For each reaction in C changing the count of output Y, add Y_P or Y_C as products to track the change:
   - $X \to Z + 2Y$ becomes $X \to Z + 2Y + 2Y_P$ since there are net 2 Y's **produced**
   - $X + 6Y \to 2Y$ becomes $X + 6Y \to 2Y + 4Y_C$ since there are net 4 Y's **consumed**
5. For concreteness, assume k=1.
   - CRN D deciding graph(f) has 2 input species. The first is A. Let the second input species be Y_C.
6. Since C stably computes f, eventually f(initial #A) more Y_P are produced than Y_C.

Impossibility of stably computing non-semilinear functions

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

Proof:

1. Let C be a CRN stably computing f.
2. We convert C to a CRN D stably deciding graph(f) = { $(x_1,x_2,\ldots,x_k,y) \in \mathbb{N}^{k+1}$ | $f(x_1,x_2,\ldots,x_k) = y$ }.
3. Then graph(f) must be semilinear, since a CRN stably decides it. (*By first theorem on previous slide.*)
   1. **Key challenge**: D will run C with D's first k inputs. But D has to test the count of C's output Y to compare it to D's last input, <u>without consuming Y</u>, since consuming Y could disrupt the correctness of C.
   2. **Solution**: we introduce two new species Y_P and Y_C representing #Y that are <u>only produced</u> by C, <u>never consumed</u>, so we are free to add reactions consuming them. This is called *dual-rail encoding*: $\#Y = \#Y_P - \#Y_C$
4. For each reaction in C changing the count of output Y, add Y_P or Y_C as products to track the change:
   - $X \to Z + 2Y$ becomes $X \to Z + 2Y + 2Y_P$ since there are net 2 Y's **produced**
   - $X + 6Y \to 2Y$ becomes $X + 6Y \to 2Y + 4Y_C$ since there are net 4 Y's **consumed**
5. For concreteness, assume k=1.
   - CRN D deciding graph(f) has 2 input species. The first is A. Let the second input species be Y_C.
6. Since C stably computes f, eventually f(initial #A) more Y_P are produced than Y_C.
7. If and only if initially $f(\#A) = \#Y_C$, then eventually $\#Y_P = \#Y_C$.

Impossibility of stably computing non-semilinear functions

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

Proof:

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

Impossibility of stably deciding a non-semilinear set

goal:

Theorem: The "squaring set" $S = \{ (x,y) \in \mathbb{N}^2 \mid x^2=y \}$ is <u>not</u> stably decidable by any CRN.

Additivity, nondecreasing sequences, minimal elements

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

Additivity, nondecreasing sequences, minimal elements

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

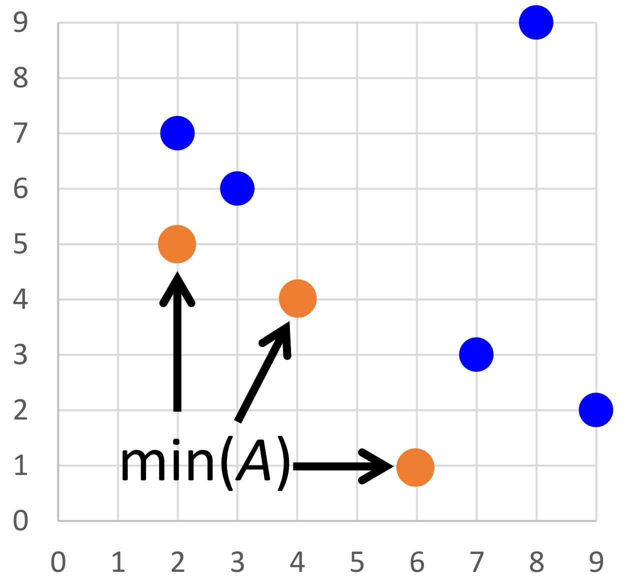
Definition: An infinite sequence of vectors $\mathbf{c}_1, \mathbf{c}_2, \ldots$ is <u>nondecreasing</u> if $\mathbf{c}_i \leq \mathbf{c}_{i+1}$ for all i. ($\mathbf{c}_i \leq \mathbf{c}_{i+1}$ means $\mathbf{c}_i(S) \leq \mathbf{c}_{i+1}(S)$ for all species S)

Additivity, nondecreasing sequences, minimal elements

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

Definition: An infinite sequence of vectors $\mathbf{c}_1$, $\mathbf{c}_2$, … is <u>nondecreasing</u> if $\mathbf{c}_i \leq \mathbf{c}_{i+1}$ for all i. ($\mathbf{c}_i \leq \mathbf{c}_{i+1}$ means $\mathbf{c}_i(S) \leq \mathbf{c}_{i+1}(S)$ for all species S)

Definition: Given $A \subseteq \mathbb{N}^d$, we say $\mathbf{y} \in A$ is <u>minimal</u> if, for all $\mathbf{x} \in A$, $\mathbf{x} \leq \mathbf{y}$ implies $\mathbf{x} = \mathbf{y}$, i.e., nothing in A is strictly smaller than $\mathbf{y}$. Let $\min(A)$ = minimal elements of A.



All vectors have a minimal vector under them

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

All vectors have a minimal vector under them

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

Proof:

1. If $\mathbf{x} \in \min(A)$ then we're done.

All vectors have a minimal vector under them

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

Proof:

1. If $\mathbf{x} \in \min(A)$ then we're done.
2. Otherwise, since $\mathbf{x} \notin \min(A)$, there is $\mathbf{x}_1 \in A$ such that $\mathbf{x}_1 < \mathbf{x}$.

All vectors have a minimal vector under them

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

Proof:
1. If $\mathbf{x} \in \min(A)$ then we're done.
2. Otherwise, since $\mathbf{x} \notin \min(A)$, there is $\mathbf{x}_1 \in A$ such that $\mathbf{x}_1 < \mathbf{x}$.
3. If $\mathbf{x}_1 \in \min(A)$ then we're done since $\mathbf{x}_1 \leq \mathbf{x}$.

All vectors have a minimal vector under them

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

Proof:

1. If $\mathbf{x} \in \min(A)$ then we're done.
2. Otherwise, since $\mathbf{x} \notin \min(A)$, there is $\mathbf{x}_1 \in A$ such that $\mathbf{x}_1 < \mathbf{x}$.
3. If $\mathbf{x}_1 \in \min(A)$ then we're done since $\mathbf{x}_1 \leq \mathbf{x}$.
4. Otherwise, since $\mathbf{x}_1 \notin \min(A)$, there is $\mathbf{x}_2 \in A$ such that $\mathbf{x}_2 < \mathbf{x}_1$.

All vectors have a minimal vector under them

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

Proof:

1. If $\mathbf{x} \in \min(A)$ then we're done.
2. Otherwise, since $\mathbf{x} \notin \min(A)$, there is $\mathbf{x}_1 \in A$ such that $\mathbf{x}_1 < \mathbf{x}$.
3. If $\mathbf{x}_1 \in \min(A)$ then we're done since $\mathbf{x}_1 \leq \mathbf{x}$.
4. Otherwise, since $\mathbf{x}_1 \notin \min(A)$, there is $\mathbf{x}_2 \in A$ such that $\mathbf{x}_2 < \mathbf{x}_1$.
5. …
6. Since there are only a finite number of $\mathbf{y}$ in $\mathbb{N}^d$ such that $\mathbf{y} < \mathbf{x}$, this process must terminate with a minimal vector $\mathbf{m} \in \min(A)$. **QED**

Dickson’s Lemma: Nondecreasing subsequences

Dickson’s Lemma: (1) Every infinite sequence ($\mathbf{x}_0$, $\mathbf{x}_1$, …) of vectors in $\mathbb{N}^d$ has an infinite nondecreasing subsequence, and (2) every set $A \subseteq \mathbb{N}^d$ has a finite number of minimal elements.

Dickson's Lemma: Nondecreasing subsequences

Dickson's Lemma: (1) Every infinite sequence ($\mathbf{x}_0$, $\mathbf{x}_1$, …) of vectors in $\mathbb{N}^d$ has an infinite nondecreasing subsequence, and (2) every set $A \subseteq \mathbb{N}^d$ has a finite number of minimal elements.

Proof:

1. We'll show condition (1) by induction on d.

Dickson's Lemma: Nondecreasing subsequences

Dickson's Lemma: (1) Every infinite sequence ($\mathbf{x}_0$, $\mathbf{x}_1$, …) of vectors in $\mathbb{N}^d$ has an infinite nondecreasing subsequence, and (2) every set $A \subseteq \mathbb{N}^d$ has a finite number of minimal elements.

Proof:

1. We'll show condition (1) by induction on *d*.
2. Base case $d = 1$: Let $X = x_0, x_1, \ldots$ be an infinite sequence of nonnegative integers.

Dickson's Lemma: Nondecreasing subsequences

Dickson's Lemma: (1) Every infinite sequence ($\mathbf{x}_0$, $\mathbf{x}_1$, …) of vectors in $\mathbb{N}^d$ has an infinite nondecreasing subsequence, and (2) every set $A \subseteq \mathbb{N}^d$ has a finite number of minimal elements.

Proof:

1. We'll show condition (1) by induction on *d*.
2. <u>Base case $d = 1$</u>: Let $X = x_0, x_1, \ldots$ be an infinite sequence of nonnegative integers.
   1. case 1: some $x \in \mathbb{N}$ appears infinitely often. Let the subsequence be (x, x, …), e.g. 1,1,5,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,…

Dickson's Lemma: Nondecreasing subsequences

Dickson's Lemma: (1) Every infinite sequence ($\mathbf{x}_0$, $\mathbf{x}_1$, …) of vectors in $\mathbb{N}^d$ has an infinite nondecreasing subsequence, and (2) every set $A \subseteq \mathbb{N}^d$ has a finite number of minimal elements.

Proof:

1. We'll show condition (1) by induction on d.
2. Base case $d = 1$: Let $X = x_0, x_1, \ldots$ be an infinite sequence of nonnegative integers.
   1. case 1: some $x \in \mathbb{N}$ appears infinitely often. Let the subsequence be $(x, x, \ldots)$, e.g. 1,1,5,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,…
   2. case 2: every $x \in \mathbb{N}$ appears finitely many times.

Dickson's Lemma: Nondecreasing subsequences

Dickson's Lemma: (1) Every infinite sequence ($\mathbf{x}_0$, $\mathbf{x}_1$, …) of vectors in $\mathbb{N}^d$ has an infinite nondecreasing subsequence, and (2) every set $A \subseteq \mathbb{N}^d$ has a finite number of minimal elements.

Proof:

1. We'll show condition (1) by induction on d.
2. <u>Base case d = 1</u>: Let $X = x_0, x_1$, … be an infinite sequence of nonnegative integers.
    1. case 1: some $x \in \mathbb{N}$ appears infinitely often. Let the subsequence be (x, x, …), e.g. 1,1,5,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,…
    2. case 2: every $x \in \mathbb{N}$ appears finitely many times.
        1. First element of subsequence is $y_0 = x_0$.

Dickson's Lemma: Nondecreasing subsequences

Dickson's Lemma: (1) Every infinite sequence ($\mathbf{x}_0$, $\mathbf{x}_1$, …) of vectors in $\mathbb{N}^d$ has an infinite nondecreasing subsequence, and (2) every set $A \subseteq \mathbb{N}^d$ has a finite number of minimal elements.

Proof:

1. We'll show condition (1) by induction on d.
2. Base case $d = 1$: Let $X = x_0, x_1, \ldots$ be an infinite sequence of nonnegative integers.
    1. case 1: some $x \in \mathbb{N}$ appears infinitely often. Let the subsequence be $(x, x, \ldots)$, e.g. 1,1,5,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,…
    2. case 2: every $x \in \mathbb{N}$ appears finitely many times.
        1. First element of subsequence is $y_0 = x_0$.
        2. Assuming we have finite increasing subsequence $y_0 < y_1 < \ldots < y_{k-1}$, let x_j be last occurrence of any integer $\leq y_{k-1}$ in original sequence $x_0, x_1, \ldots$, and let $y_k = x_{j+1}$, e.g., **2**,1,0, 3,2,1, 4,3,2, **5**,4,3, 6,5,4, 7,6,5, **8**,7,6, 9,8,7, 10,9,8, **11**,10,9…

Dickson's Lemma: Nondecreasing subsequences

Dickson's Lemma: (1) Every infinite sequence ($\mathbf{x}_0$, $\mathbf{x}_1$, …) of vectors in $\mathbb{N}^d$ has an infinite nondecreasing subsequence, and (2) every set $A \subseteq \mathbb{N}^d$ has a finite number of minimal elements.

Proof:

1. We'll show condition (1) by induction on d.
2. Base case d = 1: Let $X = x_0, x_1$, … be an infinite sequence of nonnegative integers.
    1. case 1: some $x \in \mathbb{N}$ appears infinitely often. Let the subsequence be (x, x, …), e.g. 1,1,5,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,…
    2. case 2: every $x \in \mathbb{N}$ appears finitely many times.
        1. First element of subsequence is $y_0 = x_0$.
        2. Assuming we have finite increasing subsequence $y_0 < y_1 < \ldots < y_{k-1}$, let x_j be last occurrence of any integer $\leq y_{k-1}$ in original sequence $x_0, x_1, \ldots$, and let $y_k = x_{j+1}$, e.g., **2**,1,0, 3,2,1, 4,3,2, **5**,4,3, 6,5,4, 7,6,5, **8**,7,6, 9,8,7, 10,9,8, **11**,10,9…
3. Inductive case d > 1:

Dickson's Lemma: Nondecreasing subsequences

Dickson's Lemma: (1) Every infinite sequence ($\mathbf{x}_0$, $\mathbf{x}_1$, …) of vectors in $\mathbb{N}^d$ has an infinite nondecreasing subsequence, and (2) every set $A \subseteq \mathbb{N}^d$ has a finite number of minimal elements.

Proof:

1. We'll show condition (1) by induction on d.
2. Base case $d = 1$: Let $X = x_0, x_1, \ldots$ be an infinite sequence of nonnegative integers.
   1. case 1: some $x \in \mathbb{N}$ appears infinitely often. Let the subsequence be $(x, x, \ldots)$, e.g. 1,1,5,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,…
   2. case 2: every $x \in \mathbb{N}$ appears finitely many times.
      1. First element of subsequence is $y_0 = x_0$.
      2. Assuming we have finite increasing subsequence $y_0 < y_1 < \ldots < y_{k-1}$, let x_j be last occurrence of any integer $\leq y_{k-1}$ in original sequence $x_0, x_1, \ldots$, and let $y_k = x_{j+1}$, e.g., **2**,1,0, 3,2,1, 4,3,2, **5**,4,3, 6,5,4, 7,6,5, **8**,7,6, 9,8,7, 10,9,8, **11**,10,9…
3. Inductive case $d > 1$:
   1. Inductively pick infinite subsequence X' such that the length-(d–1) prefix vectors are nondecreasing.

Dickson's Lemma: Nondecreasing subsequences

Dickson's Lemma: (1) Every infinite sequence ($\mathbf{x}_0$, $\mathbf{x}_1$, …) of vectors in $\mathbb{N}^d$ has an infinite nondecreasing subsequence, and (2) every set $A \subseteq \mathbb{N}^d$ has a finite number of minimal elements.

Proof:

1. We'll show condition (1) by induction on d.
2. Base case $d = 1$: Let $X = x_0, x_1, \ldots$ be an infinite sequence of nonnegative integers.
    1. case 1: some $x \in \mathbb{N}$ appears infinitely often. Let the subsequence be (x, x, …), e.g. 1,1,5,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,…
    2. case 2: every $x \in \mathbb{N}$ appears finitely many times.
        1. First element of subsequence is $y_0 = x_0$.
        2. Assuming we have finite increasing subsequence $y_0 < y_1 < \ldots < y_{k-1}$, let x_j be last occurrence of any integer $\leq y_{k-1}$ in original sequence $x_0, x_1, \ldots$, and let $y_k = x_{j+1}$, e.g., **2**,1,0, 3,2,1, 4,3,2, **5**,4,3, 6,5,4, 7,6,5, **8**,7,6, 9,8,7, 10,9,8, **11**,10,9…
3. Inductive case $d > 1$:
    1. Inductively pick infinite subsequence X' such that the length-(d–1) prefix vectors are nondecreasing.
    2. Pick an infinite subsequence of X' such that the d'th elements are also nondecreasing, as in base case.

Dickson's Lemma: Nondecreasing subsequences

Dickson's Lemma: (1) Every infinite sequence ($\mathbf{x}_0$, $\mathbf{x}_1$, …) of vectors in $\mathbb{N}^d$ has an infinite nondecreasing subsequence, and (2) every set $A \subseteq \mathbb{N}^d$ has a finite number of minimal elements.

Proof:

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

Dickson's Lemma: Nondecreasing subsequences

Dickson's Lemma: (1) Every infinite sequence $(\mathbf{x}_0, \mathbf{x}_1, \ldots)$ of vectors in $\mathbb{N}^d$ has an infinite nondecreasing subsequence, and (2) every set $A \subseteq \mathbb{N}^d$ has a finite number of minimal elements.

Proof:

1. We'll show condition (1) by induction on d.
2. Base case $d = 1$: Let $X = x_0, x_1, \ldots$ be an infinite sequence of nonnegative integers.
    1. case 1: some $x \in \mathbb{N}$ appears infinitely often. Let the subsequence be $(x, x, \ldots)$, e.g. 1,1,5,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,…
    2. case 2: every $x \in \mathbb{N}$ appears finitely many times.
        1. First element of subsequence is $y_0 = x_0$.
        2. Assuming we have finite increasing subsequence $y_0 < y_1 < \ldots < y_{k-1}$, let x_j be last occurrence of any integer $\leq y_{k-1}$ in original sequence $x_0, x_1, \ldots$, and let $y_k = x_{j+1}$, e.g., **2**,1,0, 3,2,1, 4,3,2, **5**,4,3, 6,5,4, 7,6,5, **8**,7,6, 9,8,7, 10,9,8, **11**,10,9…
3. Inductive case $d > 1$:
    1. Inductively pick infinite subsequence X' such that the length-$(d-1)$ prefix vectors are nondecreasing.
    2. Pick an infinite subsequence of X' such that the d'th elements are also nondecreasing, as in base case.
4. For condition (2), suppose that $\min(A)$ is infinite; put them in any order to make an infinite sequence.
5. By first condition, there's an infinite nondecreasing subsequence $\mathbf{m}_1 \leq \mathbf{m}_2 \leq \ldots$ of *distinct* vectors in $\min(A)$.

Dickson's Lemma: Nondecreasing subsequences

Dickson's Lemma: (1) Every infinite sequence $(\mathbf{x}_0, \mathbf{x}_1, \ldots)$ of vectors in $\mathbb{N}^d$ has an infinite nondecreasing subsequence, and (2) every set $A \subseteq \mathbb{N}^d$ has a finite number of minimal elements.

Proof:

1. We'll show condition (1) by induction on d.
2. Base case $d = 1$: Let $X = x_0, x_1, \ldots$ be an infinite sequence of nonnegative integers.
    1. case 1: some $x \in \mathbb{N}$ appears infinitely often. Let the subsequence be $(x, x, \ldots)$, e.g. 1,1,5,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,**3**,4,…
    2. case 2: every $x \in \mathbb{N}$ appears finitely many times.
        1. First element of subsequence is $y_0 = x_0$.
        2. Assuming we have finite increasing subsequence $y_0 < y_1 < \ldots < y_{k-1}$, let x_j be last occurrence of any integer $\leq y_{k-1}$ in original sequence $x_0, x_1, \ldots$, and let $y_k = x_{j+1}$, e.g., **2**,1,0, 3,2,1, 4,3,2, **5**,4,3, 6,5,4, 7,6,5, **8**,7,6, 9,8,7, 10,9,8, **11**,10,9…
3. Inductive case $d > 1$:
    1. Inductively pick infinite subsequence X' such that the length-$(d-1)$ prefix vectors are nondecreasing.
    2. Pick an infinite subsequence of X' such that the d'th elements are also nondecreasing, as in base case.
4. For condition (2), suppose that $\min(A)$ is infinite; put them in any order to make an infinite sequence.
5. By first condition, there's an infinite nondecreasing subsequence $\mathbf{m}_1 \leq \mathbf{m}_2 \leq \ldots$ of *distinct* vectors in $\min(A)$.
6. Since they are distinct, $\mathbf{m}_1 < \mathbf{m}_2 < \ldots$, but $\mathbf{m}_1 < \mathbf{m}_2$ contradicts the minimality of $\mathbf{m}_2$. **QED**

Properties of stable configurations

- For convenience, assume every species votes.

Properties of stable configurations

- For convenience, assume every species votes.
- Thus a stable YES-output configuration **o** with output φ(**o**) = YES is one in which, for all **o'** ∈ Reach(**o**), all NO voters are absent from **o'**. (Similarly for stable NO-output.)

Properties of stable configurations

- For convenience, assume every species votes.
- Thus a <u>stable</u> YES-output configuration **o** with output φ(**o**) = YES is one in which, for all **o'** ∈ Reach(**o**), all NO voters are absent from **o'**. (Similarly for stable NO-output.)
- Conversely, an **unstable** configuration **c** is one in which at least one of the following holds:

Properties of stable configurations

- For convenience, assume every species votes.
- Thus a stable YES-output configuration **o** with output φ(**o**) = YES is one in which, for all **o'** ∈ Reach(**o**), all NO voters are absent from **o'**. (Similarly for stable NO-output.)
- Conversely, an **unstable** configuration **c** is one in which at least one of the following holds:
  - YES and NO voters both exist already in **c** (output undefined)

Properties of stable configurations

- For convenience, assume every species votes.
- Thus a stable YES-output configuration **o** with output φ(**o**) = YES is one in which, for all **o'** ∈ Reach(**o**), all NO voters are absent from **o'**. (Similarly for stable NO-output.)
- Conversely, an **unstable** configuration **c** is one in which at least one of the following holds:
  - YES and NO voters both exist already in **c** (output undefined)
  - Only YES voters exist, but a NO voter is producible in some **c'** ∈ Reach(**c**).

Properties of stable configurations

- For convenience, assume every species votes.
- Thus a stable YES-output configuration **o** with output φ(**o**) = YES is one in which, for all **o'** ∈ Reach(**o**), all NO voters are absent from **o'**. (Similarly for stable NO-output.)
- Conversely, an **unstable** configuration **c** is one in which at least one of the following holds:
  - YES and NO voters both exist already in **c** (output undefined)
  - Only YES voters exist, but a NO voter is producible in some **c'** ∈ Reach(**c**).
  - Only NO voters exist, but a YES voter is producible in some **c'** ∈ Reach(**c**).

Properties of stable configurations

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

Properties of stable configurations

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

Observation: The <u>unstable configurations are closed upwards</u>: for all unstable **c** and all **d** ≥ **c**, **d** is also unstable.

Properties of stable configurations

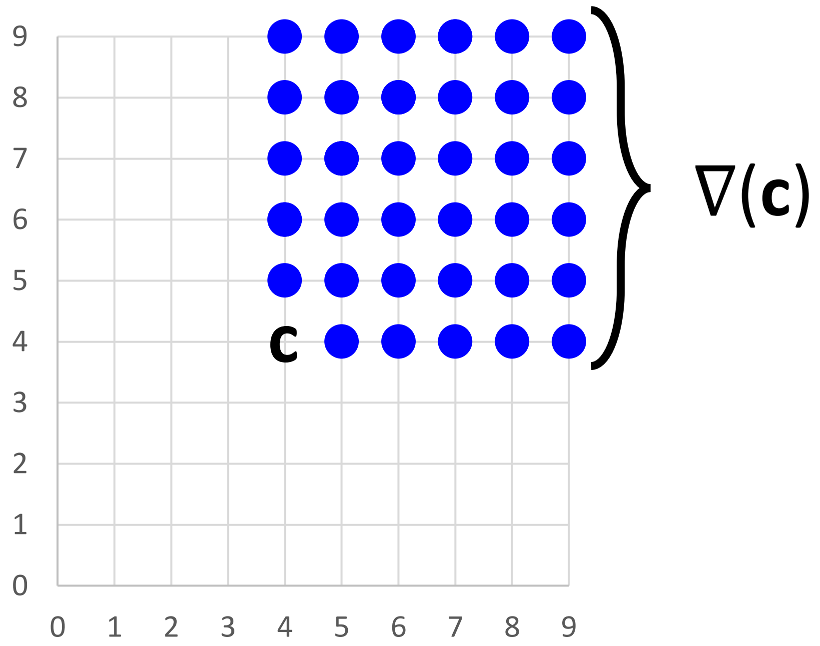
- For convenience, assume every species votes.
- Thus a <u>stable</u> YES-output configuration **o** with output φ(**o**) = YES is one in which, for all **o'** ∈ Reach(**o**), all NO voters are absent from **o'**. (Similarly for stable NO-output.)
- Conversely, an **unstable** configuration **c** is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

Observation: The <u>unstable configurations are closed upwards</u>: for all unstable **c** and all **d** ≥ **c**, **d** is also unstable.

Corollary: The <u>stable configurations are closed downwards</u>: for all stable **c** and all **b** ≤ **c**, **b** is also stable.

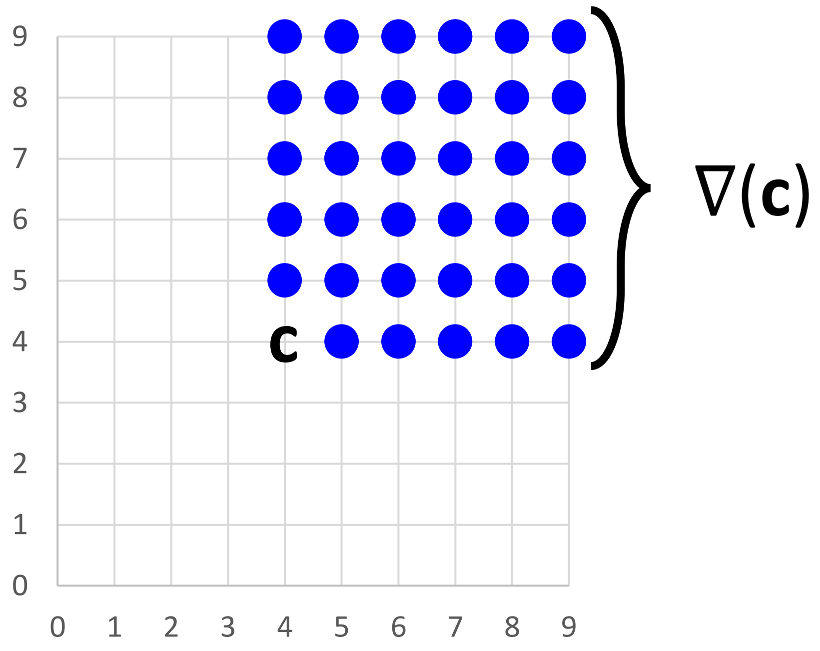
Upper cones

Definition: For all $\mathbf{c} \in \mathbb{N}^d$, let $\nabla(\mathbf{c}) = \{\, \mathbf{d} \in \mathbb{N}^d \mid \mathbf{c} \leq \mathbf{d} \,\}$ denote the <u>upper cone</u> of **c**.



Upper cones

Definition: For all $\mathbf{c} \in \mathbb{N}^d$, let $\nabla(\mathbf{c}) = \{ \mathbf{d} \in \mathbb{N}^d \mid \mathbf{c} \leq \mathbf{d} \}$ denote the <u>upper cone</u> of **c**.



Observation (*reworded from previous slide*):
For all unstable **c** and all $\mathbf{d} \in \nabla(\mathbf{c})$,
d is also unstable.

Set of unstable configurations is finite union of cones

Recall:

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

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

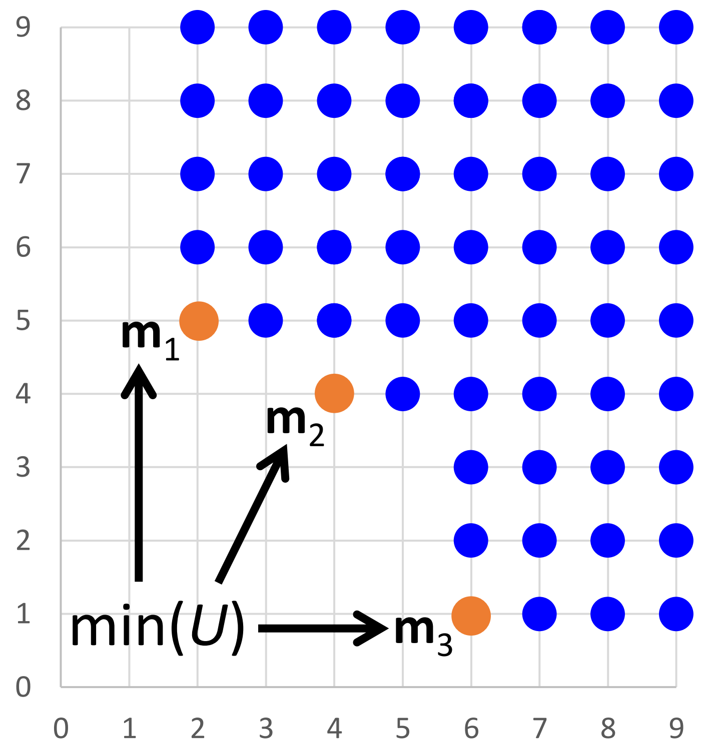
Set of unstable configurations is finite union of cones

Recall:

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

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

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



Set of unstable configurations is finite union of cones

Recall:

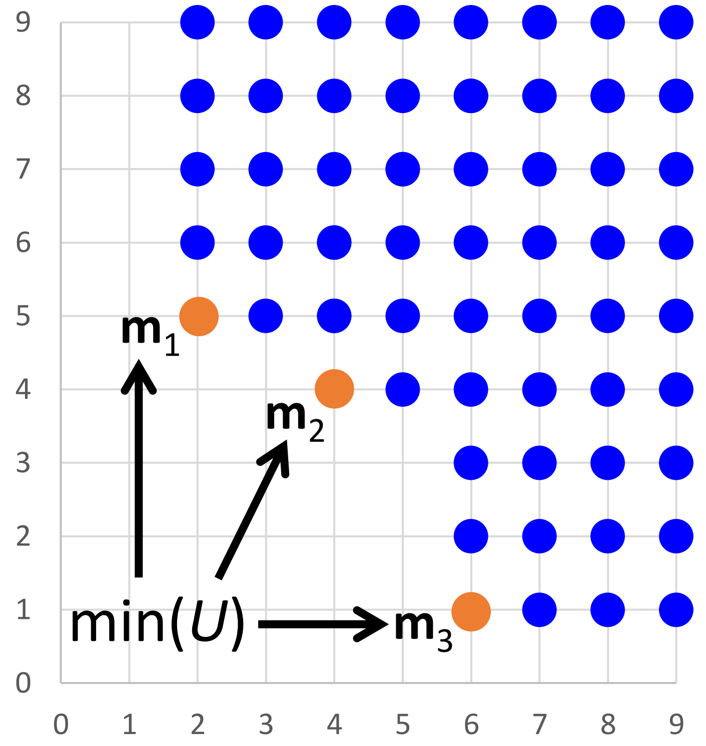
Observation 1: *Unstable configs are closed upwards*: For all unstable **c** and all **d** ∈ ∇(**c**), **d** is also unstable.

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

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

Proof:

1. Let $C = \bigcup_{\mathbf{m} \in \min(U)} \nabla(\mathbf{m})$. Need to show $C = U$.



Set of unstable configurations is finite union of cones

Recall:

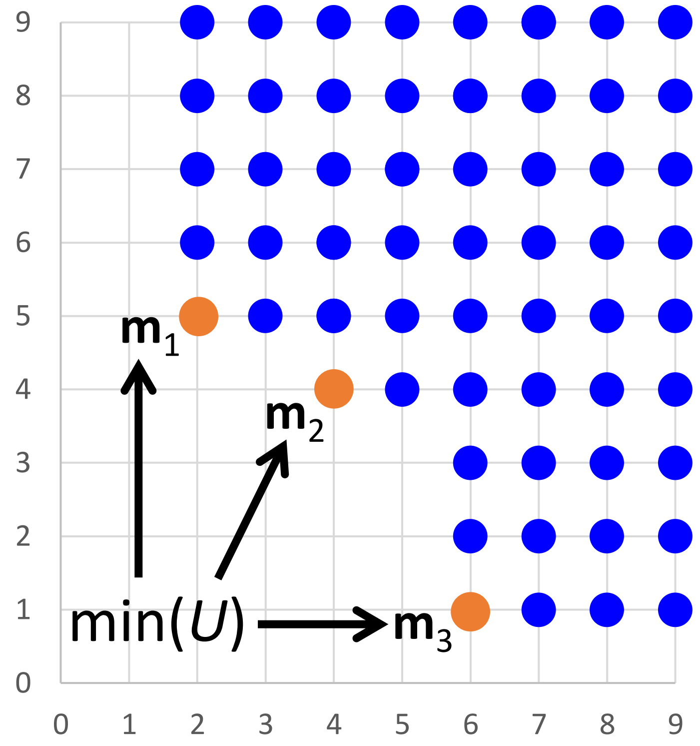
Observation 1: *Unstable configs are closed upwards*: For all unstable **c** and all **d** ∈ ∇(**c**), **d** is also unstable.

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

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

Proof:

1. Let $C = \bigcup_{\mathbf{m} \in \min(U)} \nabla(\mathbf{m})$. Need to show *C* = *U*.
2. To see that *C* ⊆ *U*, let **x** ∈ *C*, i.e., **x** ∈ ∇(**m**) for some **m** ∈ min(*U*).



Set of unstable configurations is finite union of cones

Recall:

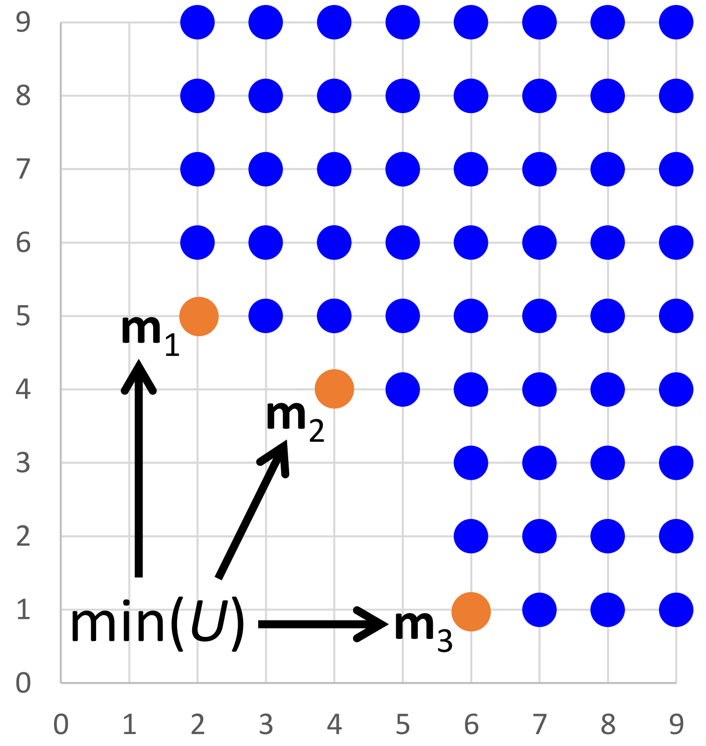
Observation 1: *Unstable configs are closed upwards*: For all unstable **c** and all **d** ∈ ∇(**c**), **d** is also unstable.

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

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

Proof:

1. Let $C = \bigcup_{\mathbf{m}\in\min(U)} \nabla(\mathbf{m})$. Need to show *C* = *U*.
2. To see that *C* ⊆ *U*, let **x** ∈ *C*, i.e., **x** ∈ ∇(**m**) for some **m** ∈ min(*U*).
3. By Observation 1, since **m** ∈ *U*, also **x** ∈ *U*, so *C* ⊆ *U*.



Set of unstable configurations is finite union of cones

Recall:

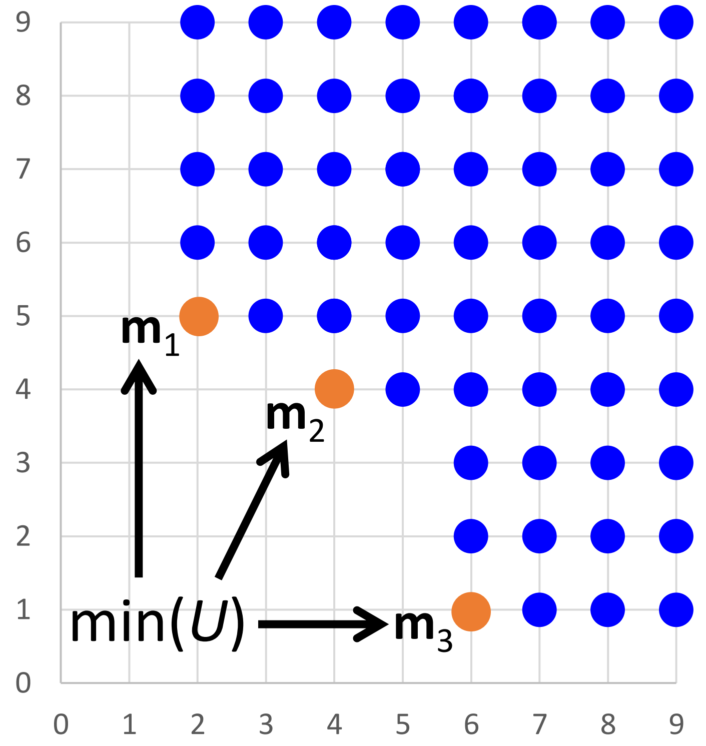
Observation 1: *Unstable configs are closed upwards*: For all unstable **c** and all **d** ∈ ∇(**c**), **d** is also unstable.

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

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

Proof:

1. Let $C = \bigcup_{\mathbf{m}\in\min(U)} \nabla(\mathbf{m})$. Need to show *C* = *U*.
2. To see that *C* ⊆ *U*, let **x** ∈ *C*, i.e., **x** ∈ ∇(**m**) for some **m** ∈ min(*U*).
3. By Observation 1, since **m** ∈ *U*, also **x** ∈ *U*, so *C* ⊆ *U*.
4. To see that *U* ⊆ *C*, let **x** ∈ *U*.



Set of unstable configurations is finite union of cones

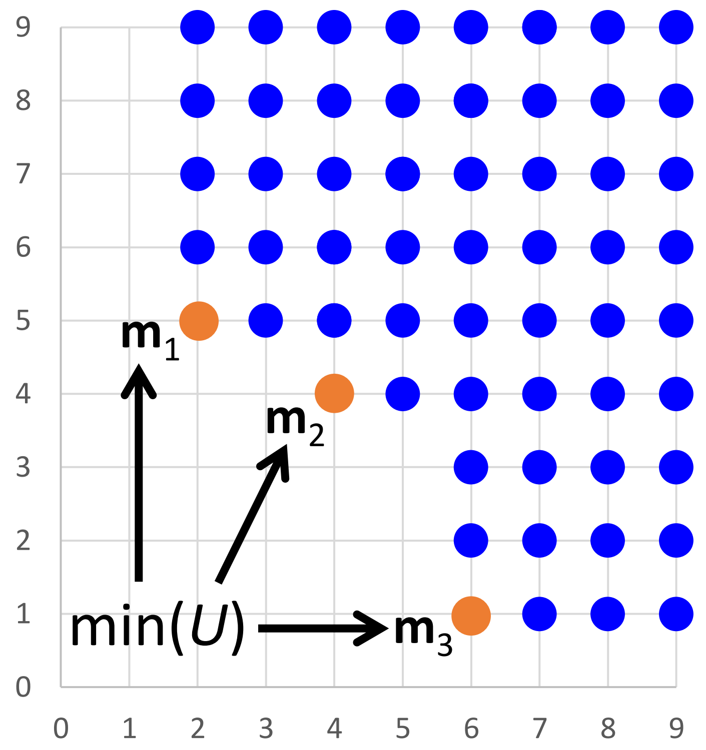
Recall:

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

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

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

Proof:
1. Let *C* = $\bigcup_{\mathbf{m}\in\min(U)} \nabla(\mathbf{m})$. Need to show *C* = *U*.
2. To see that *C* ⊆ *U*, let **x** ∈ *C*, i.e., **x** ∈ ∇(**m**) for some **m** ∈ min(*U*).
3. By Observation 1, since **m** ∈ *U*, also **x** ∈ *U*, so *C* ⊆ *U*.
4. To see that *U* ⊆ *C*, let **x** ∈ *U*.
5. By Observation 2, for some **m** ∈ min(*U*), **m** ≤ **x**.



Set of unstable configurations is finite union of cones

Recall:

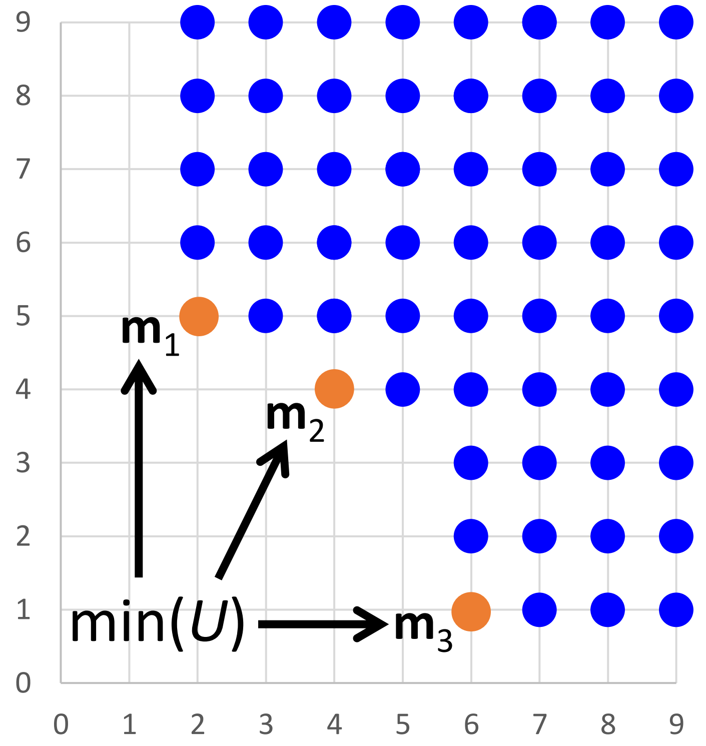
Observation 1: *Unstable configs are closed upwards*: For all unstable **c** and all **d** ∈ ∇(**c**), **d** is also unstable.

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

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

Proof:

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



Stable configurations are closed upwards for species that are already “large”

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

Stable configurations are closed upwards for species that are already "large"

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

Definition: Let τ = max { $\mathbf{m}(S)$ | $\mathbf{m} \in \min(U)$, S ∈ Λ }.
The hypercube with corner $(\tau, \tau, \ldots, \tau) \in \mathbb{N}^d$ (*and other corner at origin*) contains every minimal **m** defining U.

Stable configurations are closed upwards for species that are already "large"

Recall stable configs are <u>closed downward</u>. They are also <u>closed upward for "already large" species</u>.

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

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

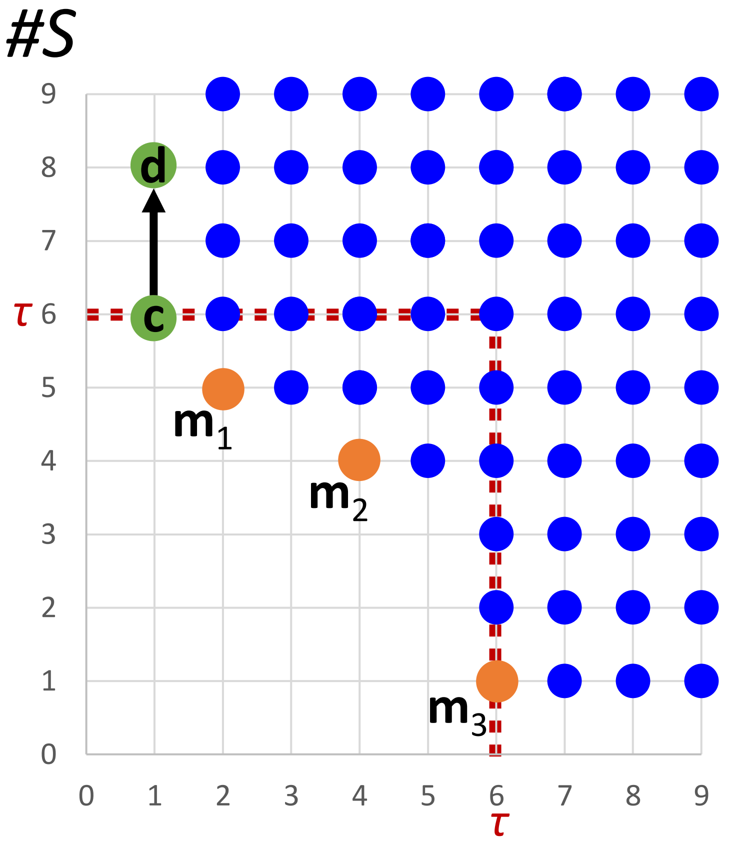
Stable configurations are closed upwards for species that are already "large"

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

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

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

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



A pumping lemma

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

A pumping lemma

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

Proof:

1. By Dickson's Lemma there is infinite nondecreasing subsequence $\mathbf{c}_0 \leq \mathbf{c}_1 \leq \ldots$, each $\mathbf{c}_i \in A$. Let $\boldsymbol{\delta}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.

A pumping lemma

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

Proof:

1. By Dickson's Lemma there is infinite nondecreasing subsequence $\mathbf{c}_0 \leq \mathbf{c}_1 \leq \ldots$, each $\mathbf{c}_i \in A$. Let $\boldsymbol{\delta}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.

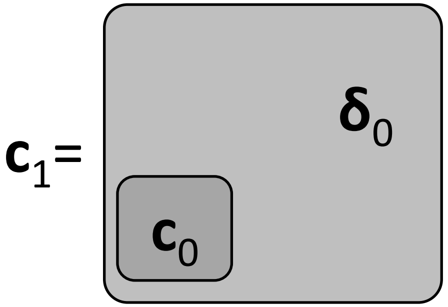
$\mathbf{c}_0$

A pumping lemma

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

Proof:

1. By Dickson's Lemma there is infinite nondecreasing subsequence $\mathbf{c}_0 \leq \mathbf{c}_1 \leq \ldots$, each $\mathbf{c}_i \in A$. Let $\boldsymbol{\delta}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.

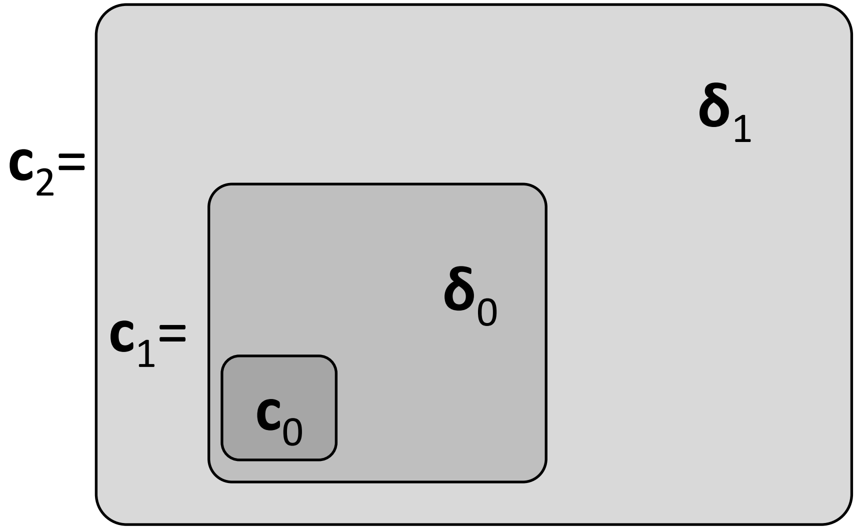


A pumping lemma

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

Proof:

1. By Dickson's Lemma there is infinite nondecreasing subsequence $\mathbf{c}_0 \leq \mathbf{c}_1 \leq \ldots$, each $\mathbf{c}_i \in A$. Let $\boldsymbol{\delta}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.

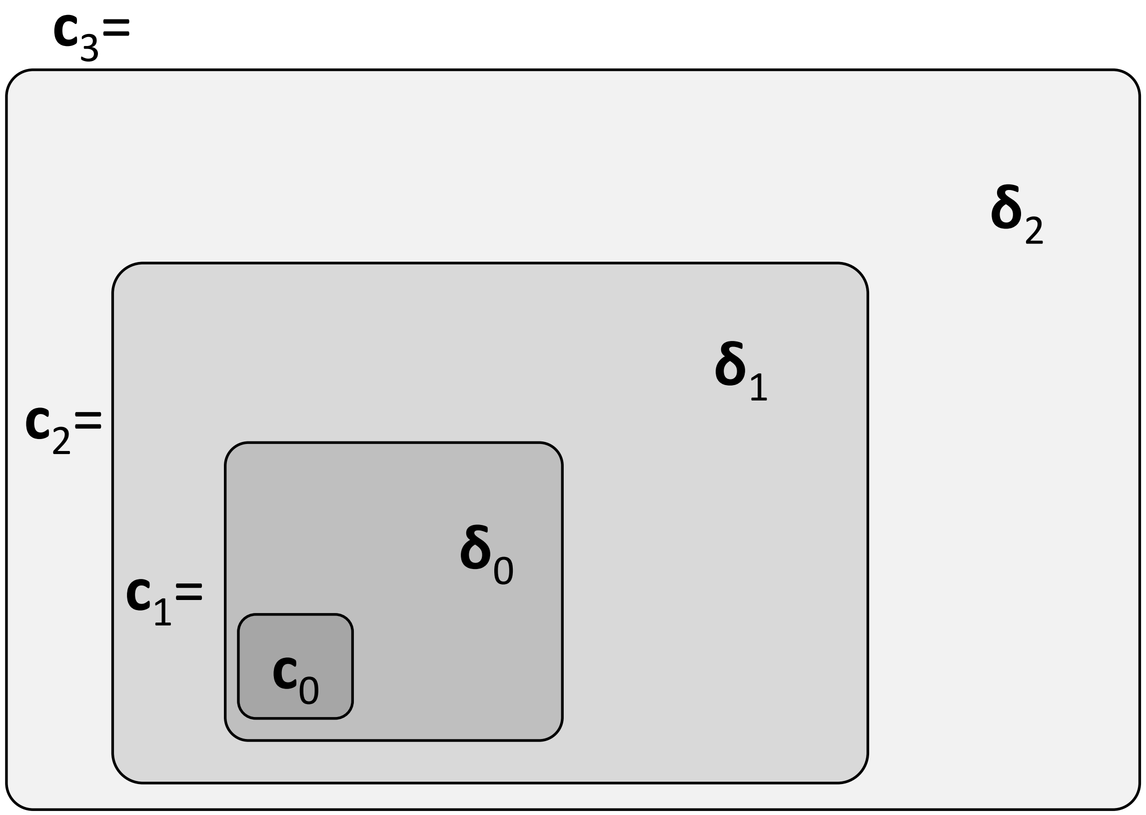


A pumping lemma

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

Proof:

1. By Dickson's Lemma there is infinite nondecreasing subsequence $\mathbf{c}_0 \leq \mathbf{c}_1 \leq \ldots$, each $\mathbf{c}_i \in A$. Let $\boldsymbol{\delta}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.

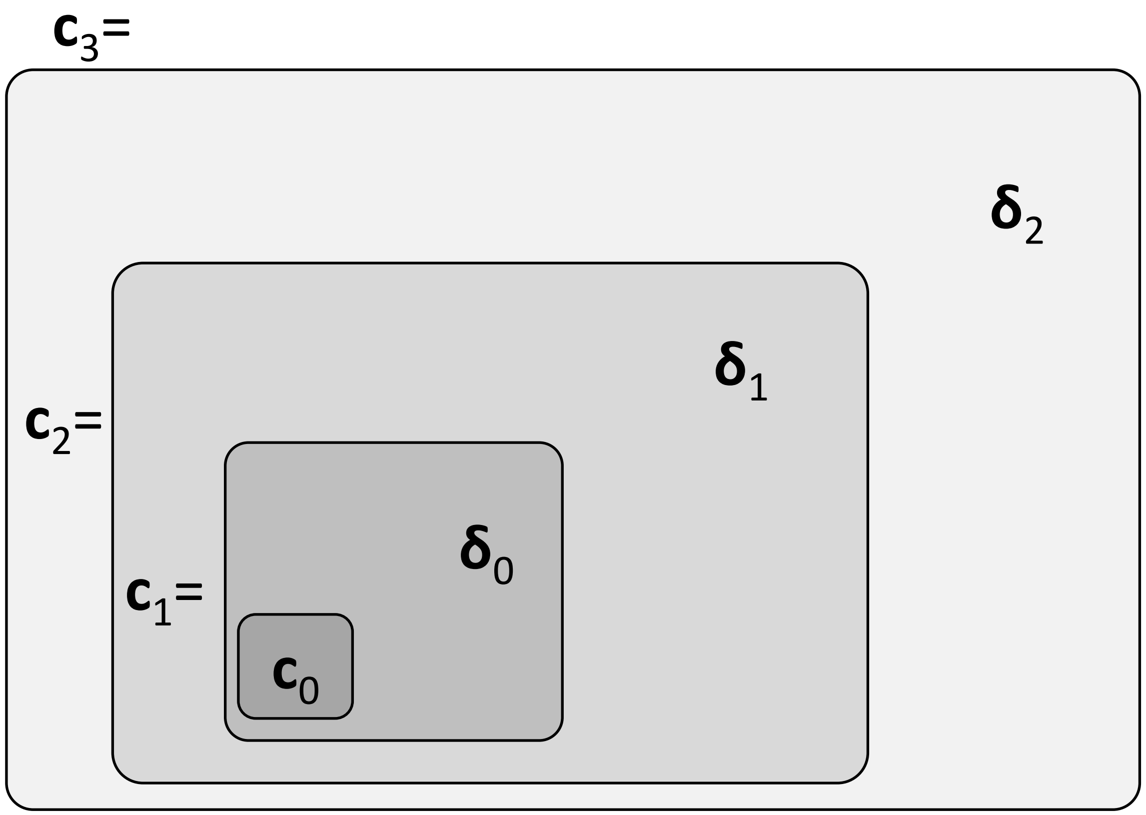


A pumping lemma

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

Proof:

1. By Dickson's Lemma there is infinite nondecreasing subsequence $\mathbf{c}_0 \leq \mathbf{c}_1 \leq \ldots$, each $\mathbf{c}_i \in A$. Let $\boldsymbol{\delta}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.
2. Define sequence of stable $\mathbf{o}_0, \mathbf{o}_1, \ldots$ inductively as follows.

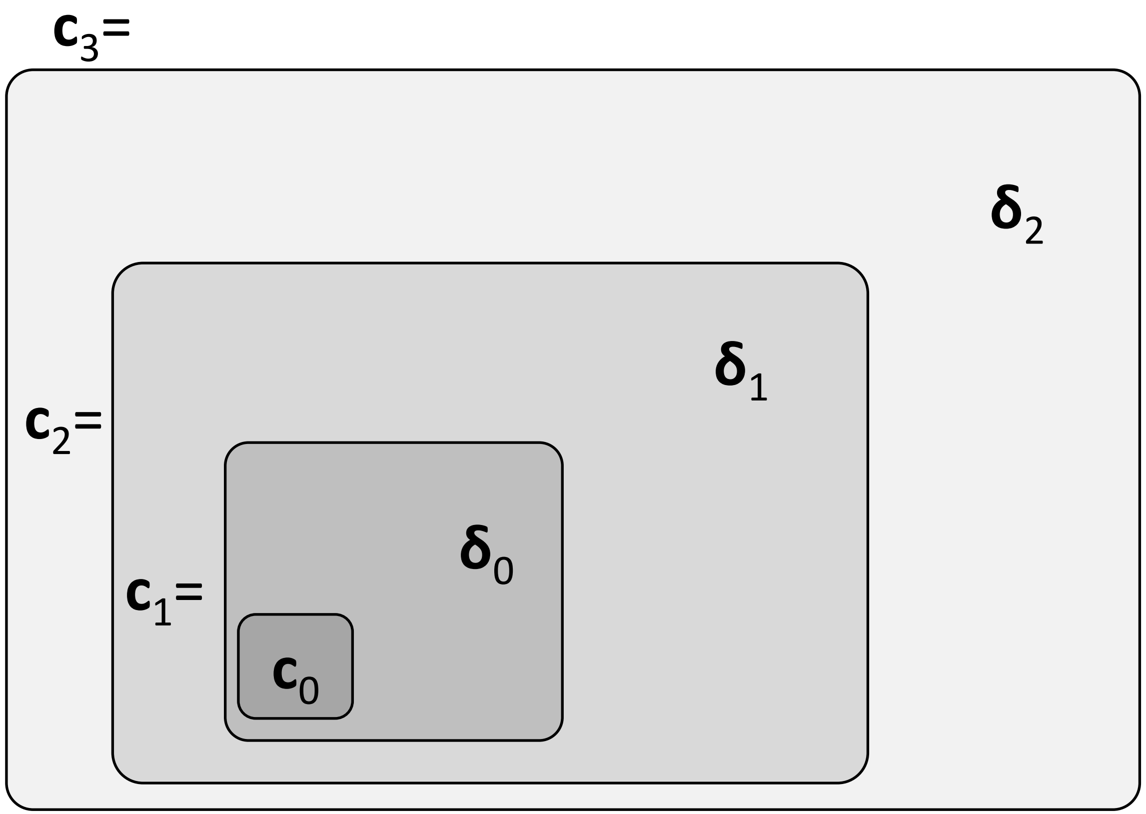


A pumping lemma

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

Proof:

1. By Dickson's Lemma there is infinite nondecreasing subsequence $\mathbf{c}_0 \leq \mathbf{c}_1 \leq \ldots$, each $\mathbf{c}_i \in A$. Let $\boldsymbol{\delta}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.
2. Define sequence of stable $\mathbf{o}_0$, $\mathbf{o}_1$, … inductively as follows.
3. Base case: $\mathbf{c}_0 \Rightarrow \mathbf{o}_0$ for some stable $\mathbf{o}_0$.

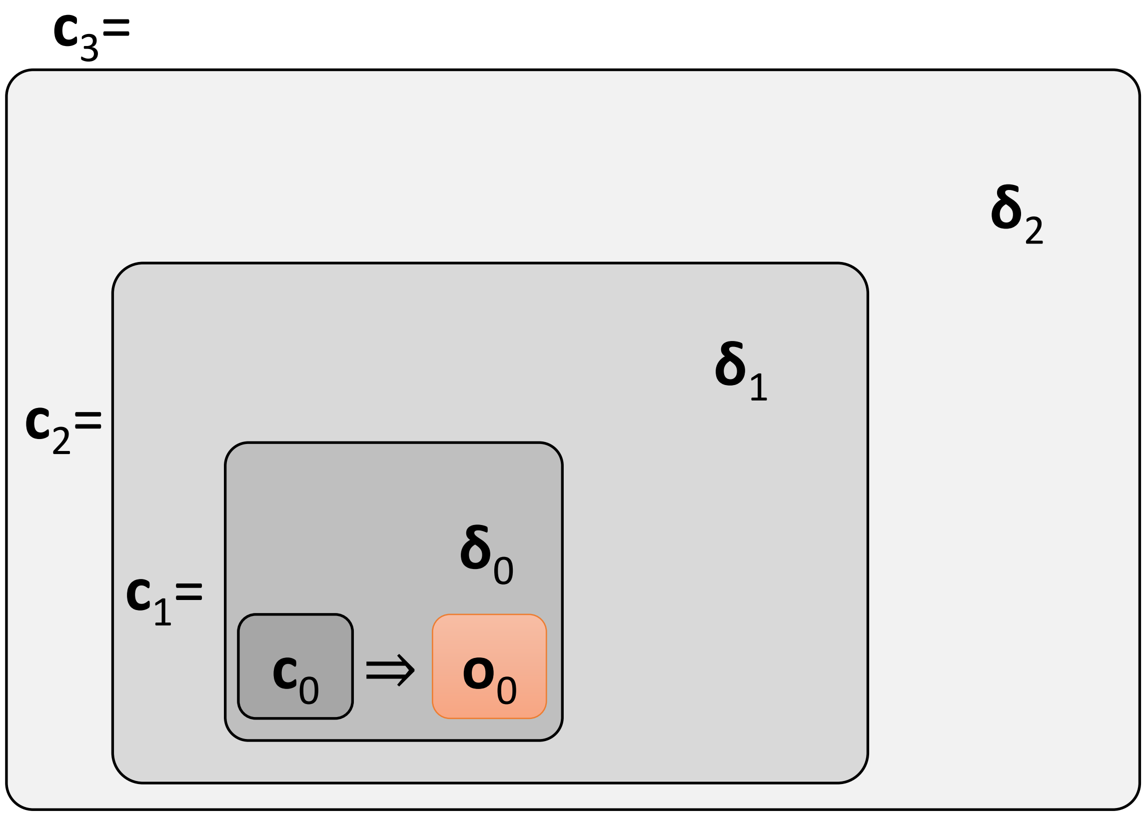


A pumping lemma

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

Proof:

1. By Dickson's Lemma there is infinite nondecreasing subsequence $\mathbf{c}_0 \leq \mathbf{c}_1 \leq \ldots$, each $\mathbf{c}_i \in A$. Let $\boldsymbol{\delta}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.
2. Define sequence of stable $\mathbf{o}_0, \mathbf{o}_1, \ldots$ inductively as follows.
3. <u>Base case</u>: $\mathbf{c}_0 \Rightarrow \mathbf{o}_0$ for some stable $\mathbf{o}_0$.

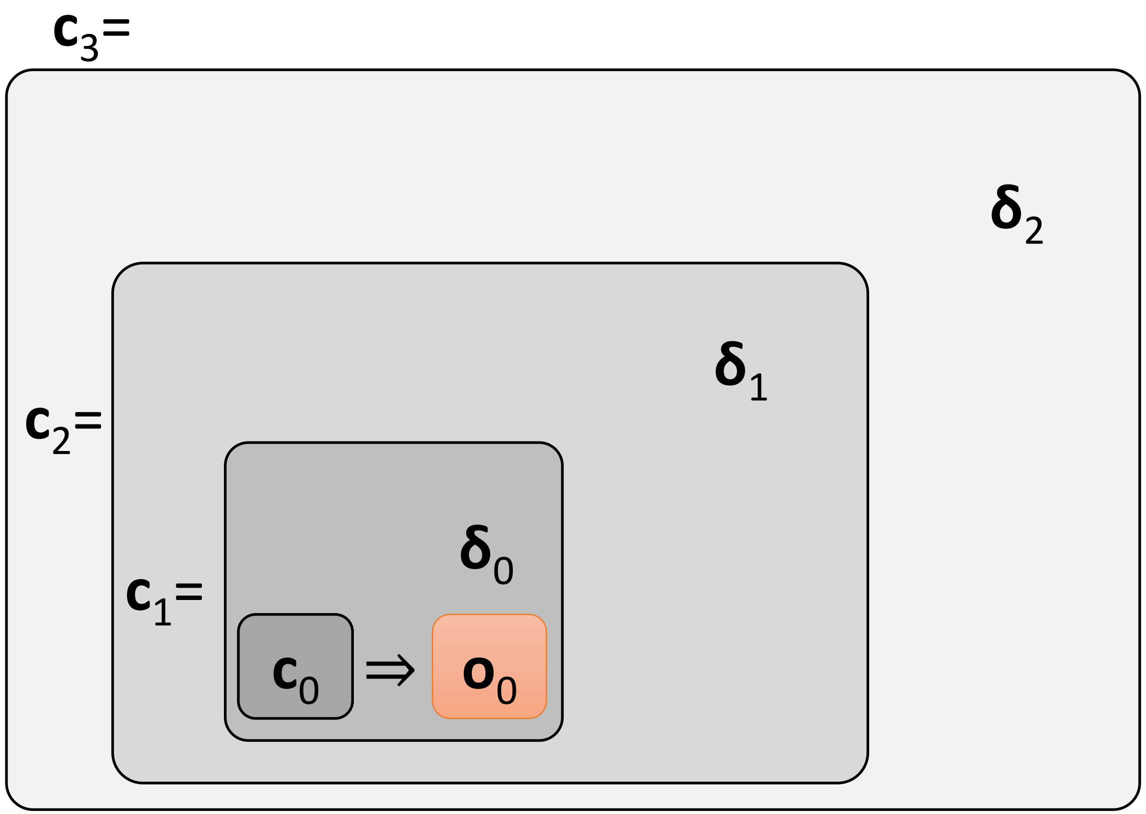


A pumping lemma

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

Proof:

1. By Dickson's Lemma there is infinite nondecreasing subsequence $\mathbf{c}_0 \leq \mathbf{c}_1 \leq \ldots$, each $\mathbf{c}_i \in A$. Let $\boldsymbol{\delta}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.
2. Define sequence of stable $\mathbf{o}_0$, $\mathbf{o}_1$, … inductively as follows.
3. Base case: $\mathbf{c}_0 \Rightarrow \mathbf{o}_0$ for some stable $\mathbf{o}_0$.
4. Inductive case: By additivity $\mathbf{c}_{i+1} = \mathbf{c}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_i + \boldsymbol{\delta}_i$.

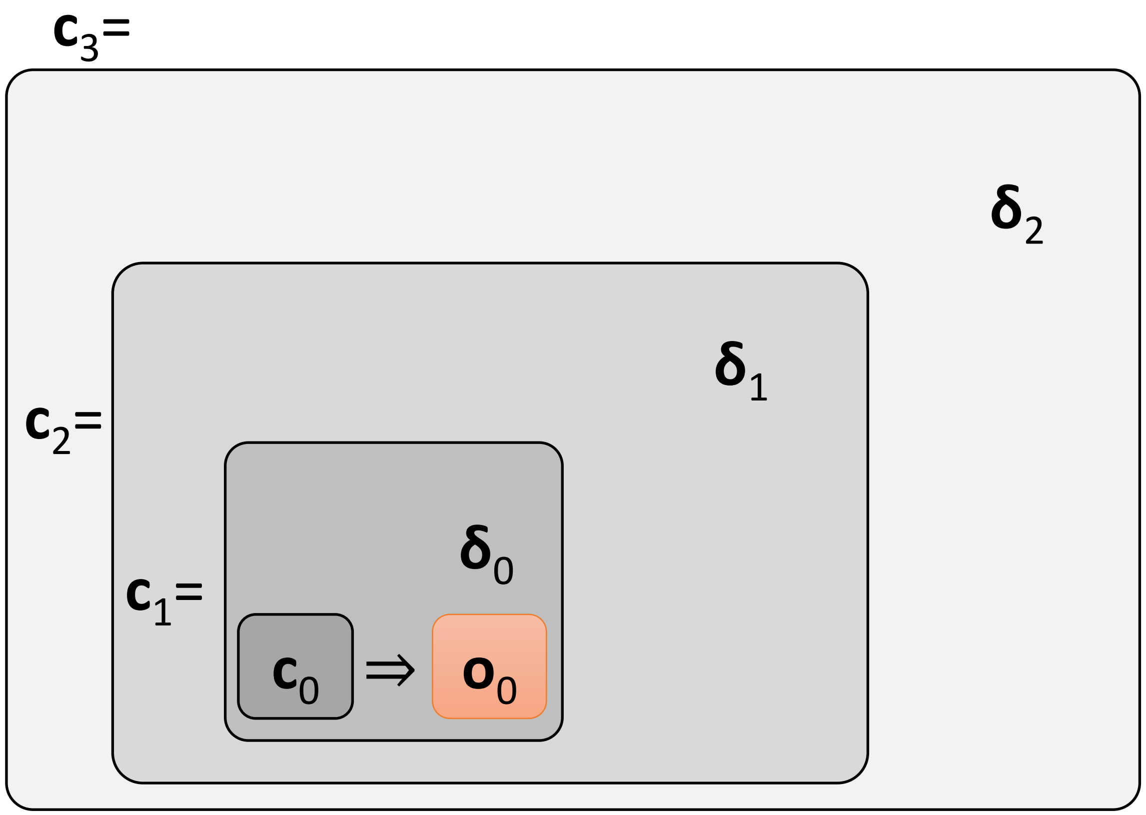


A pumping lemma

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

Proof:

1. By Dickson's Lemma there is infinite nondecreasing subsequence $\mathbf{c}_0 \leq \mathbf{c}_1 \leq \ldots$, each $\mathbf{c}_i \in A$. Let $\boldsymbol{\delta}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.
2. Define sequence of stable $\mathbf{o}_0$, $\mathbf{o}_1$, … inductively as follows.
3. Base case: $\mathbf{c}_0 \Rightarrow \mathbf{o}_0$ for some stable $\mathbf{o}_0$.
4. Inductive case: By additivity $\mathbf{c}_{i+1} = \mathbf{c}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_i + \boldsymbol{\delta}_i$.
5. By correctness $\mathbf{o}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_{i+1}$ for some stable $\mathbf{o}_{i+1}$.

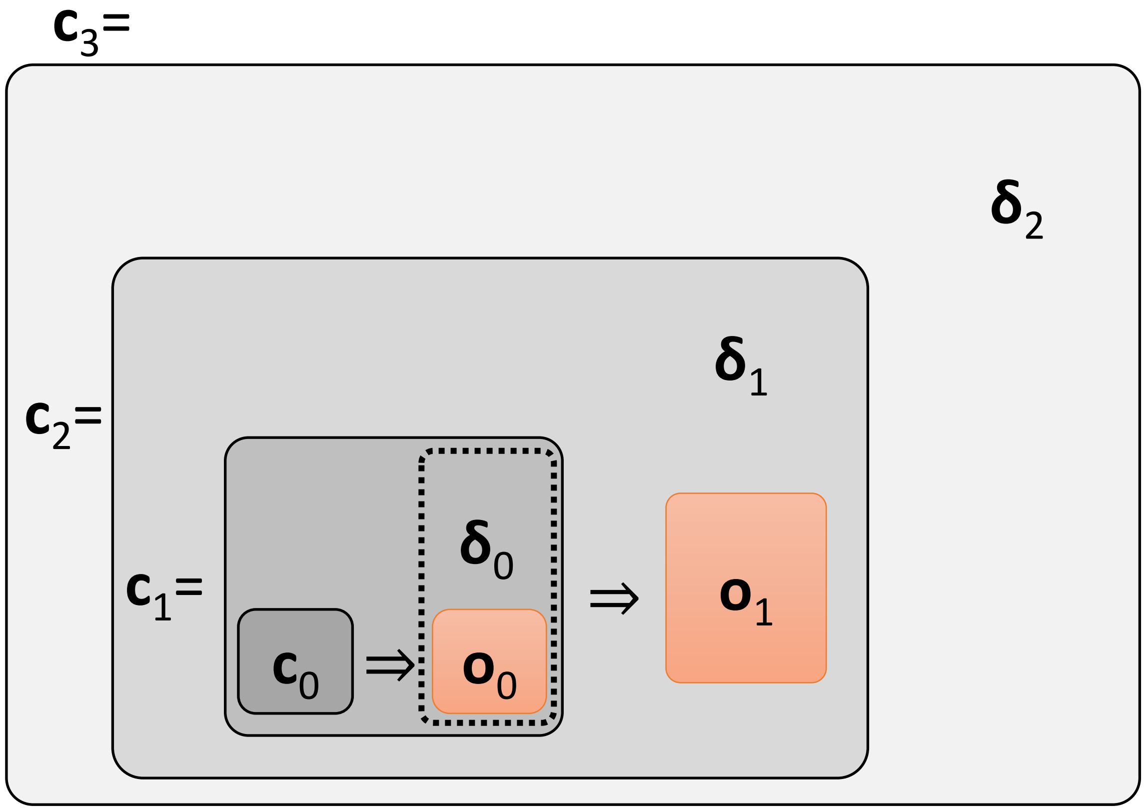


A pumping lemma

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

Proof:

1. By Dickson's Lemma there is infinite nondecreasing subsequence $\mathbf{c}_0 \le \mathbf{c}_1 \le \ldots$, each $\mathbf{c}_i \in A$. Let $\boldsymbol{\delta}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.
2. Define sequence of stable $\mathbf{o}_0$, $\mathbf{o}_1$, … inductively as follows.
3. Base case: $\mathbf{c}_0 \Rightarrow \mathbf{o}_0$ for some stable $\mathbf{o}_0$.
4. Inductive case: By additivity $\mathbf{c}_{i+1} = \mathbf{c}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_i + \boldsymbol{\delta}_i$.
5. By correctness $\mathbf{o}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_{i+1}$ for some stable $\mathbf{o}_{i+1}$.

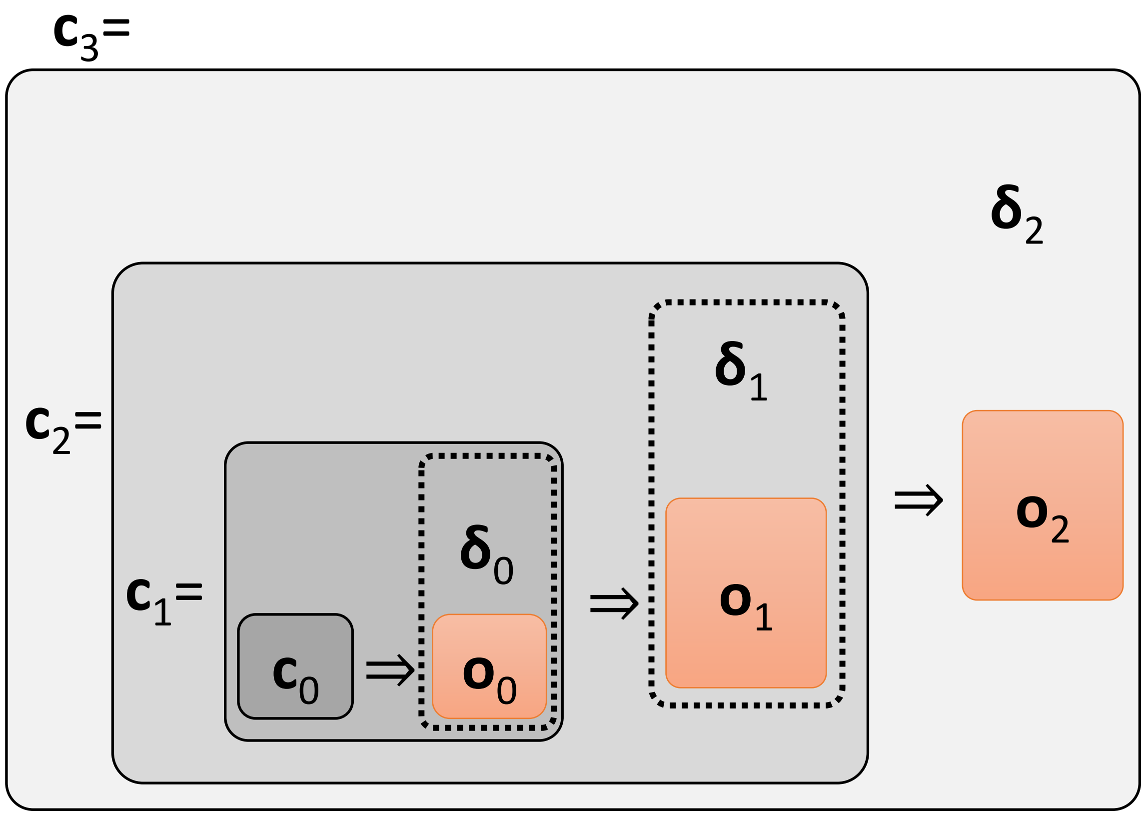


A pumping lemma

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

Proof:

1. By Dickson's Lemma there is infinite nondecreasing subsequence $\mathbf{c}_0 \leq \mathbf{c}_1 \leq \ldots$, each $\mathbf{c}_i \in A$. Let $\boldsymbol{\delta}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.
2. Define sequence of stable $\mathbf{o}_0$, $\mathbf{o}_1$, … inductively as follows.
3. Base case: $\mathbf{c}_0 \Rightarrow \mathbf{o}_0$ for some stable $\mathbf{o}_0$.
4. Inductive case: By additivity $\mathbf{c}_{i+1} = \mathbf{c}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_i + \boldsymbol{\delta}_i$.
5. By correctness $\mathbf{o}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_{i+1}$ for some stable $\mathbf{o}_{i+1}$.

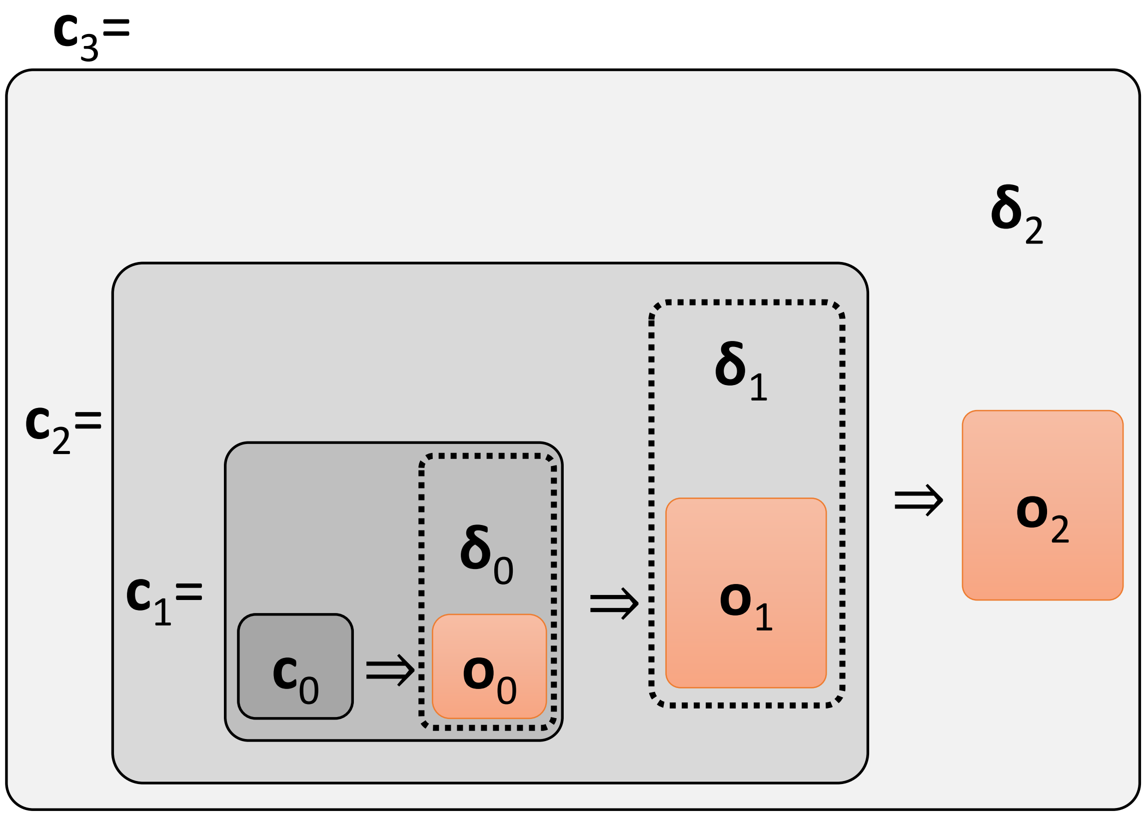


A pumping lemma

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

Proof:

1. By Dickson's Lemma there is infinite nondecreasing subsequence $\mathbf{c}_0 \leq \mathbf{c}_1 \leq \ldots$, each $\mathbf{c}_i \in A$. Let $\boldsymbol{\delta}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.
2. Define sequence of stable $\mathbf{o}_0$, $\mathbf{o}_1$, … inductively as follows.
3. Base case: $\mathbf{c}_0 \Rightarrow \mathbf{o}_0$ for some stable $\mathbf{o}_0$.
4. Inductive case: By additivity $\mathbf{c}_{i+1} = \mathbf{c}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_i + \boldsymbol{\delta}_i$.
5. By correctness $\mathbf{o}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_{i+1}$ for some stable $\mathbf{o}_{i+1}$.
6. By Dickson's Lemma pick infinite nondecreasing subsequence $\mathbf{o'}_0 \leq \mathbf{o'}_1 \leq \ldots$ of $\mathbf{o}_i$'s. For the sake of readability let's assume this is just the original sequence $\mathbf{o}_0 \leq \mathbf{o}_1 \leq \ldots$.

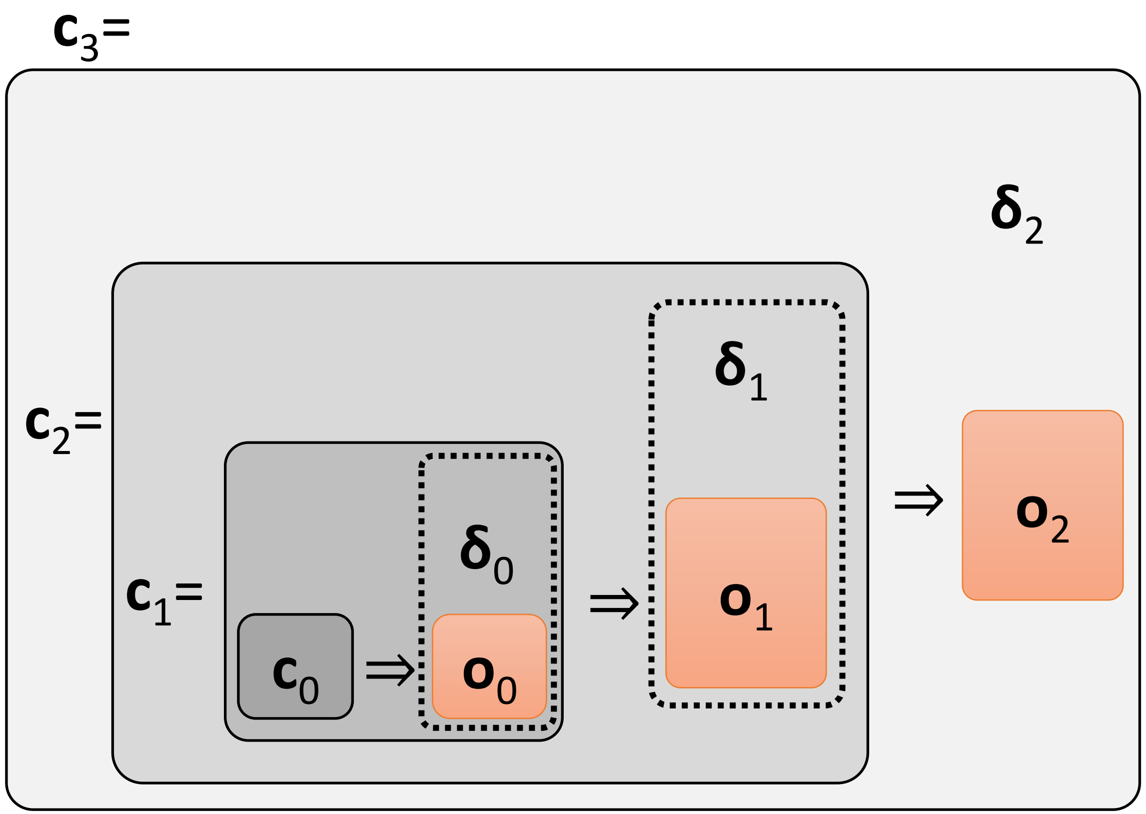


A pumping lemma

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

Proof:

1. By Dickson's Lemma there is infinite nondecreasing subsequence $\mathbf{c}_0 \leq \mathbf{c}_1 \leq \ldots$, each $\mathbf{c}_i \in A$. Let $\boldsymbol{\delta}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.
2. Define sequence of stable $\mathbf{o}_0$, $\mathbf{o}_1$, … inductively as follows.
3. <u>Base case</u>: $\mathbf{c}_0 \Rightarrow \mathbf{o}_0$ for some stable $\mathbf{o}_0$.
4. <u>Inductive case</u>: By additivity $\mathbf{c}_{i+1} = \mathbf{c}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_i + \boldsymbol{\delta}_i$.
5. By correctness $\mathbf{o}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_{i+1}$ for some stable $\mathbf{o}_{i+1}$.
6. By Dickson's Lemma pick infinite nondecreasing subsequence $\mathbf{o'}_0 \leq \mathbf{o'}_1 \leq \ldots$ of $\mathbf{o}_i$'s. For the sake of readability let's assume this is just the original sequence $\mathbf{o}_0 \leq \mathbf{o}_1 \leq \ldots$.
7. Let $\Gamma = \{ S \mid \lim_{i\to\infty} \mathbf{o}_i(S) = \infty \}$ (*species with unbounded counts*).

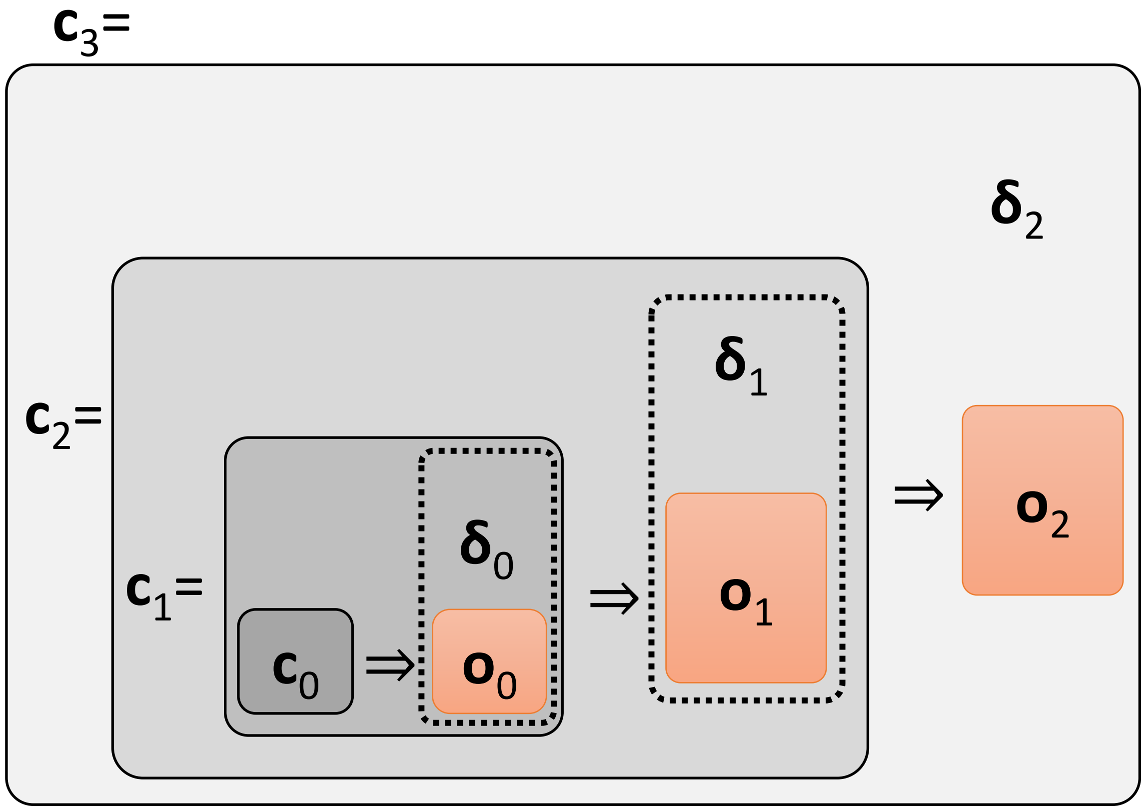


A pumping lemma

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

Proof:

1. By Dickson's Lemma there is infinite nondecreasing subsequence $\mathbf{c}_0 \leq \mathbf{c}_1 \leq \ldots$, each $\mathbf{c}_i \in A$. Let $\boldsymbol{\delta}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.
2. Define sequence of stable $\mathbf{o}_0$, $\mathbf{o}_1$, … inductively as follows.
3. Base case: $\mathbf{c}_0 \Rightarrow \mathbf{o}_0$ for some stable $\mathbf{o}_0$.
4. Inductive case: By additivity $\mathbf{c}_{i+1} = \mathbf{c}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_i + \boldsymbol{\delta}_i$.
5. By correctness $\mathbf{o}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_{i+1}$ for some stable $\mathbf{o}_{i+1}$.
6. By Dickson's Lemma pick infinite nondecreasing subsequence $\mathbf{o}'_0 \leq \mathbf{o}'_1 \leq \ldots$ of $\mathbf{o}_i$'s. For the sake of readability let's assume this is just the original sequence $\mathbf{o}_0 \leq \mathbf{o}_1 \leq \ldots$.
7. Let $\Gamma = \{ S \mid \lim_{i\to\infty} \mathbf{o}_i(S) = \infty \}$ (*species with unbounded counts*).
8. For large enough i, if $S \in \Gamma$, then $\mathbf{o}_i(S) \geq \tau$, and if $S \notin \Gamma$, then $\mathbf{o}_i(S) = c_S$ where c_S is the largest S ever gets in the $\mathbf{o}_i$'s.

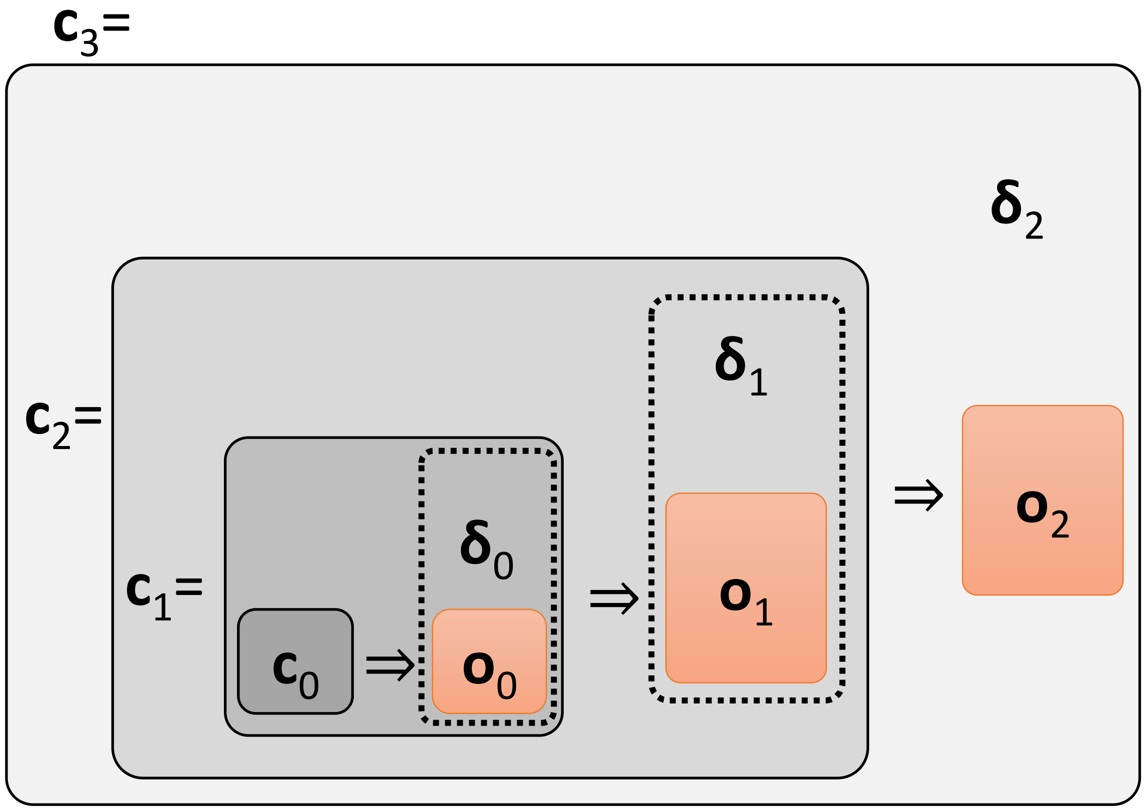


A pumping lemma

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

Proof:

1. By Dickson's Lemma there is infinite nondecreasing subsequence $\mathbf{c}_0 \leq \mathbf{c}_1 \leq \ldots$, each $\mathbf{c}_i \in A$. Let $\boldsymbol{\delta}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$.
2. Define sequence of stable $\mathbf{o}_0$, $\mathbf{o}_1$, … inductively as follows.
3. Base case: $\mathbf{c}_0 \Rightarrow \mathbf{o}_0$ for some stable $\mathbf{o}_0$.
4. Inductive case: By additivity $\mathbf{c}_{i+1} = \mathbf{c}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_i + \boldsymbol{\delta}_i$.
5. By correctness $\mathbf{o}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_{i+1}$ for some stable $\mathbf{o}_{i+1}$.
6. By Dickson's Lemma pick infinite nondecreasing subsequence $\mathbf{o}'_0 \leq \mathbf{o}'_1 \leq \ldots$ of $\mathbf{o}_i$'s. For the sake of readability let's assume this is just the original sequence $\mathbf{o}_0 \leq \mathbf{o}_1 \leq \ldots$.
7. Let $\Gamma = \{ S \mid \lim_{i\to\infty} \mathbf{o}_i(S) = \infty \}$ (*species with unbounded counts*).
8. For large enough i, if $S \in \Gamma$, then $\mathbf{o}_i(S) \geq \tau$, and if $S \notin \Gamma$, then $\mathbf{o}_i(S) = c_S$ where c_S is the largest S ever gets in the $\mathbf{o}_i$'s.
9. Then $\mathbf{o}_{i+1}(S) = \mathbf{o}_i(S)$ if $S \notin \Gamma$ and $\mathbf{o}_i(S) \geq \tau$ otherwise.



A pumping lemma (proof continued)

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

Proof: (continued)

1. Fix large enough i that $\mathbf{o}_{i+1}(S) = \mathbf{o}_i(S)$ if $S \notin \Gamma$ and $\mathbf{o}_i(S) \geq \tau$ otherwise.

A pumping lemma (proof continued)

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

Proof: (continued)

1. Fix large enough i that $\mathbf{o}_{i+1}(S) = \mathbf{o}_i(S)$ if $S \notin \Gamma$ and $\mathbf{o}_i(S) \geq \tau$ otherwise.
2. Write $\boldsymbol{\varepsilon} = \mathbf{o}_{i+1} - \mathbf{o}_i$.

A pumping lemma (proof continued)

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

Proof: (continued)

1. Fix large enough i that $\mathbf{o}_{i+1}(S) = \mathbf{o}_i(S)$ if $S \notin \Gamma$ and $\mathbf{o}_i(S) \geq \tau$ otherwise.
2. Write $\boldsymbol{\varepsilon} = \mathbf{o}_{i+1} - \mathbf{o}_i$.
3. Note that $\boldsymbol{\varepsilon}(S) > 0$ implies $\mathbf{o}_i(S) \geq \tau$.

A pumping lemma (proof continued)

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

Proof: (continued)

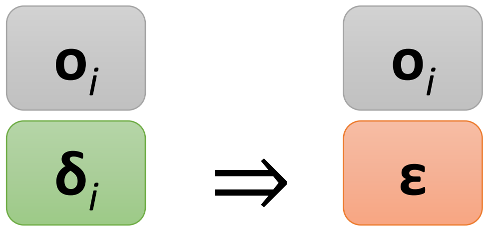
1. Fix large enough i that $\mathbf{o}_{i+1}(S) = \mathbf{o}_i(S)$ if $S \notin \Gamma$ and $\mathbf{o}_i(S) \geq \tau$ otherwise.
2. Write $\boldsymbol{\varepsilon} = \mathbf{o}_{i+1} - \mathbf{o}_i$.
3. Note that $\boldsymbol{\varepsilon}(S) > 0$ implies $\mathbf{o}_i(S) \geq \tau$.
4. Then $\mathbf{o}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_{i+1} = \mathbf{o}_i + \boldsymbol{\varepsilon}$, i.e., $\mathbf{o}_i$ is like a "catalyst" that transforms $\boldsymbol{\delta}_i$ into $\boldsymbol{\varepsilon}$.

A pumping lemma (proof continued)

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

Proof: (continued)

1. Fix large enough i that $\mathbf{o}_{i+1}(S) = \mathbf{o}_i(S)$ if $S \notin \Gamma$ and $\mathbf{o}_i(S) \geq \tau$ otherwise.
2. Write $\boldsymbol{\varepsilon} = \mathbf{o}_{i+1} - \mathbf{o}_i$.
3. Note that $\boldsymbol{\varepsilon}(S) > 0$ implies $\mathbf{o}_i(S) \geq \tau$.
4. Then $\mathbf{o}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_{i+1} = \mathbf{o}_i + \boldsymbol{\varepsilon}$, i.e., $\mathbf{o}_i$ is like a “catalyst” that transforms $\boldsymbol{\delta}_i$ into $\boldsymbol{\varepsilon}$.

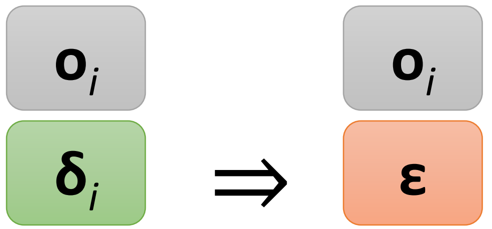


A pumping lemma (proof continued)

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

Proof: (continued)

1. Fix large enough i that $\mathbf{o}_{i+1}(S) = \mathbf{o}_i(S)$ if $S \notin \Gamma$ and $\mathbf{o}_i(S) \geq \tau$ otherwise.
2. Write $\boldsymbol{\varepsilon} = \mathbf{o}_{i+1} - \mathbf{o}_i$.
3. Note that $\boldsymbol{\varepsilon}(S) > 0$ implies $\mathbf{o}_i(S) \geq \tau$.
4. Then $\mathbf{o}_i+\boldsymbol{\delta}_i \Rightarrow \mathbf{o}_{i+1} = \mathbf{o}_i+\boldsymbol{\varepsilon}$, i.e., $\mathbf{o}_i$ is like a "catalyst" that transforms $\boldsymbol{\delta}_i$ into $\boldsymbol{\varepsilon}$.
5. Apply the same execution to n copies of $\boldsymbol{\delta}_i$: $\mathbf{o}_i+n\boldsymbol{\delta}_i \Rightarrow \mathbf{o}_i+n\boldsymbol{\varepsilon}$.

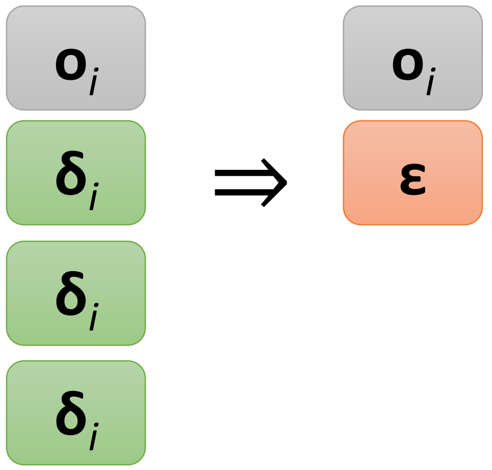


A pumping lemma (proof continued)

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

Proof: (continued)

1. Fix large enough i that $\mathbf{o}_{i+1}(S) = \mathbf{o}_i(S)$ if $S \notin \Gamma$ and $\mathbf{o}_i(S) \geq \tau$ otherwise.
2. Write $\boldsymbol{\varepsilon} = \mathbf{o}_{i+1} - \mathbf{o}_i$.
3. Note that $\boldsymbol{\varepsilon}(S) > 0$ implies $\mathbf{o}_i(S) \geq \tau$.
4. Then $\mathbf{o}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_{i+1} = \mathbf{o}_i + \boldsymbol{\varepsilon}$, i.e., $\mathbf{o}_i$ is like a "catalyst" that transforms $\boldsymbol{\delta}_i$ into $\boldsymbol{\varepsilon}$.
5. Apply the same execution to n copies of $\boldsymbol{\delta}_i$: $\mathbf{o}_i + n\boldsymbol{\delta}_i \Rightarrow \mathbf{o}_i + n\boldsymbol{\varepsilon}$.

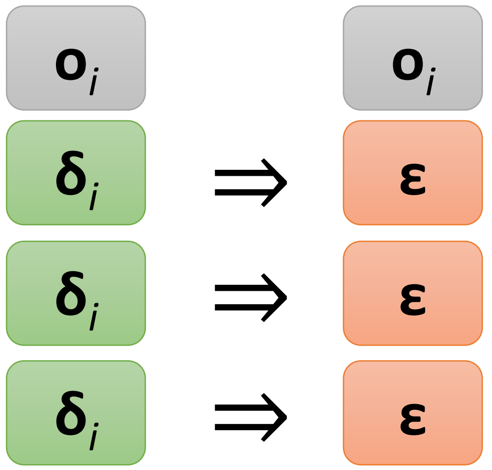


A pumping lemma (proof continued)

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

Proof: (continued)

1. Fix large enough i that $\mathbf{o}_{i+1}(S) = \mathbf{o}_i(S)$ if $S \notin \Gamma$ and $\mathbf{o}_i(S) \geq \tau$ otherwise.
2. Write $\boldsymbol{\varepsilon} = \mathbf{o}_{i+1} - \mathbf{o}_i$.
3. Note that $\boldsymbol{\varepsilon}(S) > 0$ implies $\mathbf{o}_i(S) \geq \tau$.
4. Then $\mathbf{o}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_{i+1} = \mathbf{o}_i + \boldsymbol{\varepsilon}$, i.e., $\mathbf{o}_i$ is like a "catalyst" that transforms $\boldsymbol{\delta}_i$ into $\boldsymbol{\varepsilon}$.
5. Apply the same execution to n copies of $\boldsymbol{\delta}_i$: $\mathbf{o}_i + n\boldsymbol{\delta}_i \Rightarrow \mathbf{o}_i + n\boldsymbol{\varepsilon}$.

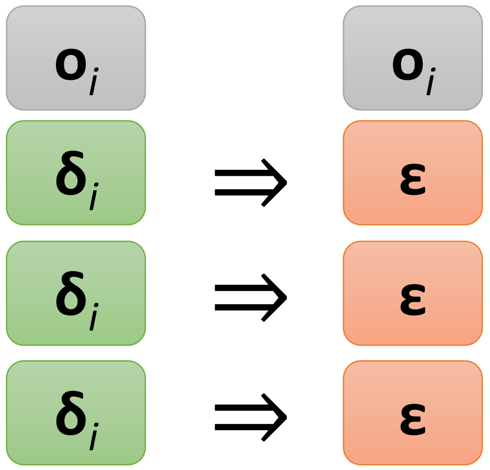


A pumping lemma (proof continued)

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

Proof: (continued)

1. Fix large enough i that $\mathbf{o}_{i+1}(S) = \mathbf{o}_i(S)$ if $S \notin \Gamma$ and $\mathbf{o}_i(S) \geq \tau$ otherwise.
2. Write $\boldsymbol{\varepsilon} = \mathbf{o}_{i+1} - \mathbf{o}_i$.
3. Note that $\boldsymbol{\varepsilon}(S) > 0$ implies $\mathbf{o}_i(S) \geq \tau$.
4. Then $\mathbf{o}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_{i+1} = \mathbf{o}_i + \boldsymbol{\varepsilon}$, i.e., $\mathbf{o}_i$ is like a "catalyst" that transforms $\boldsymbol{\delta}_i$ into $\boldsymbol{\varepsilon}$.
5. Apply the same execution to n copies of $\boldsymbol{\delta}_i$: $\mathbf{o}_i + n\boldsymbol{\delta}_i \Rightarrow \mathbf{o}_i + n\boldsymbol{\varepsilon}$.
6. **6.** $\mathbf{o}_i + n\boldsymbol{\varepsilon}$ is larger than $\mathbf{o}_i$ only on species S with count $\mathbf{o}_i(S) \geq \tau$.

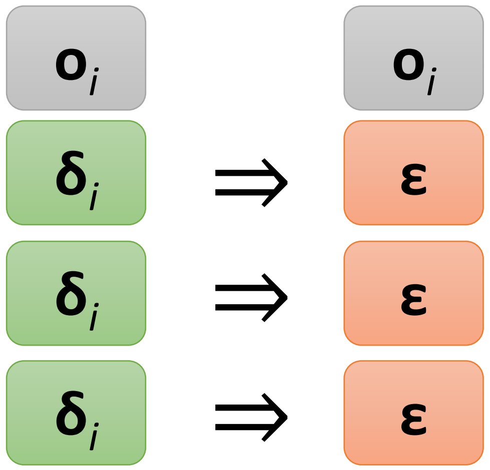


A pumping lemma (proof continued)

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

Proof: (continued)

1. Fix large enough i that $\mathbf{o}_{i+1}(S) = \mathbf{o}_i(S)$ if $S \notin \Gamma$ and $\mathbf{o}_i(S) \geq \tau$ otherwise.
2. Write $\boldsymbol{\varepsilon} = \mathbf{o}_{i+1} - \mathbf{o}_i$.
3. Note that $\boldsymbol{\varepsilon}(S) > 0$ implies $\mathbf{o}_i(S) \geq \tau$.
4. Then $\mathbf{o}_i + \boldsymbol{\delta}_i \Rightarrow \mathbf{o}_{i+1} = \mathbf{o}_i + \boldsymbol{\varepsilon}$, i.e., $\mathbf{o}_i$ is like a "catalyst" that transforms $\boldsymbol{\delta}_i$ into $\boldsymbol{\varepsilon}$.
5. Apply the same execution to n copies of $\boldsymbol{\delta}_i$: $\mathbf{o}_i + n\boldsymbol{\delta}_i \Rightarrow \mathbf{o}_i + n\boldsymbol{\varepsilon}$.
6. $\mathbf{o}_i + n\boldsymbol{\varepsilon}$ is larger than $\mathbf{o}_i$ only on species S with count $\mathbf{o}_i(S) \geq \tau$.
7. By closure of stable configurations upwards for "already large" species, since $\mathbf{o}_i$ is stable, $\mathbf{o}_i + n\boldsymbol{\varepsilon}$ is also stable, with the same output YES, since they have the same species present.

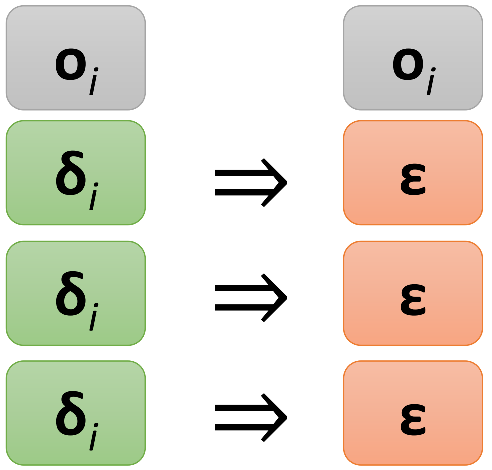


A pumping lemma (proof continued)

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

Proof: (continued)

1. Fix large enough i that $\mathbf{o}_{i+1}(S) = \mathbf{o}_i(S)$ if $S \notin \Gamma$ and $\mathbf{o}_i(S) \geq \tau$ otherwise.
2. Write $\boldsymbol{\varepsilon} = \mathbf{o}_{i+1} - \mathbf{o}_i$.
3. Note that $\boldsymbol{\varepsilon}(S) > 0$ implies $\mathbf{o}_i(S) \geq \tau$.
4. Then $\mathbf{o}_i+\boldsymbol{\delta}_i \Rightarrow \mathbf{o}_{i+1} = \mathbf{o}_i+\boldsymbol{\varepsilon}$, i.e., $\mathbf{o}_i$ is like a "catalyst" that transforms $\boldsymbol{\delta}_i$ into $\boldsymbol{\varepsilon}$.
5. Apply the same execution to n copies of $\boldsymbol{\delta}_i$: $\mathbf{o}_i+n\boldsymbol{\delta}_i \Rightarrow \mathbf{o}_i+n\boldsymbol{\varepsilon}$.
6. $\mathbf{o}_i+n\boldsymbol{\varepsilon}$ is larger than $\mathbf{o}_i$ only on species S with count $\mathbf{o}_i(S) \geq \tau$.
7. By closure of stable configurations upwards for "already large" species, since $\mathbf{o}_i$ is stable, $\mathbf{o}_i+n\boldsymbol{\varepsilon}$ is also stable, with the same output YES, since they have the same species present.
8. In other words, we can reach from $\mathbf{c}_i+n\boldsymbol{\delta}_i$ to a stable YES configuration, so $\mathbf{c}_i+n\boldsymbol{\delta}_i \in A$ for all $n \in \mathbb{N}$.

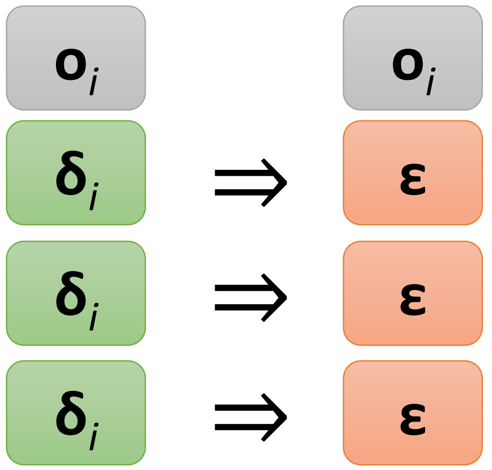


A pumping lemma (proof continued)

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

Proof: (continued)

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



Impossibility of stably deciding squaring set

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

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

Impossibility of stably deciding squaring set

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

Theorem: The "squaring set" $S = \{ (x,y) \mid x^2=y \}$ is <u>not</u> stably decidable by any CRN.

Proof:

1. By our Pumping Lemma, there are points $\mathbf{c}=(x,x^2)$ and $\mathbf{d}=(z,z^2)$, $x < z$, such that, letting $\boldsymbol{\delta} = \mathbf{d}-\mathbf{c}$, for all $n \in \mathbb{N}$, $\mathbf{c}+n\boldsymbol{\delta} \in S$.

Impossibility of stably deciding squaring set

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

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

Proof:

1. By our Pumping Lemma, there are points $\mathbf{c}=(x,x^2)$ and $\mathbf{d}=(z,z^2)$, $x < z$, such that, letting $\boldsymbol{\delta} = \mathbf{d}-\mathbf{c}$, for all $n \in \mathbb{N}$, $\mathbf{c}+n\boldsymbol{\delta} \in S$.
2. Claim: the point $\mathbf{c}+2\boldsymbol{\delta} \notin S$, contradicting our Pumping Lemma.

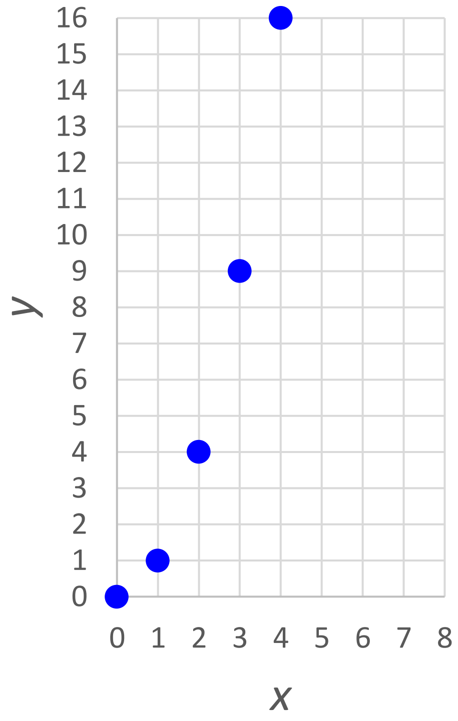
Impossibility of stably deciding squaring set

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

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

Proof:

1. By our Pumping Lemma, there are points $\mathbf{c}=(x,x^2)$ and $\mathbf{d}=(z,z^2)$, $x < z$, such that, letting $\boldsymbol{\delta} = \mathbf{d}-\mathbf{c}$, for all $n \in \mathbb{N}$, $\mathbf{c}+n\boldsymbol{\delta} \in S$.
2. Claim: the point $\mathbf{c}+2\boldsymbol{\delta} \notin S$, contradicting our Pumping Lemma.
3. Proof: by picture. (straight line intersects a parabola at ≤ 2 points)



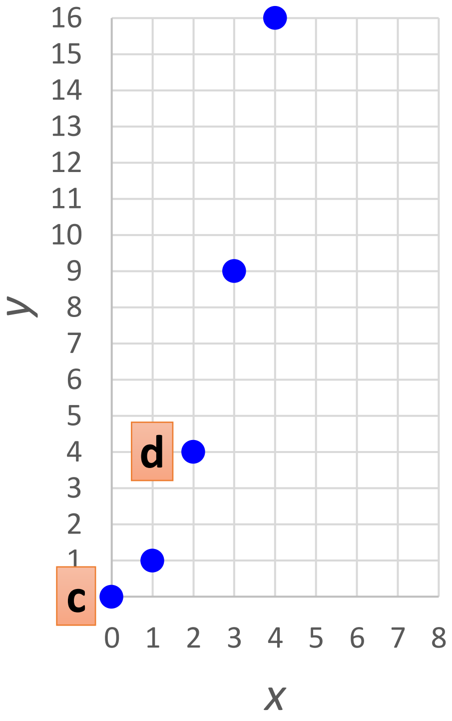
Impossibility of stably deciding squaring set

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

Theorem: The "squaring set" $S = \{ (x,y) \mid x^2=y \}$ is <u>not</u> stably decidable by any CRN.

Proof:

1. By our Pumping Lemma, there are points $\mathbf{c}=(x,x^2)$ and $\mathbf{d}=(z,z^2)$, $x < z$, such that, letting $\boldsymbol{\delta} = \mathbf{d}-\mathbf{c}$, for all $n \in \mathbb{N}$, $\mathbf{c}+n\boldsymbol{\delta} \in S$.
2. <u>Claim</u>: the point $\mathbf{c}+2\boldsymbol{\delta} \notin S$, contradicting our Pumping Lemma.
3. Proof: by picture. (straight line intersects a parabola at ≤ 2 points)



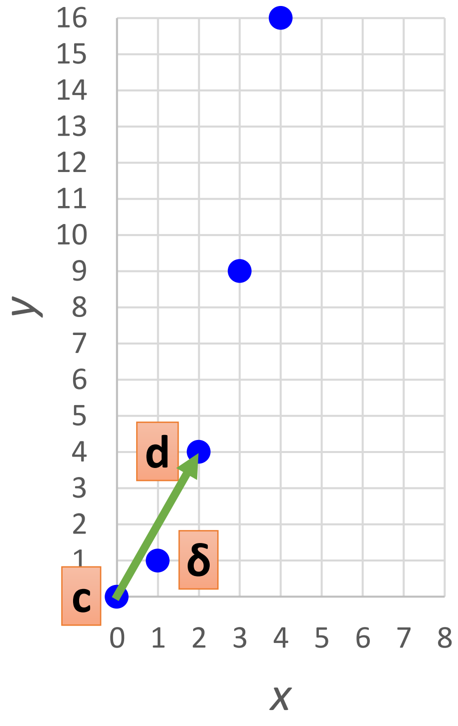
Impossibility of stably deciding squaring set

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

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

Proof:

1. By our Pumping Lemma, there are points **c**=(x,x^2) and **d**=(z,z^2), $x < z$, such that, letting $\boldsymbol{\delta}$ = **d**–**c**, for all $n \in \mathbb{N}$, **c**+$n\boldsymbol{\delta} \in S$.
2. Claim: the point **c**+2$\boldsymbol{\delta} \notin S$, contradicting our Pumping Lemma.
3. Proof: by picture. (straight line intersects a parabola at ≤ 2 points)



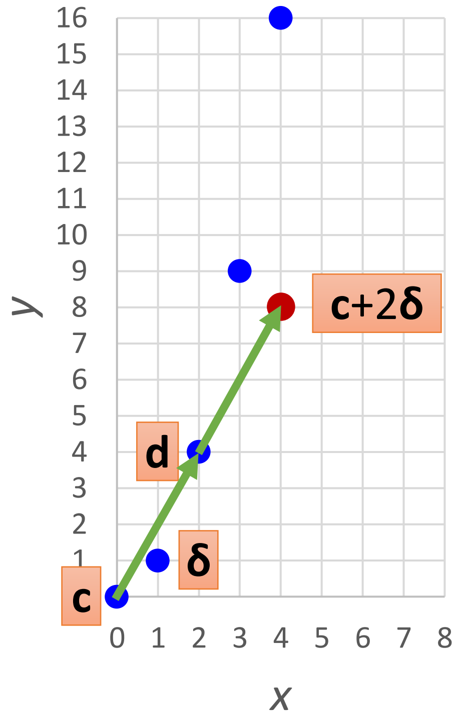
Impossibility of stably deciding squaring set

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

Theorem: The "squaring set" $S = \{ (x,y) \mid x^2=y \}$ is <u>not</u> stably decidable by any CRN.

Proof:

1. By our Pumping Lemma, there are points **c**=(x,x^2) and **d**=(z,z^2), $x < z$, such that, letting $\boldsymbol{\delta}$ = **d**–**c**, for all $n \in \mathbb{N}$, **c**+$n\boldsymbol{\delta} \in S$.
2. <u>Claim</u>: the point **c**+2$\boldsymbol{\delta} \notin S$, contradicting our Pumping Lemma.
3. Proof: by picture. (straight line intersects a parabola at ≤ 2 points)



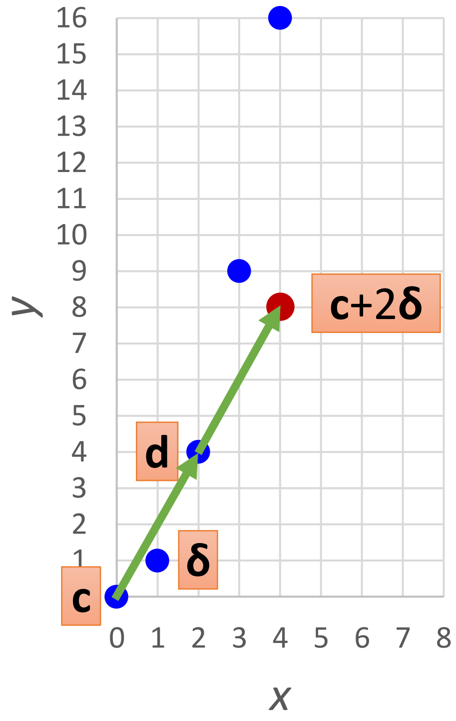
Impossibility of stably deciding squaring set

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

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

Proof:

1. By our Pumping Lemma, there are points $\mathbf{c}=(x,x^2)$ and $\mathbf{d}=(z,z^2)$, $x < z$, such that, letting $\boldsymbol{\delta} = \mathbf{d}-\mathbf{c}$, for all $n \in \mathbb{N}$, $\mathbf{c}+n\boldsymbol{\delta} \in S$.
2. Claim: the point $\mathbf{c}+2\boldsymbol{\delta} \notin S$, contradicting our Pumping Lemma.
3. Proof: by picture. (straight line intersects a parabola at ≤ 2 points)
4. Formally, suppose otherwise: $\mathbf{c}+2\boldsymbol{\delta} = (2z-x,\ 2z^2-x^2) \in S$.



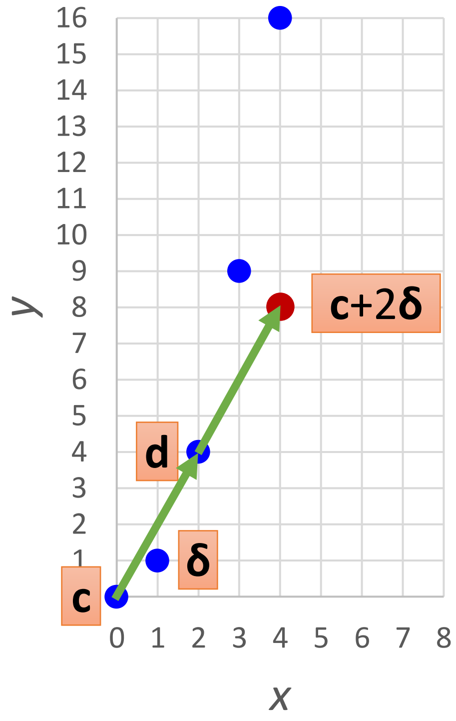
Impossibility of stably deciding squaring set

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

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

Proof:

1. By our Pumping Lemma, there are points **c**=(x,x^2) and **d**=(z,z^2), $x < z$, such that, letting $\boldsymbol{\delta}$ = **d**–**c**, for all $n \in \mathbb{N}$, **c**+$n\boldsymbol{\delta} \in S$.
2. Claim: the point **c**+2$\boldsymbol{\delta} \notin S$, contradicting our Pumping Lemma.
3. Proof: by picture. (straight line intersects a parabola at ≤ 2 points)
4. Formally, suppose otherwise: **c**+2$\boldsymbol{\delta}$ = $(2z–x, 2z^2–x^2) \in S$.
5. Then $(2z–x)^2 = (2z^2–x^2)$, so



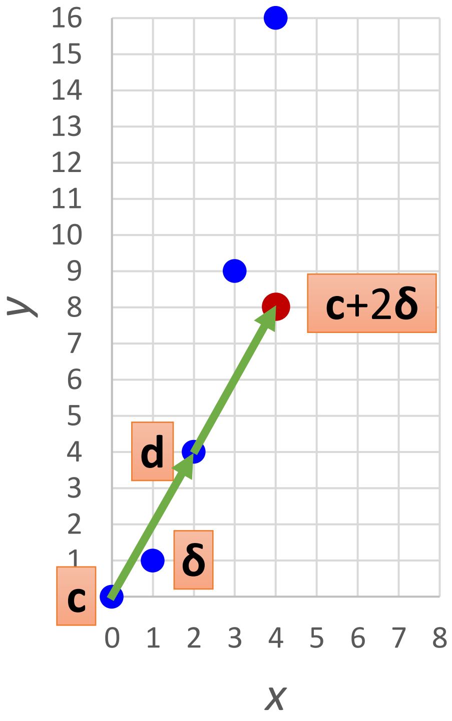
Impossibility of stably deciding squaring set

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

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

Proof:

1. By our Pumping Lemma, there are points **c**=(x,x^2) and **d**=(z,z^2), $x < z$, such that, letting $\boldsymbol{\delta}$ = **d**–**c**, for all $n \in \mathbb{N}$, **c**+$n\boldsymbol{\delta} \in S$.
2. Claim: the point **c**+2$\boldsymbol{\delta} \notin S$, contradicting our Pumping Lemma.
3. Proof: by picture. (straight line intersects a parabola at ≤ 2 points)
4. Formally, suppose otherwise: **c**+2$\boldsymbol{\delta}$ = $(2z–x, 2z^2–x^2) \in S$.
5. Then $(2z–x)^2 = (2z^2–x^2)$, so
   $0 \quad = (2z–x)^2 – (2z^2–x^2)$



Impossibility of stably deciding squaring set

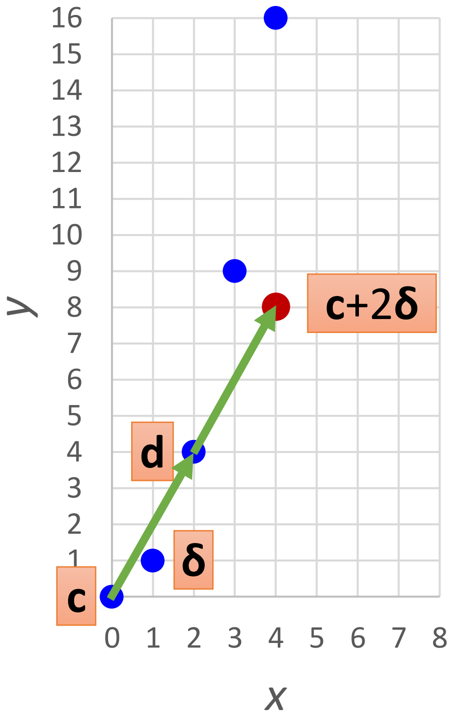
Pumping Lemma : Suppose a CRN stably decides infinite set $A \subseteq \mathbb{N}^d$.
Then there are **c**<**d** such that, letting $\boldsymbol{\delta}$ = **d**–**c**, for all $n \in \mathbb{N}$, $\mathbf{c}+n\boldsymbol{\delta} \in A$.

Theorem: The "squaring set" $S = \{ (x,y) \mid x^2=y \}$ is <u>not</u> stably decidable by any CRN.

Proof:

1. By our Pumping Lemma, there are points $\mathbf{c}=(x,x^2)$ and $\mathbf{d}=(z,z^2)$, $x < z$, such that, letting $\boldsymbol{\delta} = \mathbf{d}-\mathbf{c}$, for all $n \in \mathbb{N}$, $\mathbf{c}+n\boldsymbol{\delta} \in S$.
2. <u>Claim</u>: the point $\mathbf{c}+2\boldsymbol{\delta} \notin S$, contradicting our Pumping Lemma.
3. Proof: by picture. (straight line intersects a parabola at ≤ 2 points)
4. Formally, suppose otherwise: $\mathbf{c}+2\boldsymbol{\delta} = (2z-x,\ 2z^2-x^2) \in S$.
5. Then $(2z-x)^2 = (2z^2-x^2)$, so

$$
\begin{aligned}
0 &= (2z-x)^2 - (2z^2-x^2) \\
&= (4z^2-4xz+x^2) - (2z^2-x^2)
\end{aligned}
$$



Impossibility of stably deciding squaring set

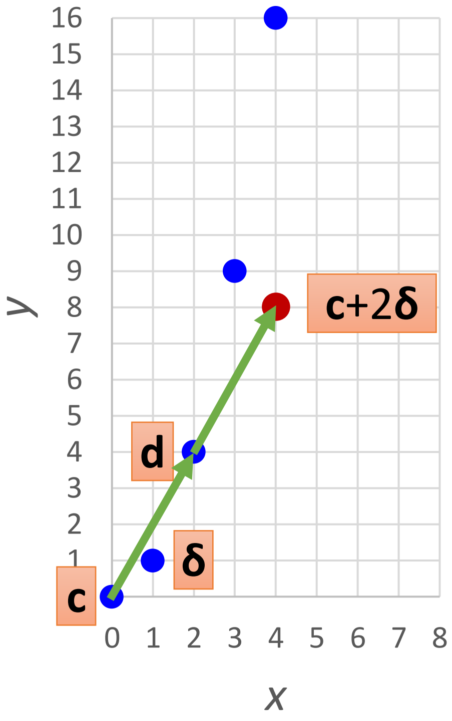
Pumping Lemma : Suppose a CRN stably decides infinite set $A \subseteq \mathbb{N}^d$. Then there are **c**<**d** such that, letting $\boldsymbol{\delta}$ = **d**–**c**, for all $n \in \mathbb{N}$, **c**+$n\boldsymbol{\delta} \in A$.

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

Proof:

1. By our Pumping Lemma, there are points $\mathbf{c}=(x,x^2)$ and $\mathbf{d}=(z,z^2)$, $x < z$, such that, letting $\boldsymbol{\delta} = \mathbf{d}-\mathbf{c}$, for all $n \in \mathbb{N}$, $\mathbf{c}+n\boldsymbol{\delta} \in S$.
2. Claim: the point $\mathbf{c}+2\boldsymbol{\delta} \notin S$, contradicting our Pumping Lemma.
3. Proof: by picture. (straight line intersects a parabola at ≤ 2 points)
4. Formally, suppose otherwise: $\mathbf{c}+2\boldsymbol{\delta} = (2z-x,\ 2z^2-x^2) \in S$.
5. Then $(2z-x)^2 = (2z^2-x^2)$, so

$$\begin{aligned} 0 &= (2z-x)^2 - (2z^2-x^2) \\ &= (4z^2-4xz+x^2) - (2z^2-x^2) \\ &= 2z^2 - 4xz + 2x^2 \end{aligned}$$



Impossibility of stably deciding squaring set

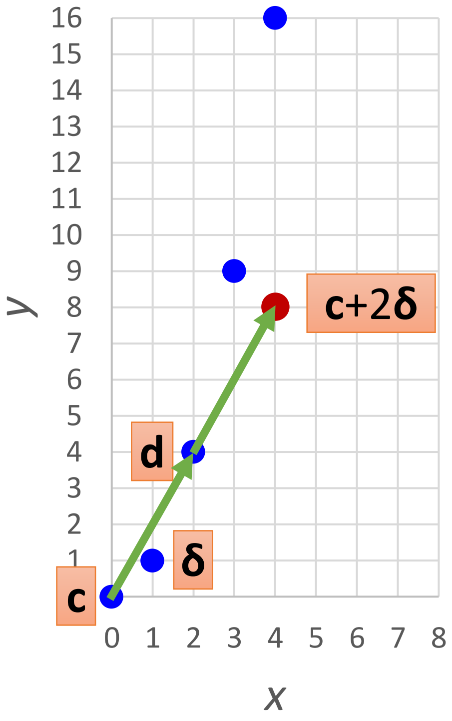
Pumping Lemma : Suppose a CRN stably decides infinite set $A \subseteq \mathbb{N}^d$. Then there are **c**<**d** such that, letting $\boldsymbol{\delta}$ = **d**–**c**, for all $n \in \mathbb{N}$, **c**+$n\boldsymbol{\delta} \in A$.

Theorem: The "squaring set" $S = \{ (x,y) \mid x^2=y \}$ is <u>not</u> stably decidable by any CRN.

Proof:

1. By our Pumping Lemma, there are points **c**=(x,x^2) and **d**=(z,z^2), $x < z$, such that, letting $\boldsymbol{\delta}$ = **d**–**c**, for all $n \in \mathbb{N}$, **c**+$n\boldsymbol{\delta} \in S$.
2. <u>Claim</u>: the point **c**+2$\boldsymbol{\delta} \notin S$, contradicting our Pumping Lemma.
3. Proof: by picture. (straight line intersects a parabola at ≤ 2 points)
4. Formally, suppose otherwise: **c**+2$\boldsymbol{\delta}$ = $(2z–x,\ 2z^2–x^2) \in S$.
5. Then $(2z–x)^2 = (2z^2–x^2)$, so

$$\begin{aligned} 0 &= (2z-x)^2 - (2z^2-x^2) \\ &= (4z^2-4xz+x^2) - (2z^2-x^2) \\ &= 2z^2 - 4xz + 2x^2 \\ &= 2(z-x)^2, \text{ which contradicts } x \neq z. \textbf{ QED} \end{aligned}$$



Limits of *efficient* stable computation

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

$a \to y+y$

$b \to y+y+y$

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

[Angluin, Aspnes, Eisenstat, Fast computation by population protocols with a leader, *DISC* 2006]
[Chen, Doty, Soloveichik, Deterministic function computation with chemical reaction networks, *DNA* 2012]

Known time lower bounds: leader election/majority

Leader election

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

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

Majority (and other "explicit" predicates)

Majority (and many other "explicit" predicates such as equality) require $\Omega(n$ / polylog $n)$ time, even with up to ½ log log n states.*

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

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

**[Alistarh, Aspnes, Gelashvili, SODA 2018]: "*output dominance*" = changing positive counts of states in a stable configuration leaves it able to reach a stable configuration with the same output

Known time lower bounds: “most” predicates/functions

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

[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 $\Omega(n)$ 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 $\geq m$

[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 $\Omega(n)$ time to compute
- Definition: φ: $\mathbb{N}^k \rightarrow \{Y,N\}$ 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: $\mathbb{N}^k \rightarrow \mathbb{N}$ 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 $\mathbb{N}^k$"

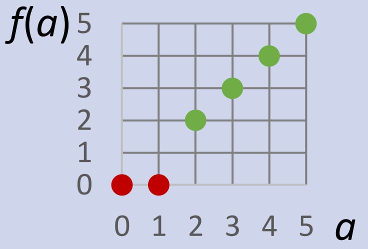
[Belleville, Doty, Soloveichik, *Hardness of computing and approximating predicates and functions with leaderless population protocols*, *ICALP* 2017]

Known time lower bounds: "most" predicates/functions

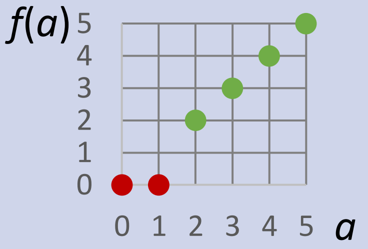
- Informal: "most" semilinear predicates and functions not known to be computable in $o(n)$ time, actually require at least $\Omega(n)$ time to compute
- Definition: φ: $\mathbb{N}^k \to \{Y,N\}$ is eventually constant if there is $m \in \mathbb{N}$ so that $\varphi(\boldsymbol{a}) = \varphi(\boldsymbol{b})$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

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

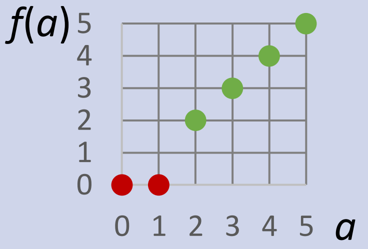
What is currently known/unknown

	Predicates	**Functions**
computable in $O(\log n)$ time	detection (constant unless changing between 0 and positive) $a>0$ **AND** $(b>0$ **OR** $c=0)$	$\mathbb{N}$-linear $3a + b + 2c$
not computable in less than $\Omega(n)$ 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 *all* positive values $a>1$?	eventually $\mathbb{N}$-linear but not $\mathbb{N}$-linear $f(a) = \begin{cases} a & \text{if } a>1, \\ 0 & \text{otherwise} \end{cases}$ 

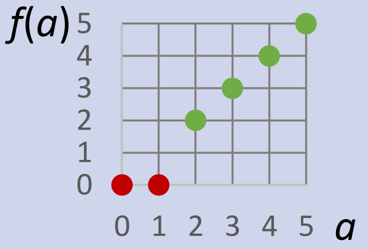
What is currently known/unknown

	Predicates	Functions
computable in $O(\log n)$ time	detection (constant unless changing between 0 and positive) $a>0$ **AND** $(b>0$ **OR** $c=0)$	$\mathbb{N}$-linear $3a + b + 2c$
not computable in less than $\Omega(n)$ 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 *all* 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 (constant unless changing between 0 and positive) $a>0$ **AND** ($b>0$ **OR** $c=0$)	$\mathbb{N}$-linear $3a + b + 2c$
not computable in less than $\Omega(n)$ 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 *all* 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 (constant unless changing between 0 and positive) $a>0$ **AND** $(b>0$ **OR** $c=0)$	$\mathbb{N}$-linear $3a + b + 2c$
not computable in less than $\Omega(n)$ 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 *all* 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 slides

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

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

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*)

<u>parity</u>: $\varphi(a)$ = "a is odd"

<u>without</u> a leader

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

<u>with</u> 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

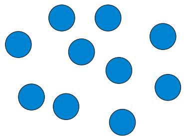
$$L_e + A \to L_o$$
$$L_o + A \to L_e$$

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]

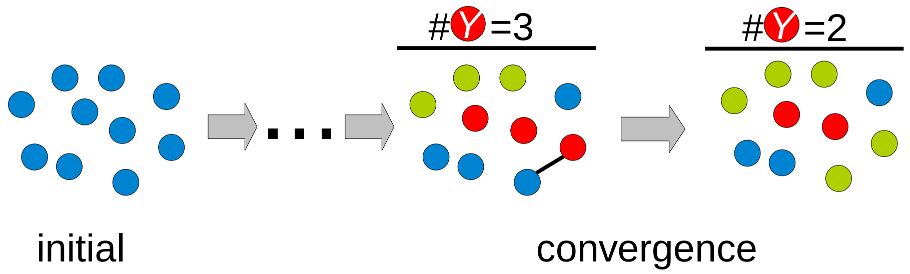
Convergence vs stabilization and leader vs anarchy

Convergence vs stabilization and leader vs anarchy

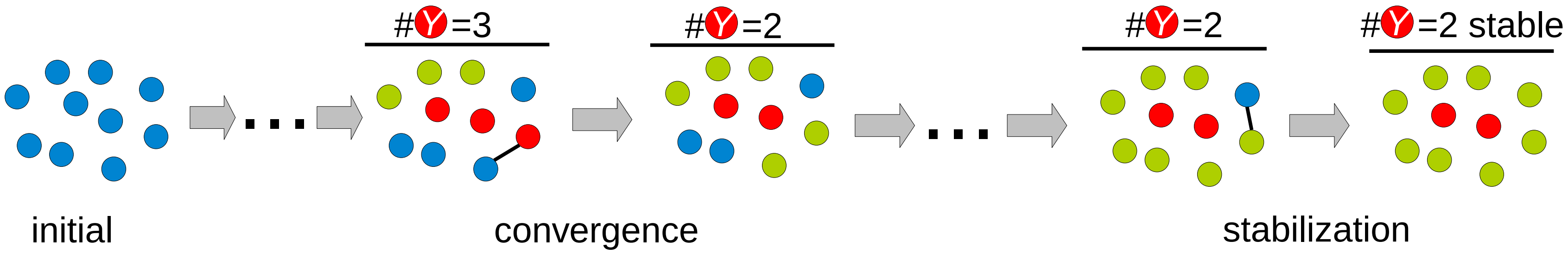


initial

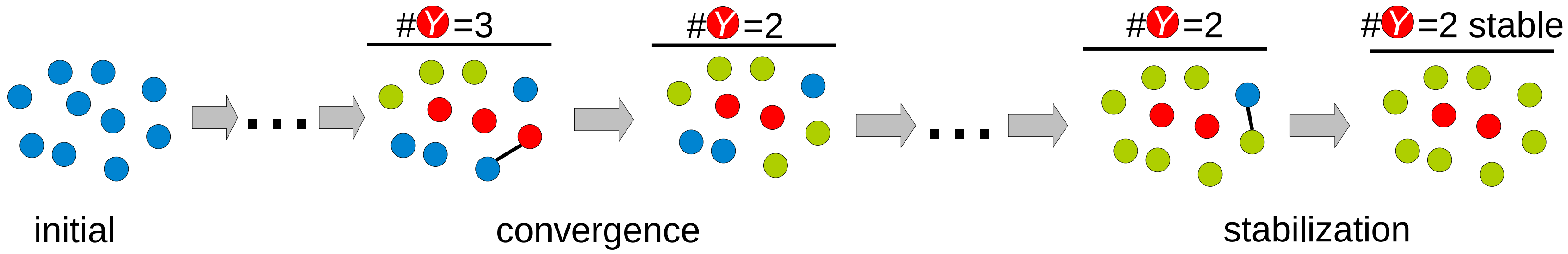
Convergence vs stabilization and leader vs anarchy



Convergence vs stabilization and leader vs anarchy

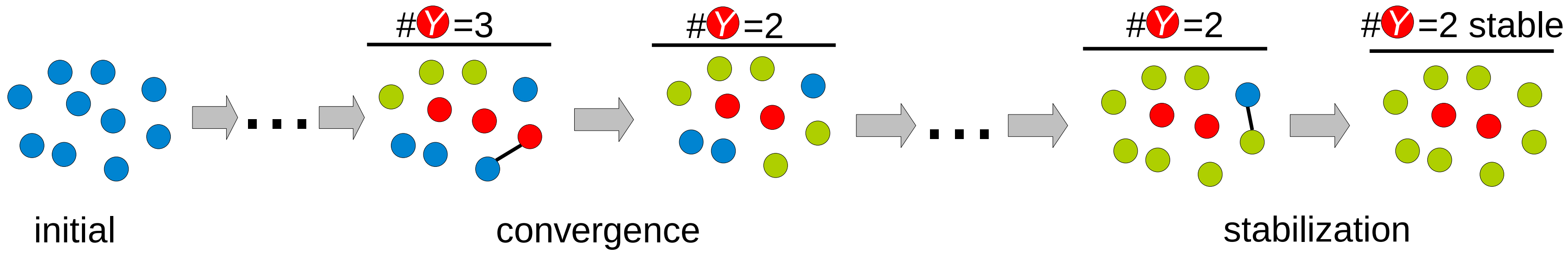


Convergence vs stabilization and leader vs anarchy



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

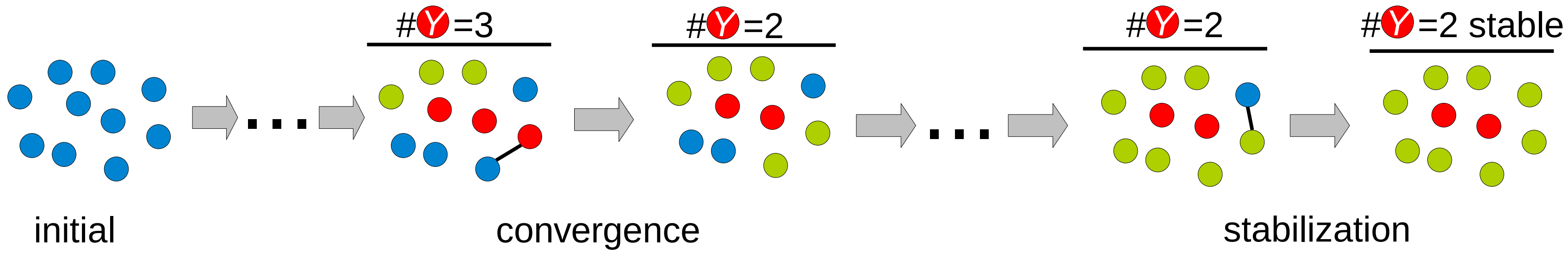
Convergence vs stabilization and leader vs anarchy



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

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

Convergence vs stabilization and leader vs anarchy

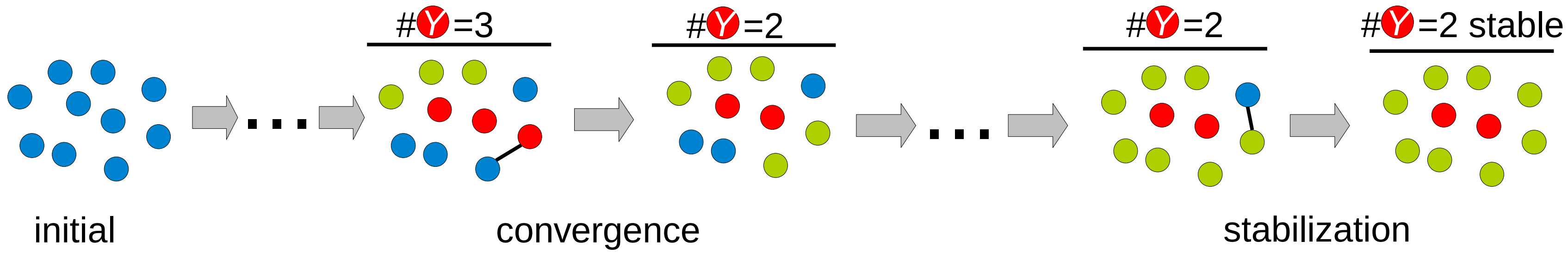


Theorem: Without a leader, all non-eventually constant predicates and non-eventually-$\mathbb{N}$-linear functions require at least $\Omega(n)$ stabilization time. [Belleville, Doty, Soloveichik, *ICALP* 2017]

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

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

Convergence vs stabilization and leader vs anarchy



Theorem: Without a leader, all non-eventually constant predicates and non-eventually-ℕ-linear functions require at least Ω(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-ℕ-linear functions require at least Ω(n) stabilization time.

False conjecture: Without a leader, all non-detection predicates and non-ℕ-linear functions require at least Ω(n) convergence time.

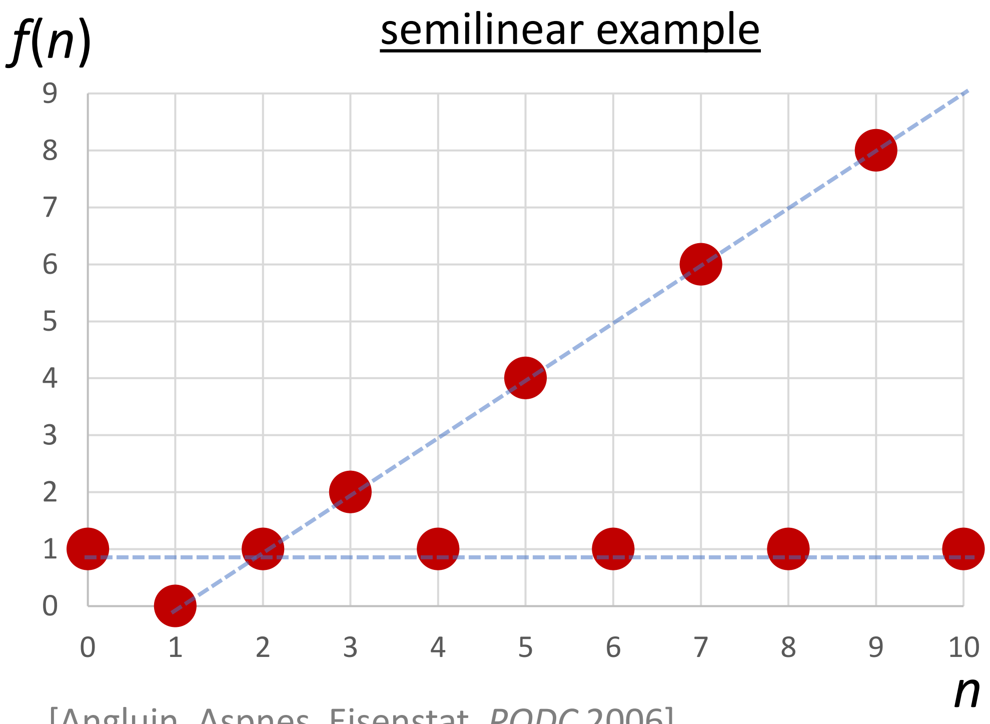
[resolved negatively by Kosowski, Uznański, *Population Protocols are Fast* , PODC Brief Announcement 2018]

What if we use real-valued concentrations?

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

What if we use real-valued concentrations?

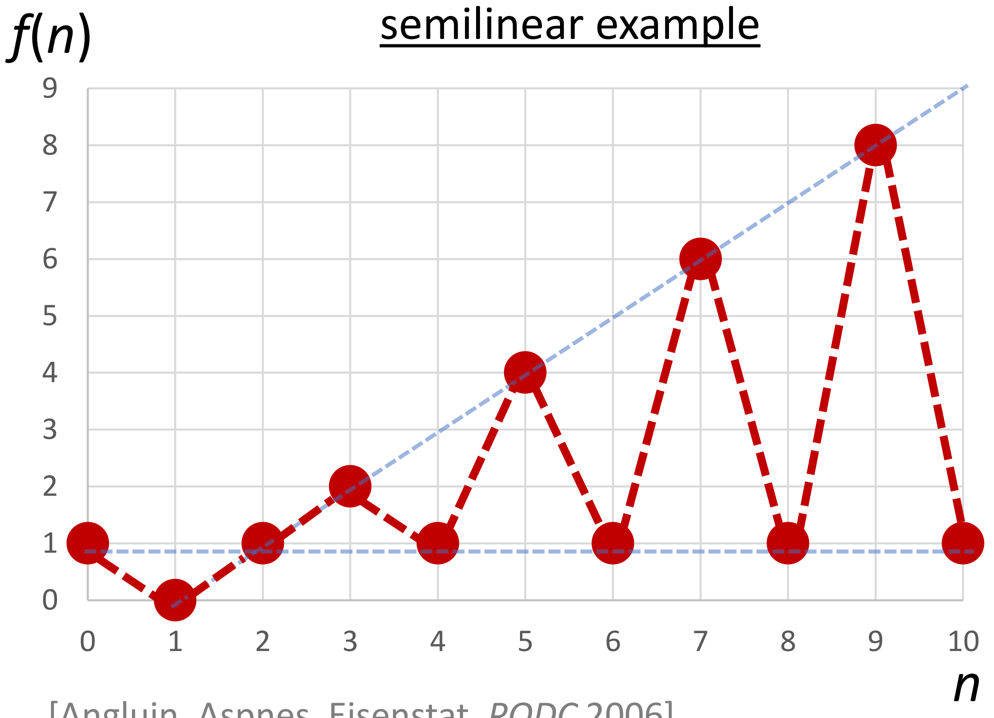
Theorem: A function is stably computable by an integer-valued chemical reaction network if and only if it is semilinear.



[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.

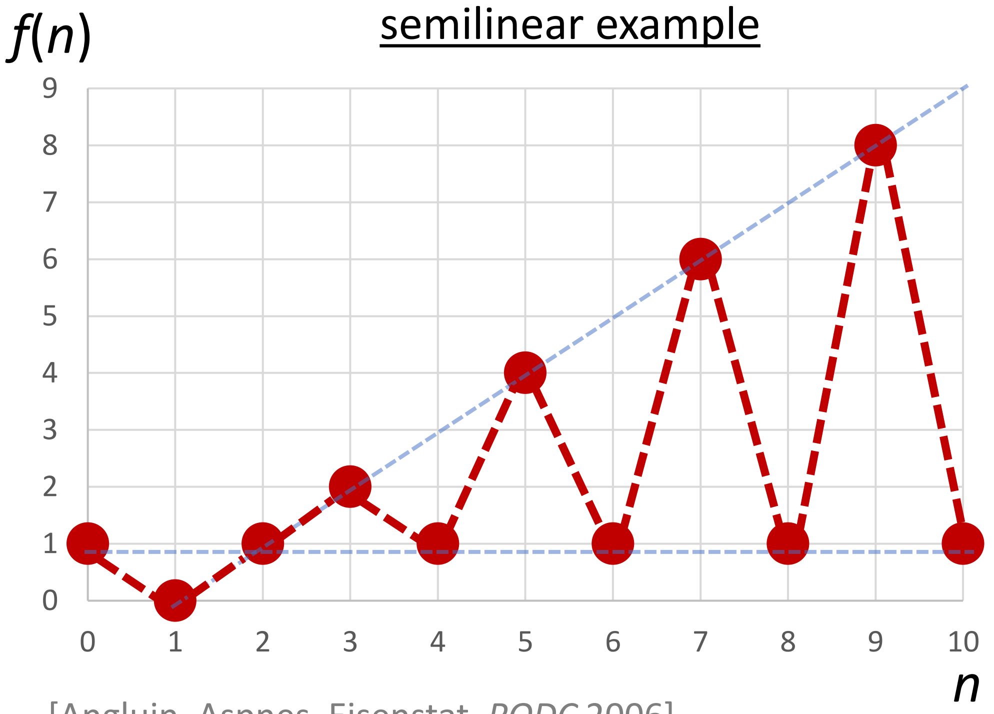


[Angluin, Aspnes, Eisenstat, *PODC* 2006]
[Chen, Doty, Soloveichik, *DNA* 2012]

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.

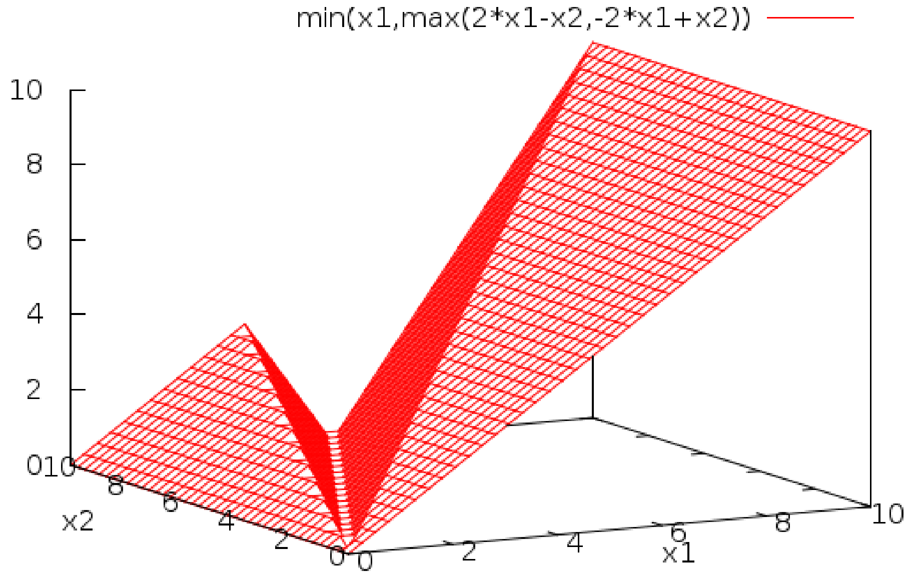
<u>semilinear example</u>



[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.

<u>continuous piecewise linear example</u>



[Chen, Doty, Reeves, Soloveichik, *JACM* 2023]

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

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

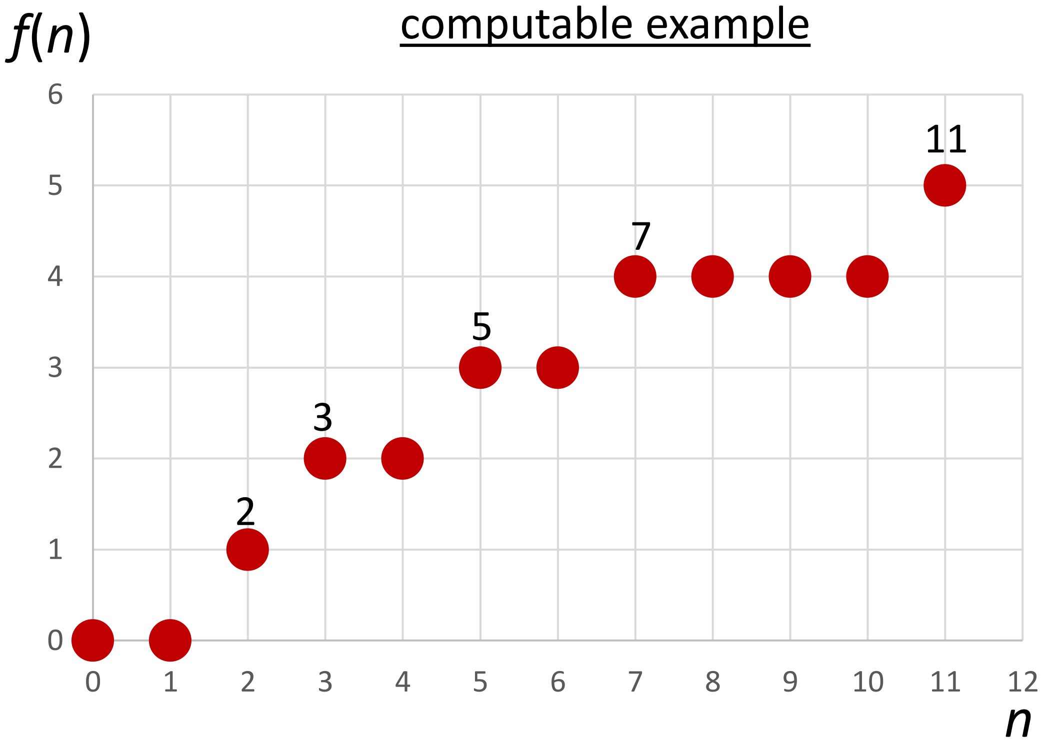
[Soloveichik, Cook, Bruck, Winfree, *Natural Computing* 2008]

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]

computable example



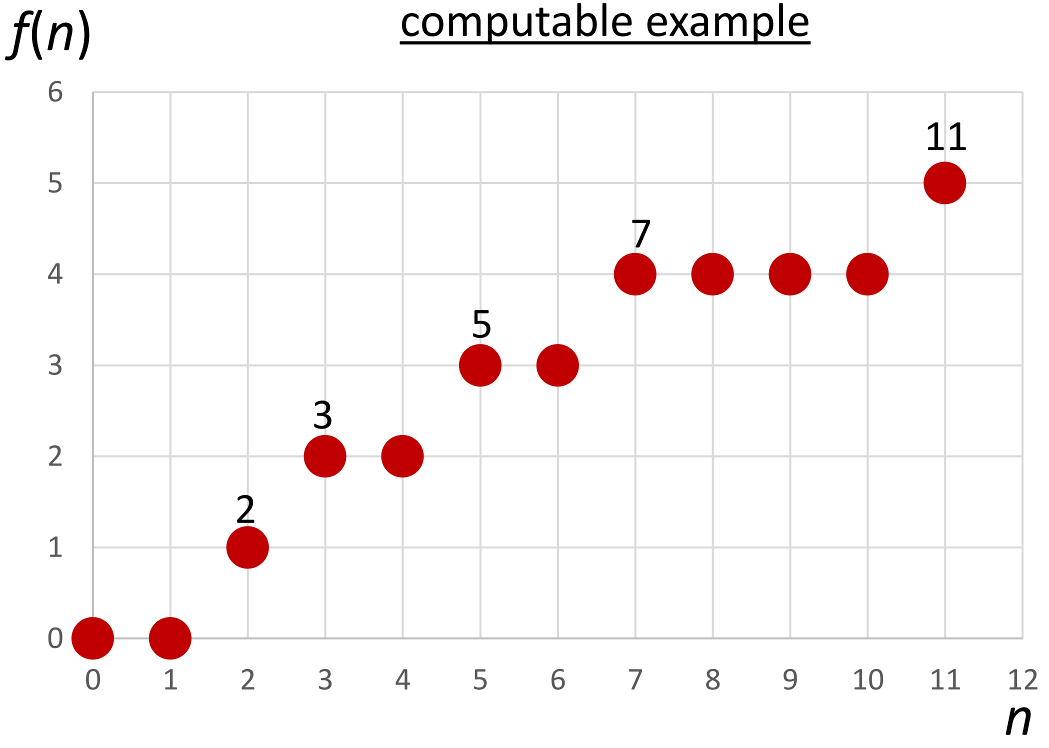
What if we allow a small probability of error? (i.e., allow reaction rates to influence outcome)

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) …

computable example



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

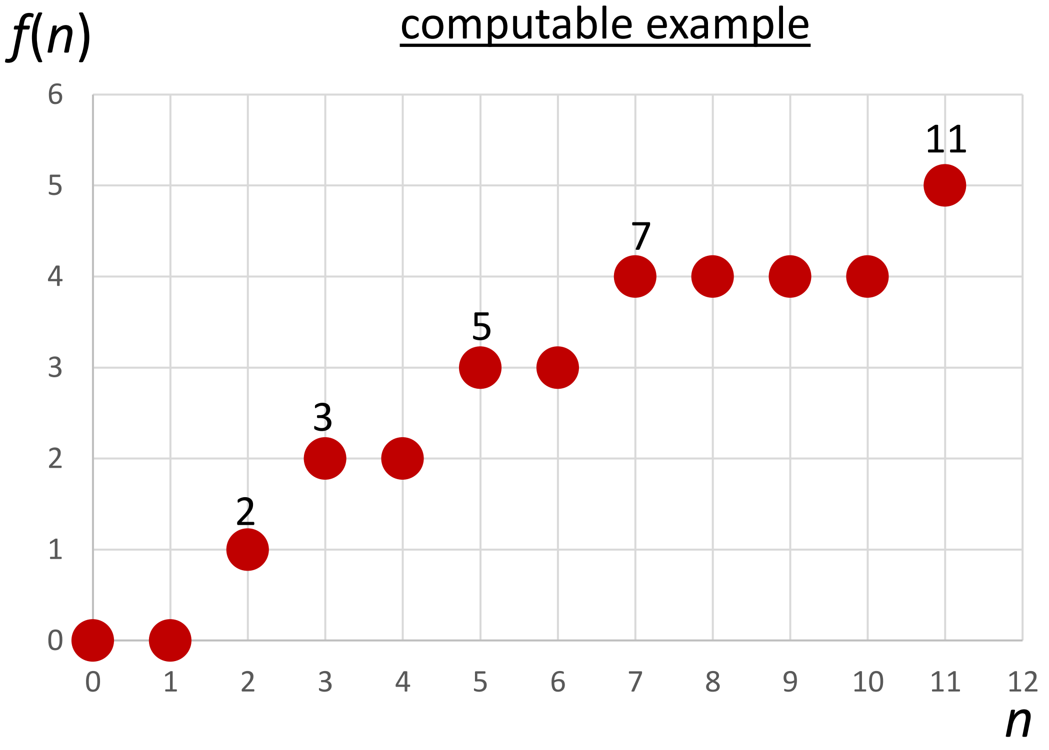
Theorem: A function is computable with probability of error < 1% by an integer-valued chemical reaction network if and only if it is computable by any algorithm whatsoever…

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

… "efficiently" (polynomial-time slowdown) …

… **if** we have an initial leader.

computable example



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

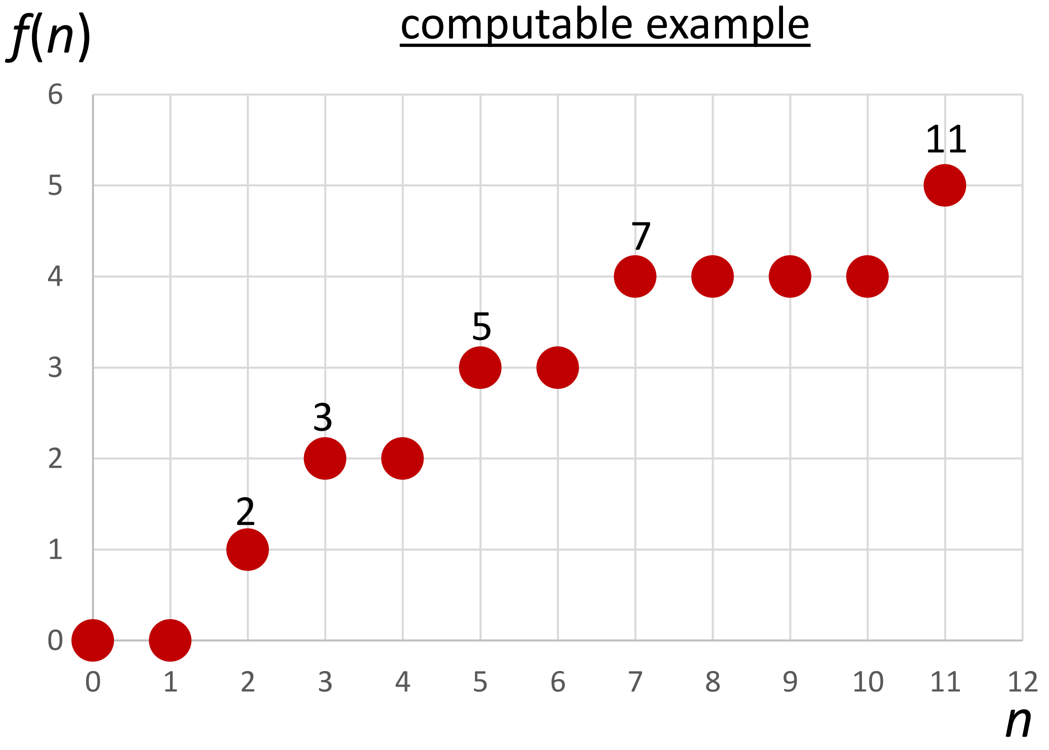
Theorem: A function is computable with probability of error < 1% by an integer-valued chemical reaction network if and only if it is computable by any algorithm whatsoever…

[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)

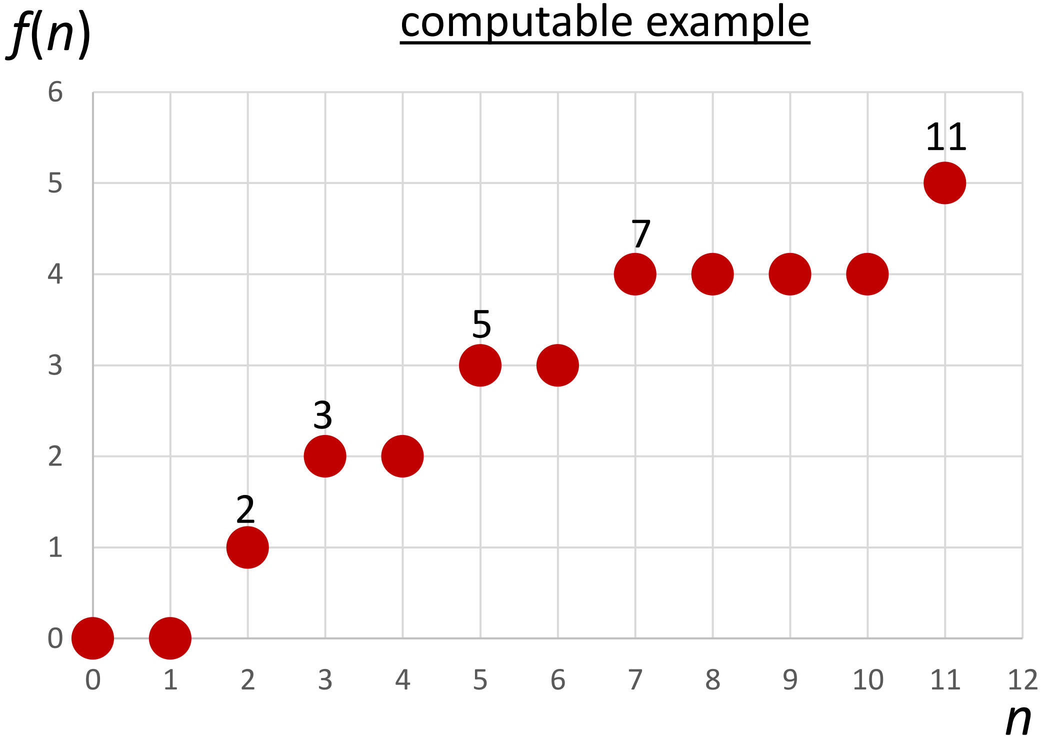
Theorem: A function is computable with probability of error < 1% by an integer-valued chemical reaction network if and only if it is computable by <u>any algorithm whatsoever</u>…

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

… "efficiently" (polynomial-time slowdown) …

… **if** we have an initial leader.

<u>computable example</u>



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*. <u>Computational Methods in Systems Biology – CMSB</u> 2017]

mass-action kinetics:

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

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

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

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

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

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 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]

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

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

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

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

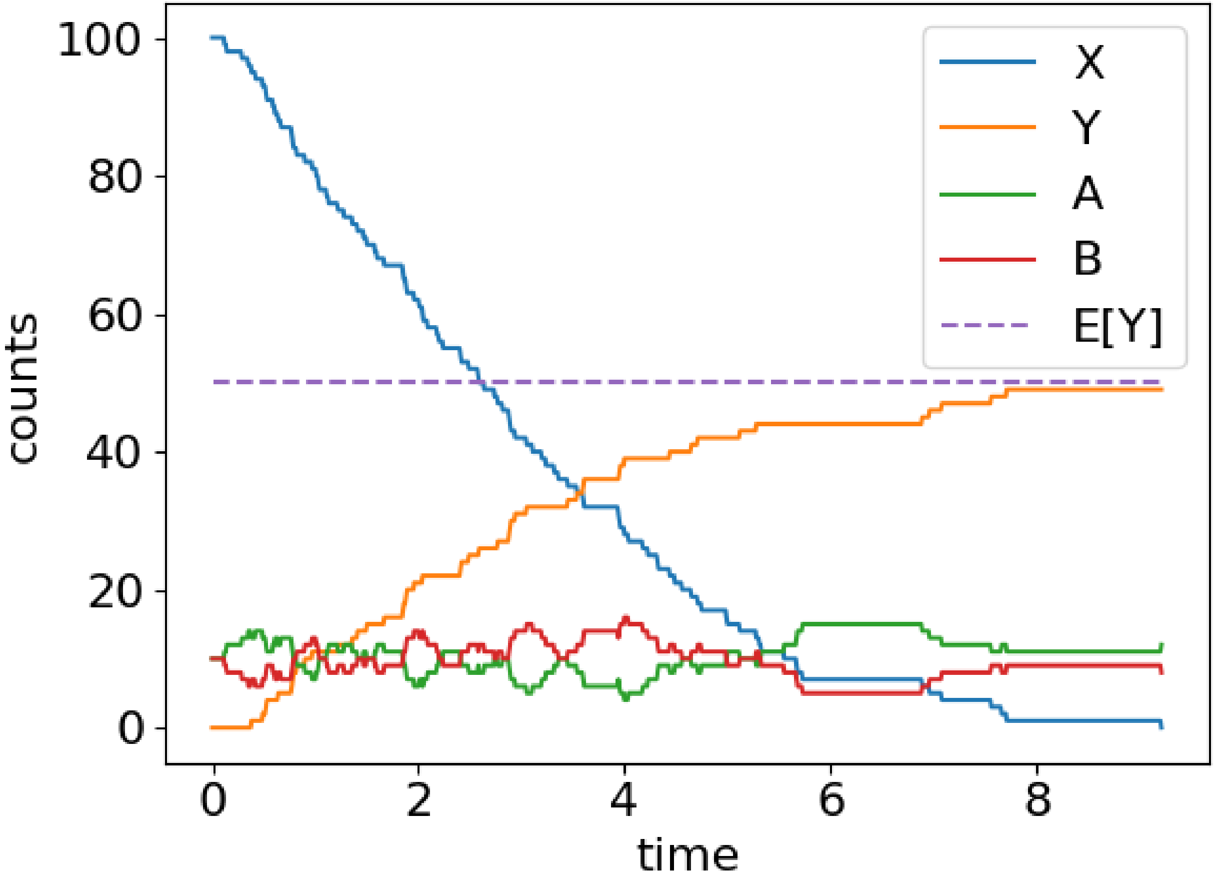
Fast approximate division by 2

n = 100 ε = 0.1

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

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

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

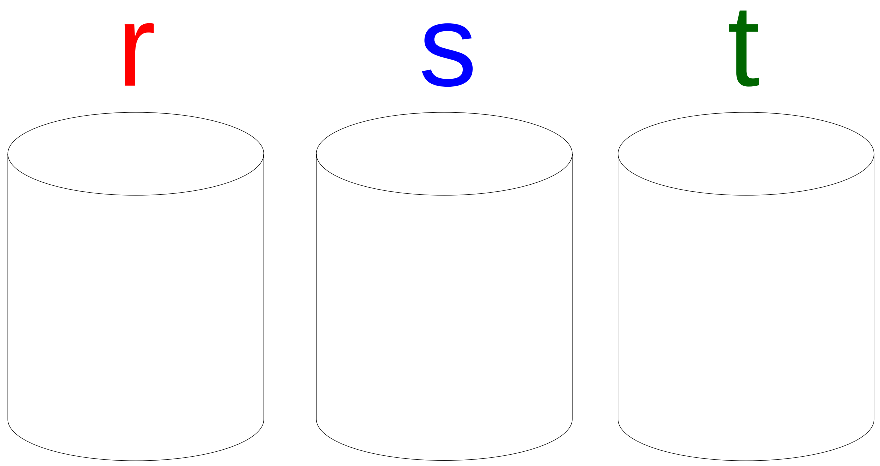


[Belleville, Doty, Soloveichik, *Hardness of computing and approximating predicates and functions with leaderless population protocols*, <ins>ICALP</ins> 2017]

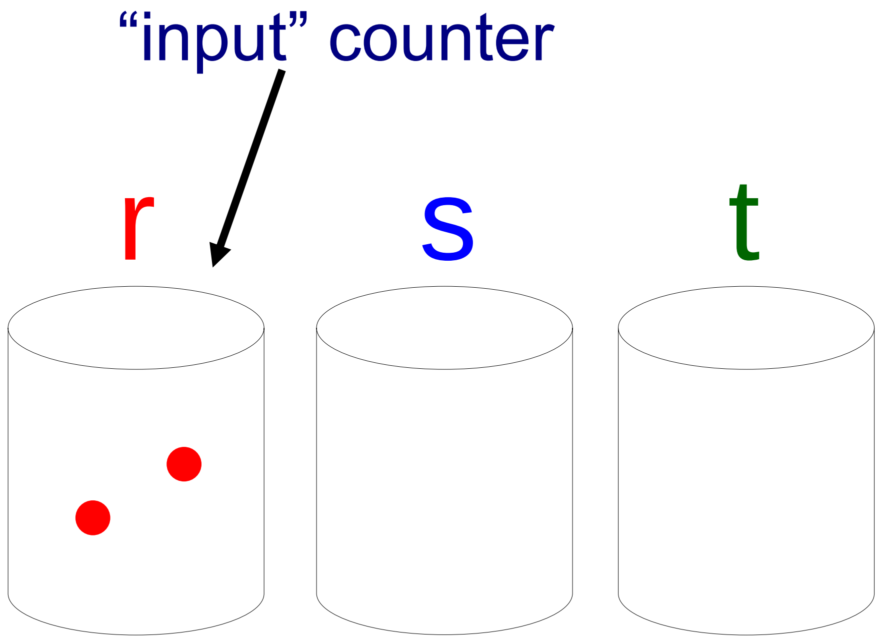
CRN computation with a small chance of error

Counter (register) machine

Counter (register) machine



Counter (register) machine



Counter (register) machine

1) **dec** r

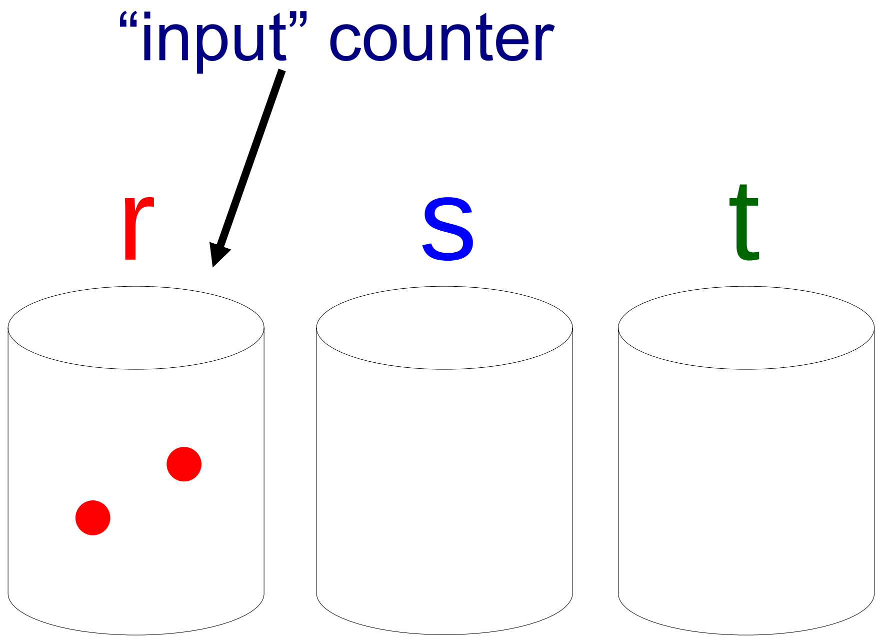
2) **inc** s

3) **inc** s

4) **inc** s

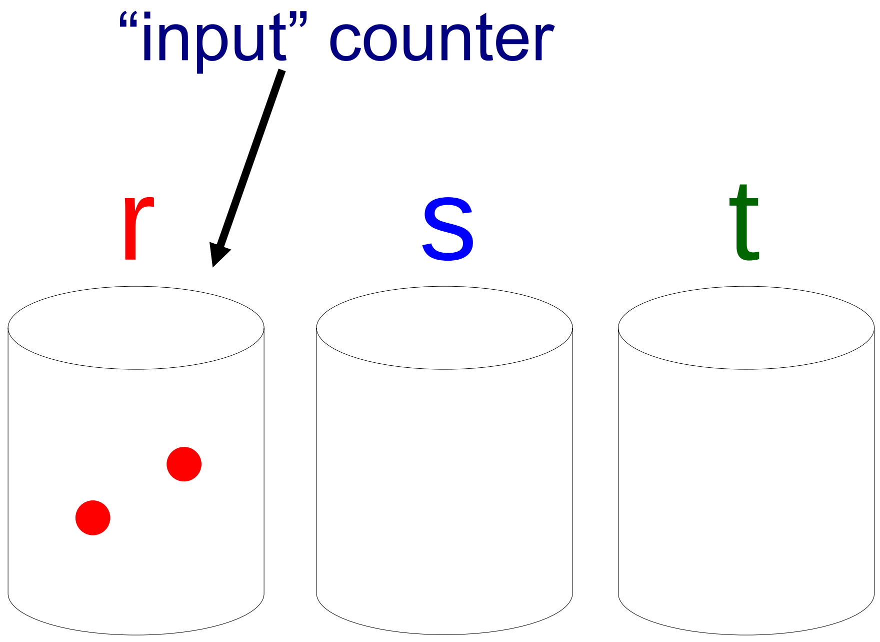
5) **dec** t

6) **inc** s



Counter (register) machine

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



Counter (register) machine

1) **dec** r

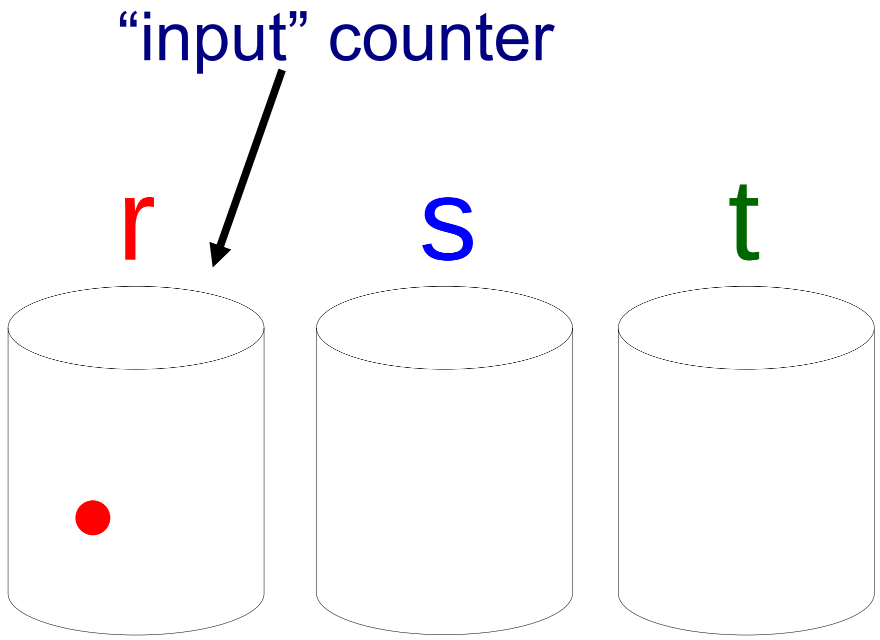
2) **inc** s

3) **inc** s

4) **inc** s

5) **dec** t

6) **inc** s



Counter (register) machine

1) **dec** r

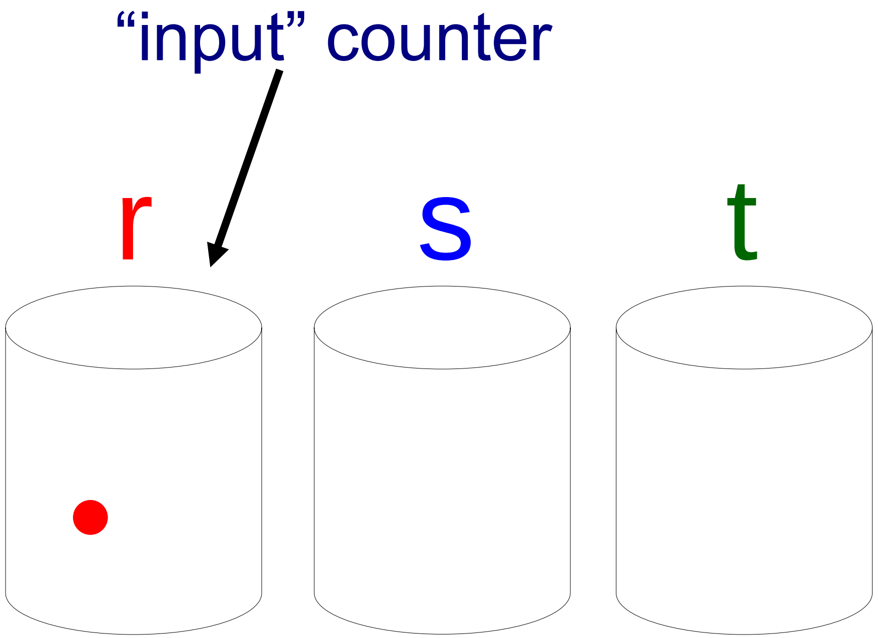
2) **inc** s

3) **inc** s

4) **inc** s

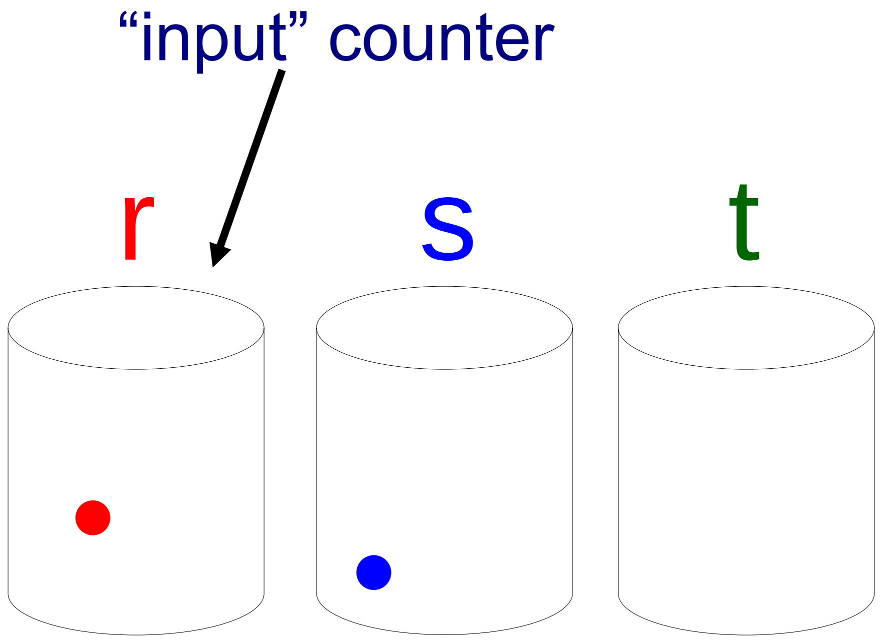
5) **dec** t

6) **inc** s



Counter (register) machine

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



Counter (register) machine

1) **dec** r

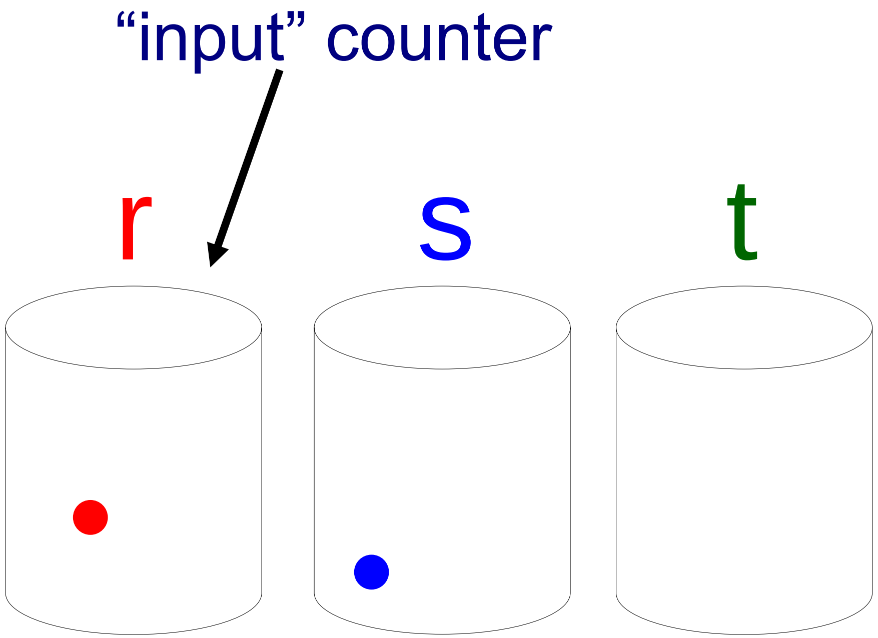
2) **inc** s

3) **inc** s

4) **inc** s

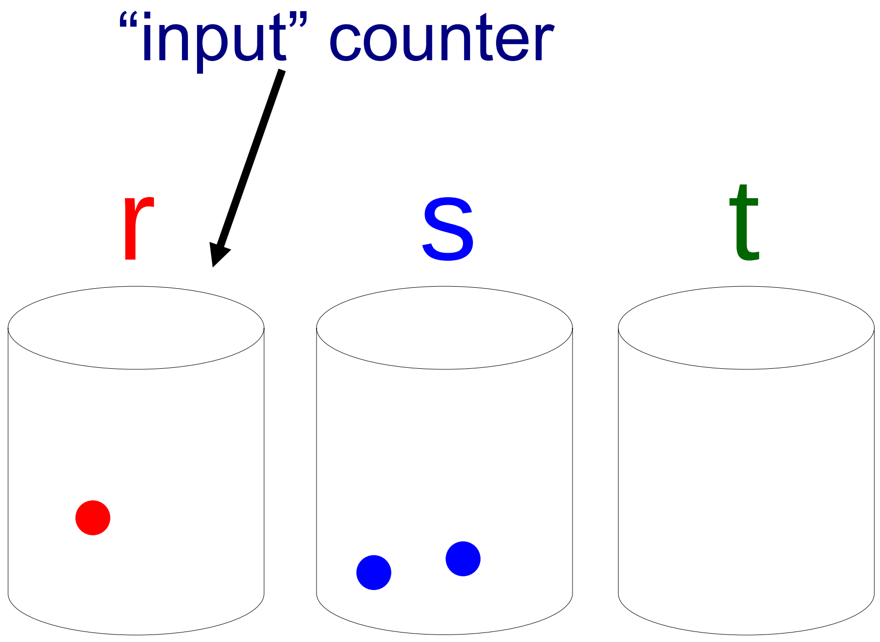
5) **dec** t

6) **inc** s



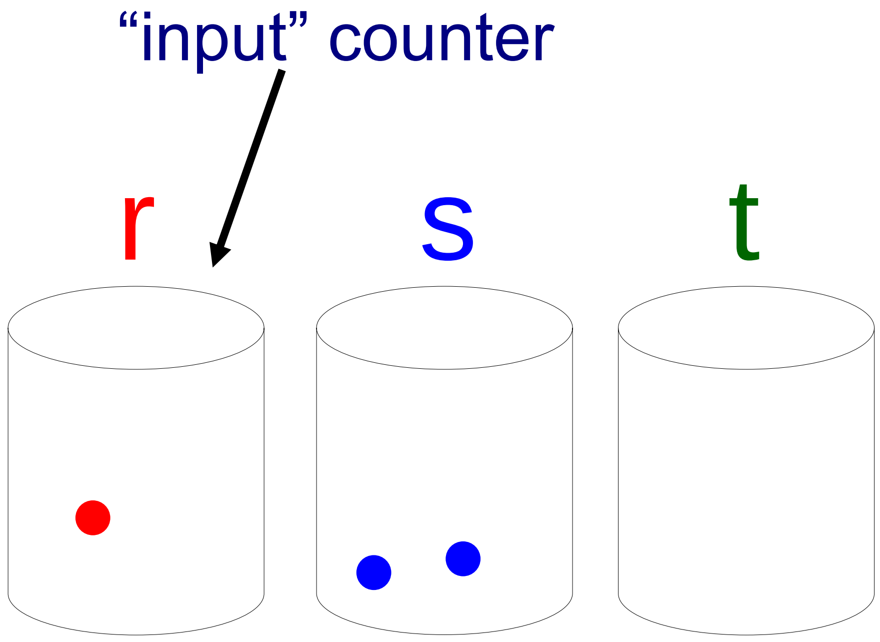
Counter (register) machine

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



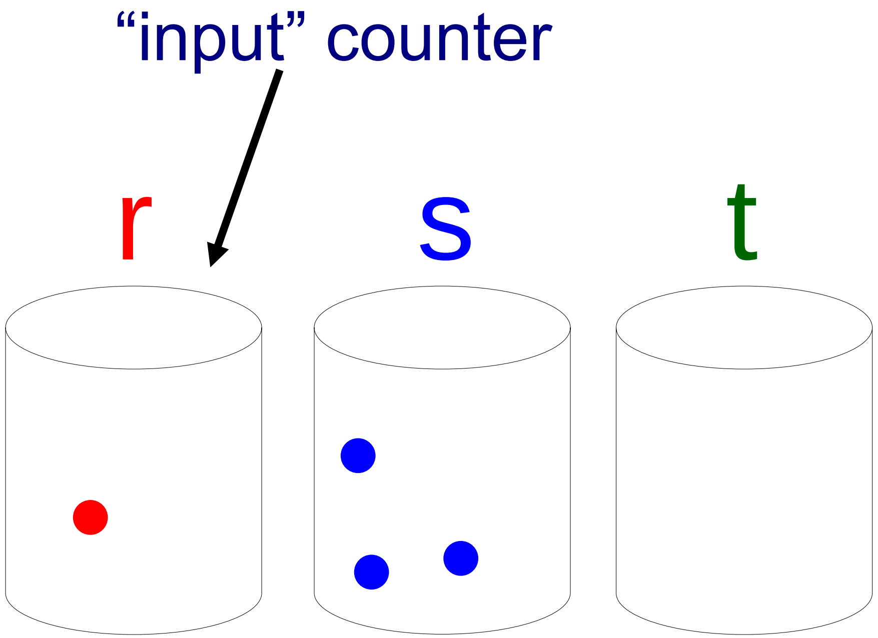
Counter (register) machine

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



Counter (register) machine

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



Counter (register) machine

1) **dec** r

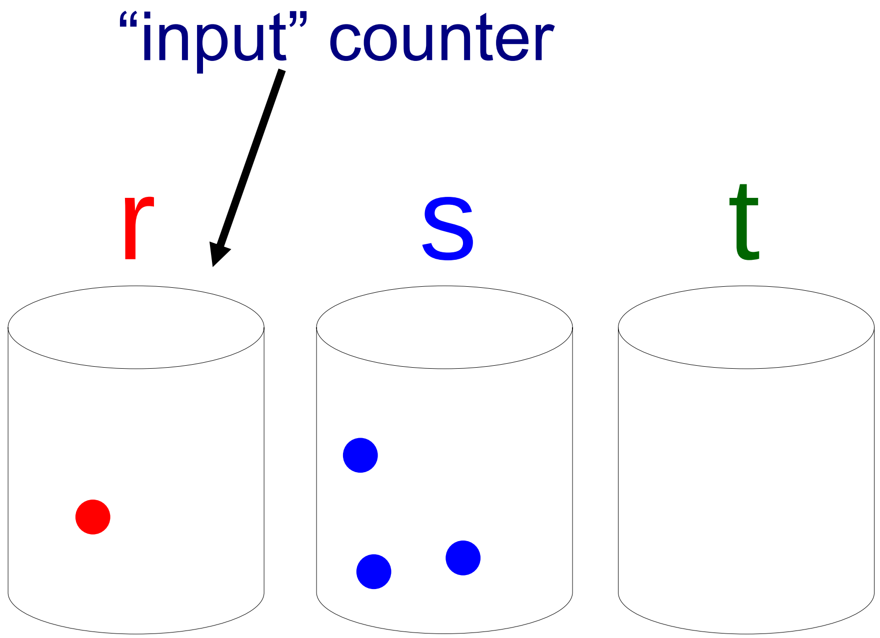
2) **inc** s

3) **inc** s

4) **inc** s

5) **dec** t

6) **inc** s



Counter (register) machine

1) **dec** r if empty goto 6

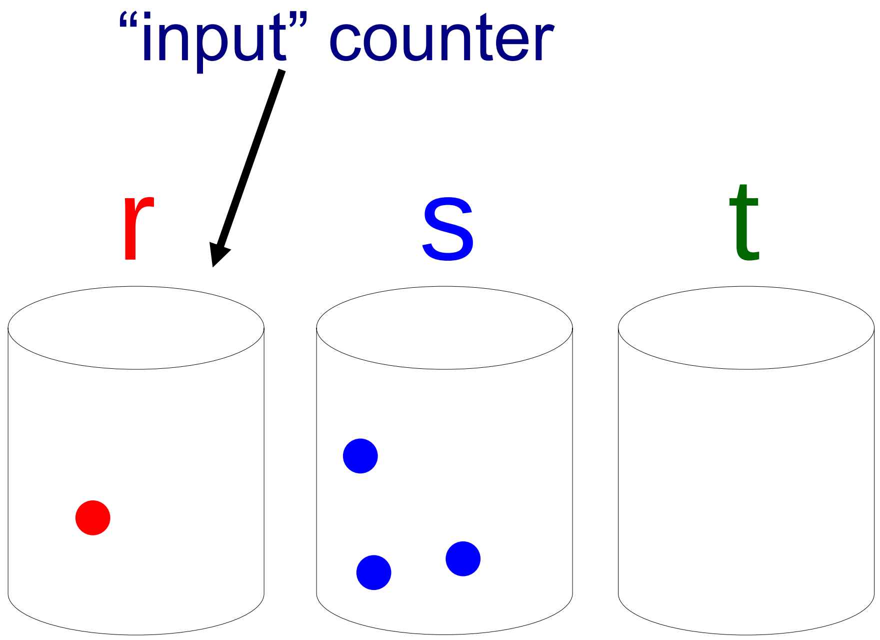
2) **inc** s

3) **inc** s

4) **inc** s

5) **dec** t if empty goto 1

6) **inc** s



Counter (register) machine

1) **dec** r if empty goto 6

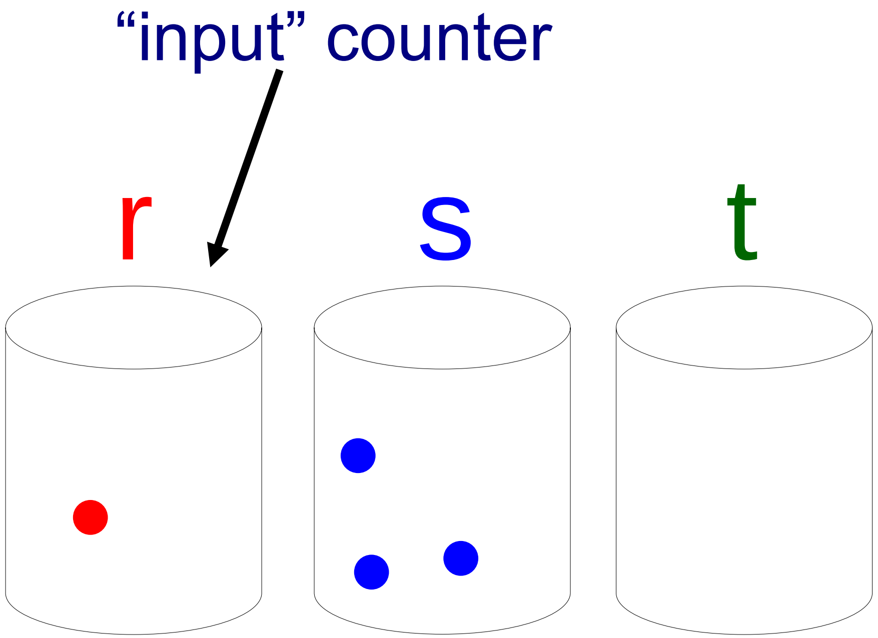
2) **inc** s

3) **inc** s

4) **inc** s

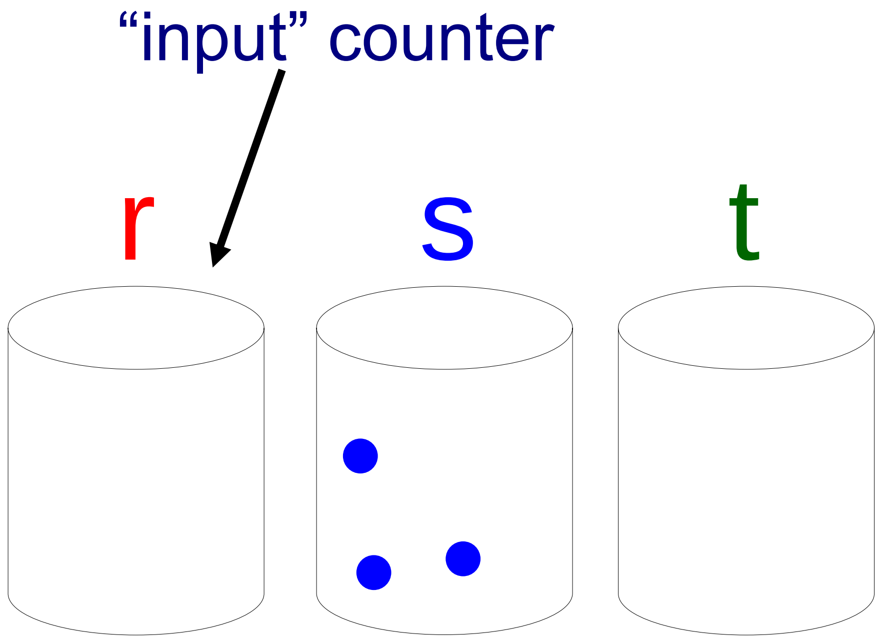
5) **dec** t if empty goto 1

6) **inc** s



Counter (register) machine

1) **dec** r if empty goto 6
2) **inc** s
3) **inc** s
4) **inc** s
5) **dec** t if empty goto 1
6) **inc** s



Counter (register) machine

1) **dec** r if empty goto 6

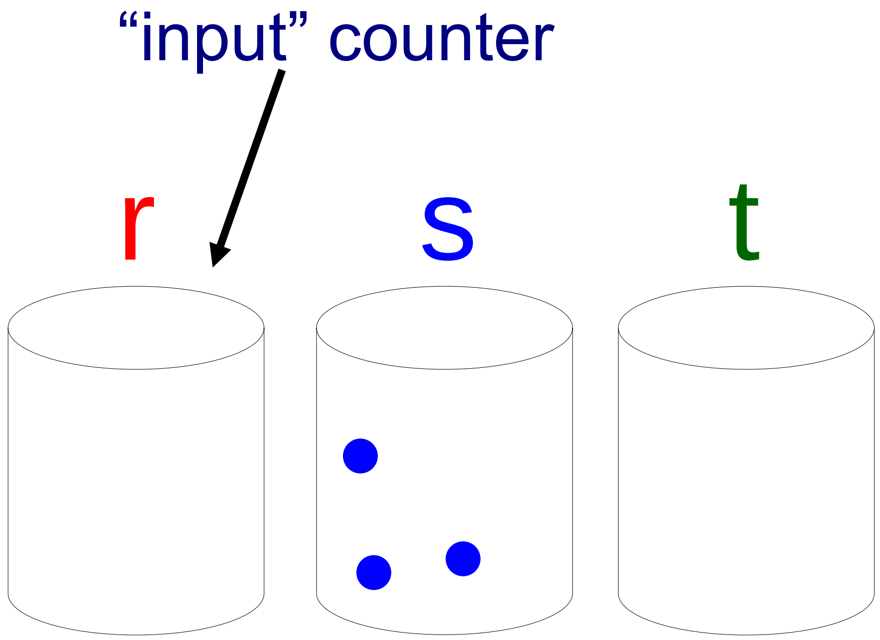
2) **inc** s

3) **inc** s

4) **inc** s

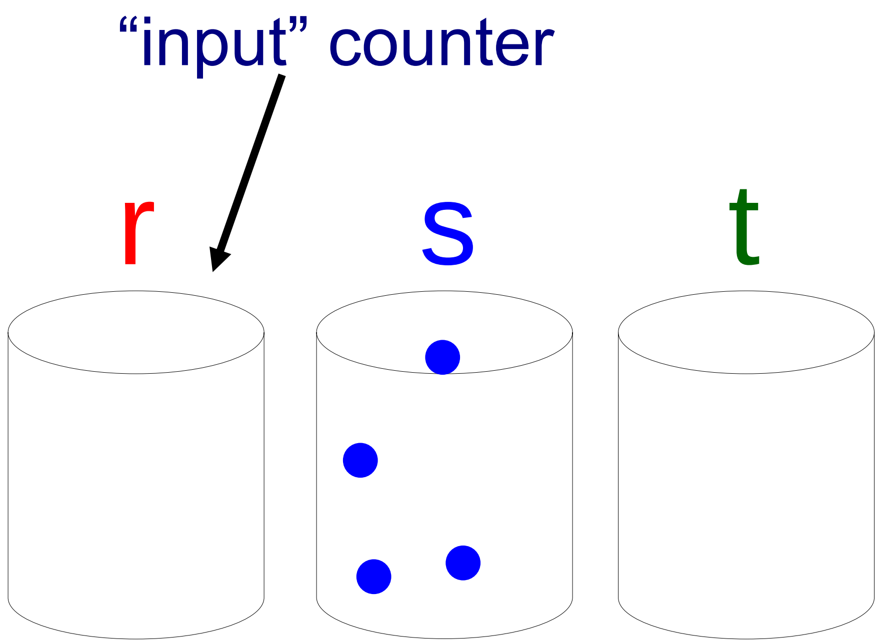
5) **dec** t if empty goto 1

6) **inc** s



Counter (register) machine

1) **dec** r if empty goto 6
2) **inc** s
3) **inc** s
4) **inc** s
5) **dec** t if empty goto 1
6) **inc** s



Counter (register) machine

1) **dec** r   if empty goto 6

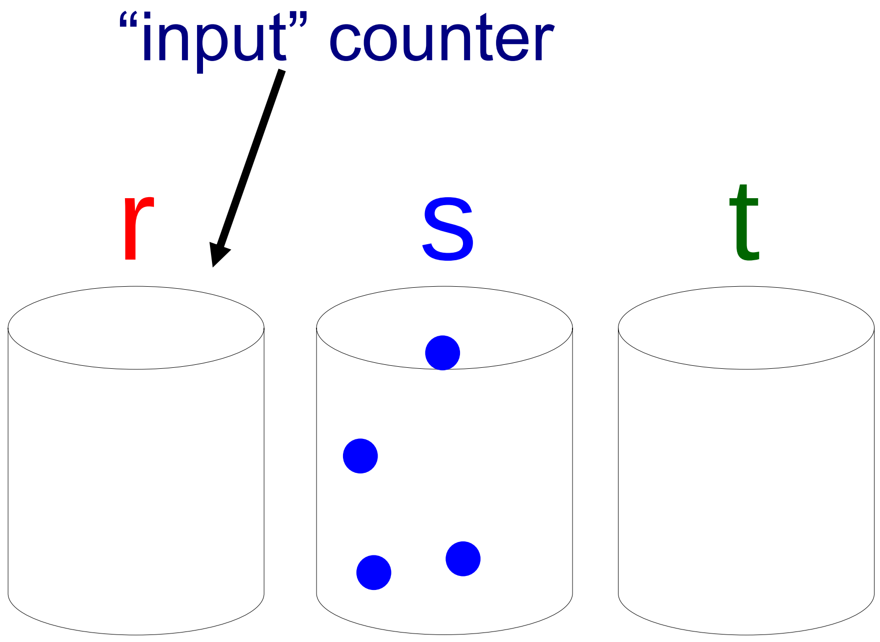
2) **inc** s

3) **inc** s

4) **inc** s

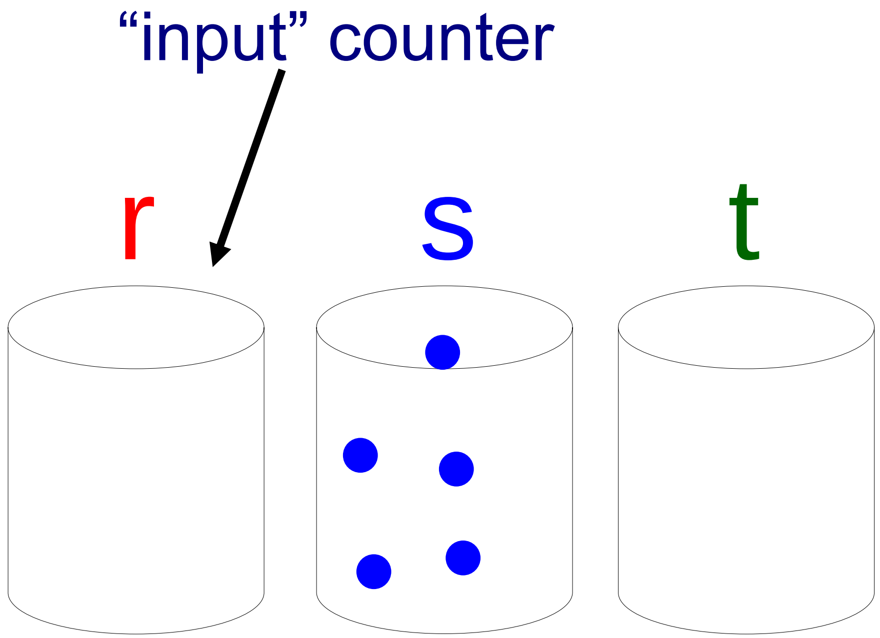
5) **dec** t   if empty goto 1

6) **inc** s



Counter (register) machine

1) **dec** r if empty goto 6
2) **inc** s
3) **inc** s
4) **inc** s
5) **dec** t if empty goto 1
6) **inc** s



Counter (register) machine

1) **dec** r if empty goto 6

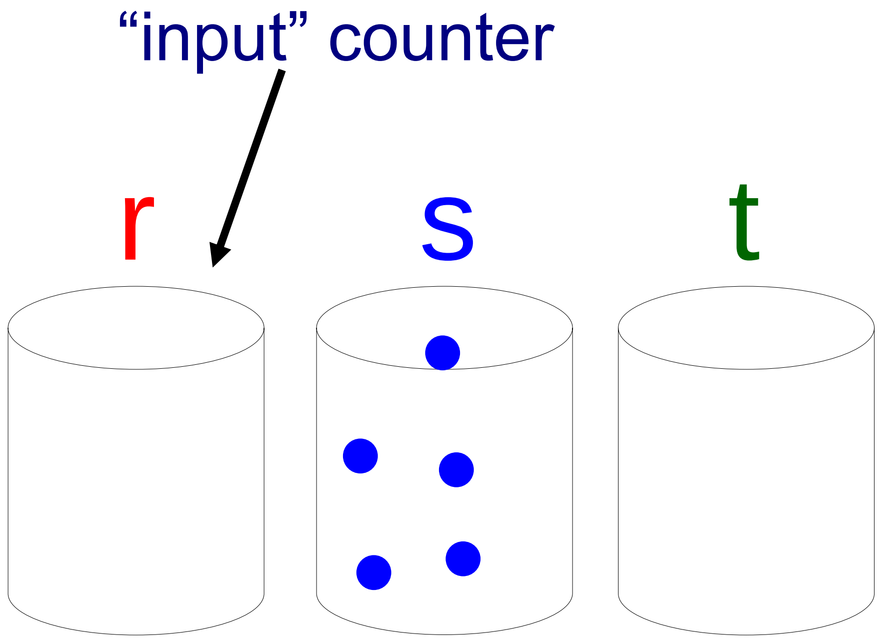
2) **inc** s

3) **inc** s

4) **inc** s

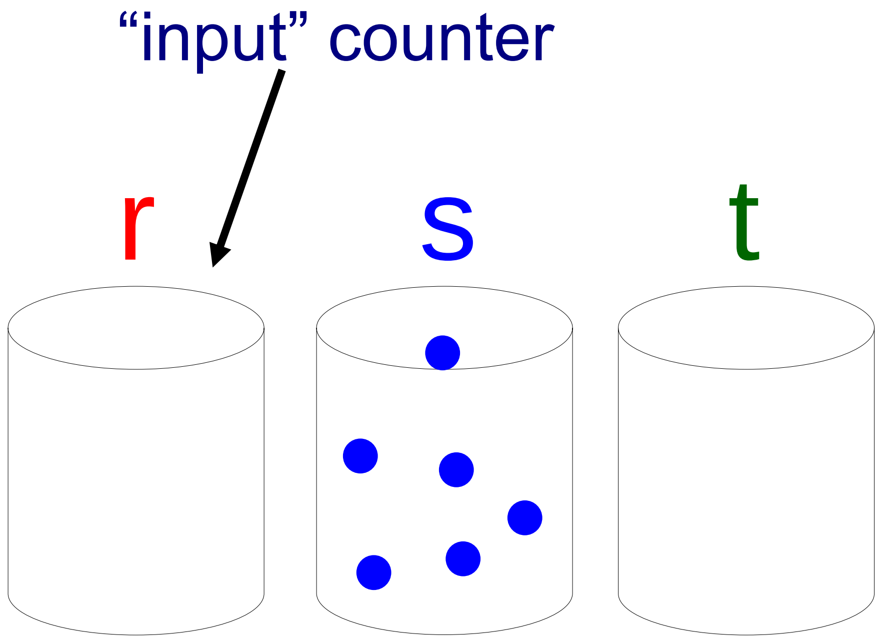
5) **dec** t if empty goto 1

6) **inc** s



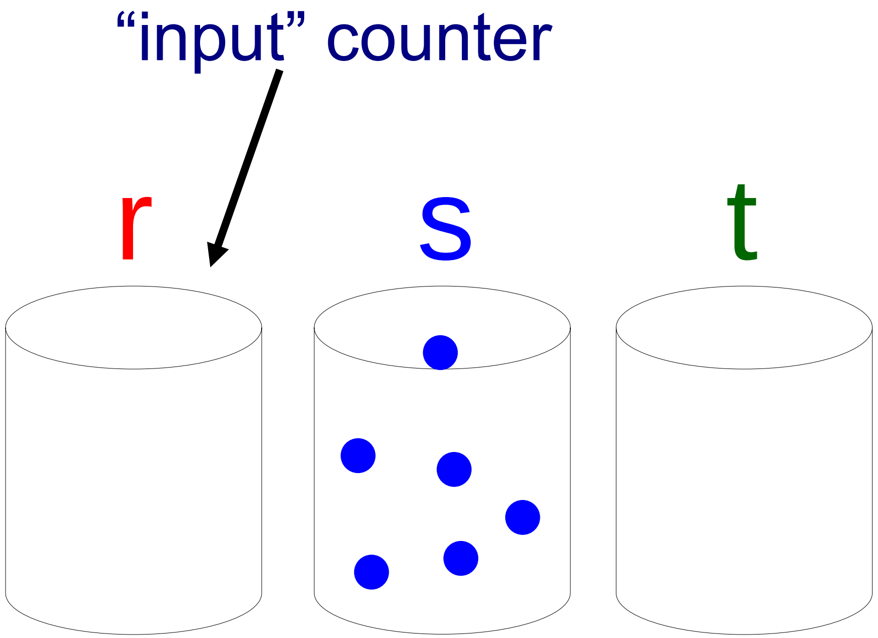
Counter (register) machine

1) **dec** r if empty goto 6
2) **inc** s
3) **inc** s
4) **inc** s
5) **dec** t if empty goto 1
6) **inc** s



Counter (register) machine

1) **dec** r if empty goto 6
2) **inc** s
3) **inc** s
4) **inc** s
5) **dec** t if empty goto 1
6) **inc** s



Counter (register) machine

1) **dec** r if empty goto 6

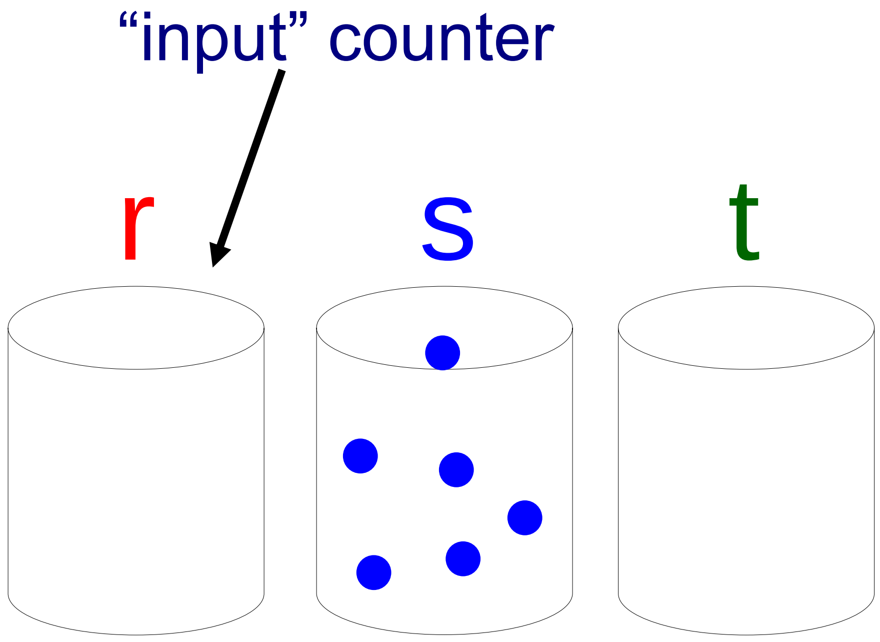
2) **inc** s

3) **inc** s

4) **inc** s

5) **dec** t if empty goto 1

6) **inc** s



Counter (register) machine

1) **dec** r   if empty goto 6

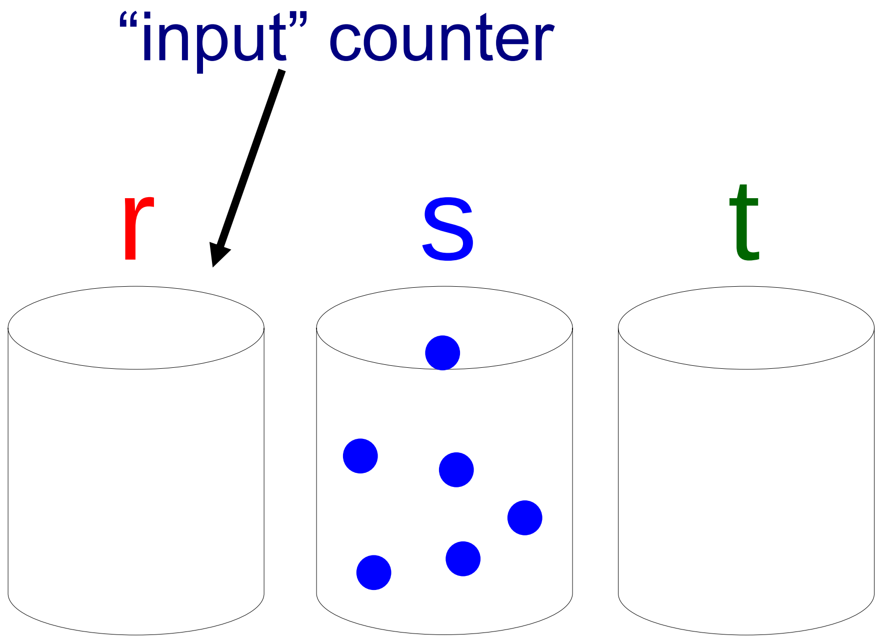
2) **inc** s

3) **inc** s

4) **inc** s

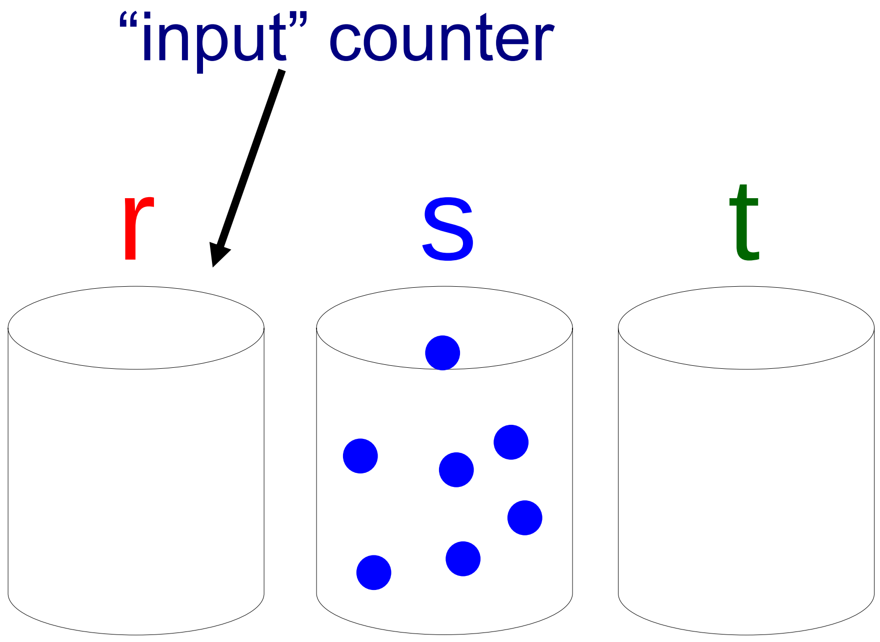
5) **dec** t   if empty goto 1

6) **inc** s



Counter (register) machine

1) **dec** r if empty goto 6
2) **inc** s
3) **inc** s
4) **inc** s
5) **dec** t if empty goto 1
6) **inc** s



Counter (register) machine

1) **dec** r if empty goto 6

2) **inc** s

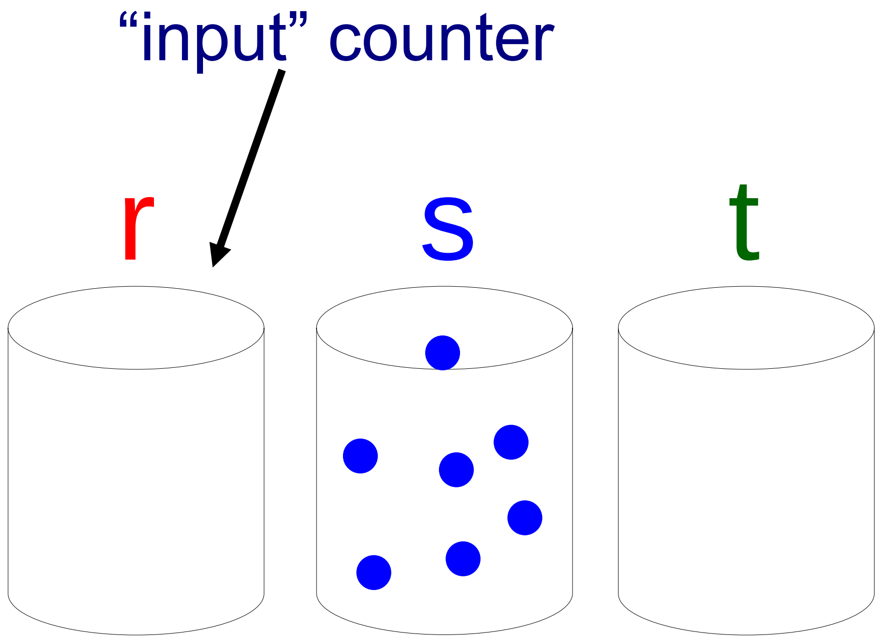
3) **inc** s

4) **inc** s

5) **dec** t if empty goto 1

6) **inc** s

HALT



Counter (register) machine

1) **dec** r if empty goto 6

2) **inc** s

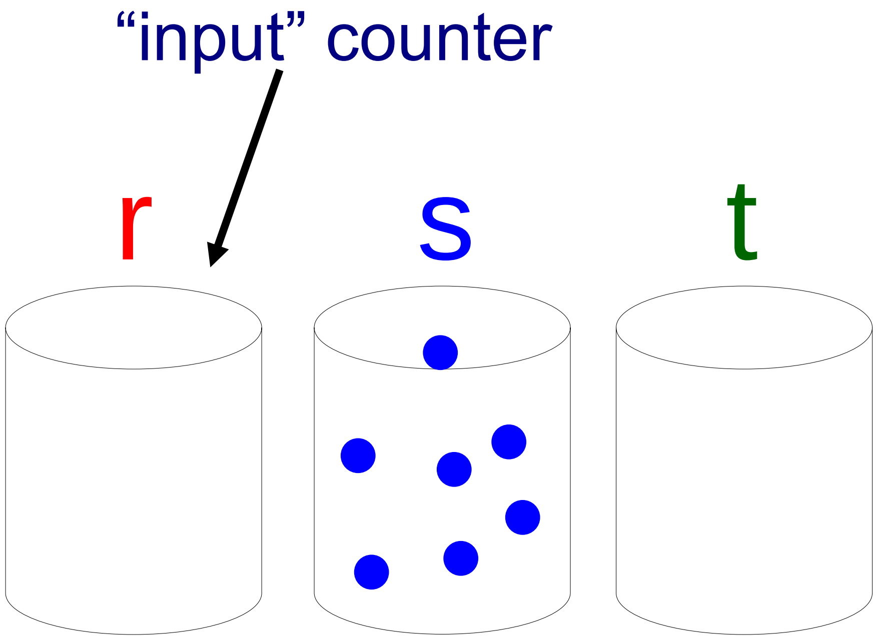
3) **inc** s

4) **inc** s

5) **dec** t if empty goto 1

6) **inc** s

HALT



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

Counter machines

- Finite state machine with a fixed number of counters c_1, c_2, …, c_k, each holding a nonnegative integer.

Counter machines

- Finite state machine with a fixed number of counters c_1, c_2, …, c_k, each holding a nonnegative integer.
- Start with inputs n_1, n_2, …, $n_l \in \mathbb{N}$ as values of c_1, c_2, …, c_l, and c_{l+1}, …, c_k start 0.

Counter machines

- Finite state machine with a fixed number of counters c_1, c_2, …, c_k, each holding a nonnegative integer.
- Start with inputs n_1, n_2, …, $n_l \in \mathbb{N}$ as values of c_1, c_2, …, c_l, and c_{l+1}, …, c_k start 0.
- Finite-state machine, where each state is one of:

Counter machines

- Finite state machine with a fixed number of counters $c_1, c_2, \ldots, c_k$, each holding a nonnegative integer.
- Start with inputs $n_1, n_2, \ldots, n_l \in \mathbb{N}$ as values of $c_1, c_2, \ldots, c_l$, and $c_{l+1}, \ldots, c_k$ start 0.
- Finite-state machine, where each state is one of:
  - **`inc`** `c`: increment counter *c*

Counter machines

- Finite state machine with a fixed number of counters c_1, c_2, …, c_k, each holding a nonnegative integer.
- Start with inputs $n_1, n_2, \ldots, n_l \in \mathbb{N}$ as values of c_1, c_2, …, c_l, and c_{l+1}, …, c_k start 0.
- Finite-state machine, where each state is one of:
  - **`inc`** `c`: increment counter c
  - **`dec`** `c`: decrement counter c; no effect if $c = 0$

Counter machines

- Finite state machine with a fixed number of counters $c_1, c_2, \ldots, c_k$, each holding a nonnegative integer.
- Start with inputs $n_1, n_2, \ldots, n_l \in \mathbb{N}$ as values of $c_1, c_2, \ldots, c_l$, and $c_{l+1}, \ldots, c_k$ start 0.
- Finite-state machine, where each state is one of:
  - **`inc`** `c`: increment counter c
  - **`dec`** `c`: decrement counter c; no effect if $c = 0$
  - **`if`** `c=0` **`goto`** `i`: if counter c is 0, then jump to state i

Counter machines

- Finite state machine with a fixed number of counters c_1, c_2, …, c_k, each holding a nonnegative integer.
- Start with inputs $n_1, n_2, \ldots, n_l \in \mathbb{N}$ as values of c_1, c_2, …, c_l, and c_{l+1}, …, c_k start 0.
- Finite-state machine, where each state is one of:
  - **`inc`** `c`: increment counter c
  - **`dec`** `c`: decrement counter c; no effect if $c = 0$
  - **`if`** `c=0` **`goto`** `i`: if counter c is 0, then jump to state i
  - **`goto`** `i` (can be shorthand for **`if`** `c=0` **`goto`** `i` for unused c)

Counter machines

- Finite state machine with a fixed number of counters c_1, c_2, …, c_k, each holding a nonnegative integer.
- Start with inputs $n_1, n_2, \ldots, n_l \in \mathbb{N}$ as values of c_1, c_2, …, c_l, and c_{l+1}, …, c_k start 0.
- Finite-state machine, where each state is one of:
  - **`inc`** `c`: increment counter c
  - **`dec`** `c`: decrement counter c; no effect if $c = 0$
  - **`if`** `c=0` **`goto`** `i`: if counter c is 0, then jump to state i
  - **`goto`** `i` (can be shorthand for **`if`** `c=0` **`goto`** `i` for unused c)
- may also have `accept/reject` semantics, or interpret the final value of some counter as the output

Example counter machines

input a

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

Example counter machines

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

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

Example counter machines

input a $f(a) = 2a$

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

```
1. while a>0:
2.   <instruction>
3.   <instruction>
…
i. …
```

is a shorthand for

```
1. if a=0 goto i
2. <instruction>
3. <instruction>
…
i-1. goto 1
i. …
```

Example counter machines

input a $f(a) = 2a$

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

```
1. while a>0:
2.   <instruction>
3.   <instruction>
…
i. …
```

is a shorthand for

```
1. if a=0 goto i
2. <instruction>
3. <instruction>
…
i-1. goto 1
i. …
```

input a

```
1. while a>0:
2.   dec a
3.   dec a
4.   inc b
```

Example counter machines

input a $f(a) = 2a$

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

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

```
1. while a>0:
2.   dec a
3.   dec a
4.   inc b
```

```
1. while a>0:
2.   <instruction>
3.   <instruction>
…
i. …
```

is a shorthand for

```
1. if a=0 goto i
2. <instruction>
3. <instruction>
…
i-1. goto 1
i. …
```

Example counter machines

input a $f(a) = 2a$

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

```
1. while a>0:
2.   <instruction>
3.   <instruction>
…
i. …
```

is a shorthand for

```
1. if a=0 goto i
2. <instruction>
3. <instruction>
…
i-1. goto 1
i. …
```

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

```
1. while a>0:
2.   dec a
3.   dec a
4.   inc b
```

input a

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

Example counter machines

input a $f(a) = 2a$

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

```
1. while a>0:
2.   <instruction>
3.   <instruction>
…
i. …
```

is a shorthand for

```
1. if a=0 goto i
2. <instruction>
3. <instruction>
…
i-1. goto 1
i. …
```

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

```
1. while a>0:
2.   dec a
3.   dec a
4.   inc b
```

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

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

Example counter machines

input a   *f*(*a*) = 2*a*

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

```
1. while a>0:
2.   <instruction>
3.   <instruction>
…
i. …
```

is a shorthand for

```
1. if a=0 goto i
2. <instruction>
3. <instruction>
…
i-1. goto 1
i. …
```

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

```
1. while a>0:
2.   dec a
3.   dec a
4.   inc b
```

inputs a,b

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

input a $\varphi(a)$ = "*a* is odd"

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

Example counter machines

input a $f(a) = 2a$

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

input a $f(a) = \lfloor a/2 \rfloor$

```
1. while a>0:
2.   dec a
3.   dec a
4.   inc b
```

input a $\varphi(a)$ = "a is odd"

```
1. if a=0 goto 7
2. dec a
3. if a=0 goto 6
4. dec a
5. goto 1
6. accept
7. reject
```

```
1. while a>0:
2.   <instruction>
3.   <instruction>
…
i. …
```

is a shorthand for

```
1. if a=0 goto i
2. <instruction>
3. <instruction>
…
i-1. goto 1
i. …
```

inputs a,b $f(a,b) = ab$

```
1. while a>0:
2.   dec a
3.   while b>0:
4.     dec b
5.     inc c
6.     inc d
7.   while c>0:
8.     dec c
9.     inc b
```

Example counter machines

input a $f(a) = 2a$

```
1. if a=0 goto 6
2.   dec a
3.   inc b
4.   inc b
5. goto 1
6. end
```

```
1. while a>0:
2.   <instruction>
3.   <instruction>
…
i. …
```

is a shorthand for

```
1. if a=0 goto i
2. <instruction>
3. <instruction>
…
i-1. goto 1
i. …
```

input a $f(a) = \lfloor a/2 \rfloor$

```
1. while a>0:
2.   dec a
3.   dec a
4.   inc b
```

inputs a,b $f(a,b) = ab$

```
1. while a>0:
2.   dec a
3.   while b>0:
4.     dec b
5.     inc c
6.     inc d
7.   while c>0:
8.     dec c
9.     inc b
```

input a $\varphi(a)$ = "*a* is odd"

```
1. if a=0 goto 7
2. dec a
3. if a=0 goto 6
4. dec a
5. goto 1
6. accept
7. reject
```

input a

```
1. inc b
2. while a>0:
3.   dec a
4.   while b>0:
5.     dec b
6.     inc c
7.     inc c
8.   while c>0:
9.     dec c
10.    inc b
```

Example counter machines

input a $f(a) = 2a$

```
1. if a=0 goto 6
2.    dec a
3.    inc b
4.    inc b
5. goto 1
6. end
```

```
1. while a>0:
2.    <instruction>
3.    <instruction>
…
i. …
```

is a shorthand for

```
1. if a=0 goto i
2. <instruction>
3. <instruction>
…
i-1. goto 1
i. …
```

input a $f(a) = \lfloor a/2 \rfloor$

```
1. while a>0:
2.    dec a
3.    dec a
4.    inc b
```

inputs a,b $f(a,b) = ab$

```
1. while a>0:
2.    dec a
3.    while b>0:
4.       dec b
5.       inc c
6.       inc d
7.    while c>0:
8.       dec c
9.       inc b
```

input a $\varphi(a)$ = "a is odd"

```
1. if a=0 goto 7
2. dec a
3. if a=0 goto 6
4. dec a
5. goto 1
6. accept
7. reject
```

input a $f(a) = 2^a$

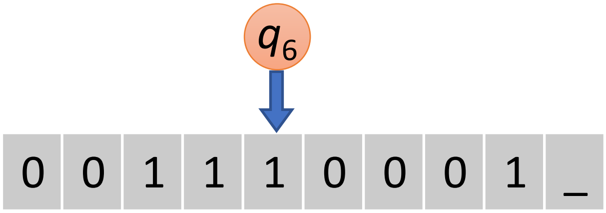
```
1. inc b
2. while a>0:
3.    dec a
4.    while b>0:
5.       dec b
6.       inc c
7.       inc c
8.    while c>0:
9.       dec c
10.      inc b
```

3-counter machines are Turing universal

3-counter machines are Turing universal

Assume Turing machine

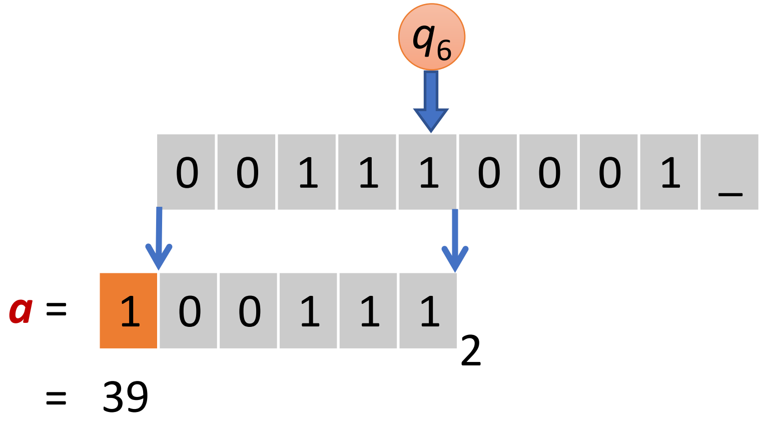
- has a single blank on rightmost cell
- if rightmost blank overwritten, it grows a *new* blank cell to right



3-counter machines are Turing universal

Assume Turing machine

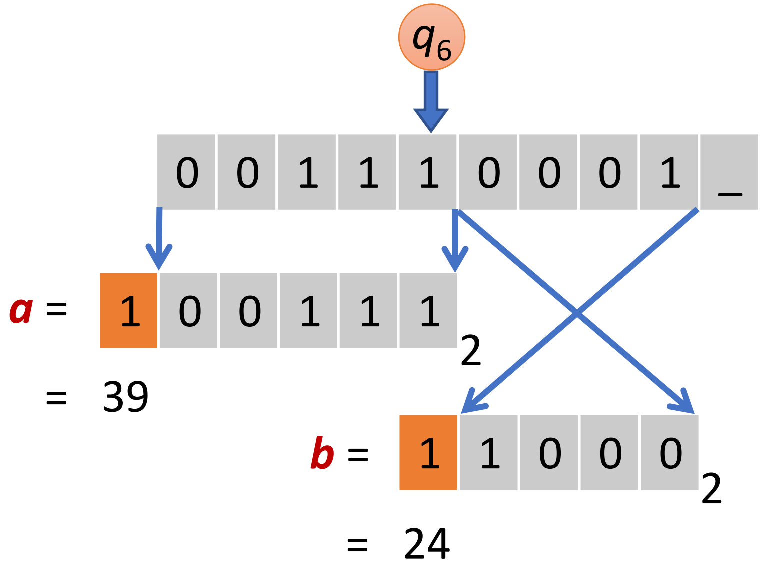
- has a single blank on rightmost cell
- if rightmost blank overwritten, it grows a *new* blank cell to right



Interpret tape on each side of tape head as binary number; append new leading 1 to make this mapping 1-1, in case the binary string has no leading 1 already, since 00111_2, 0111_2, and 111_2 are all considered the number 7.

3-counter machines are Turing universal

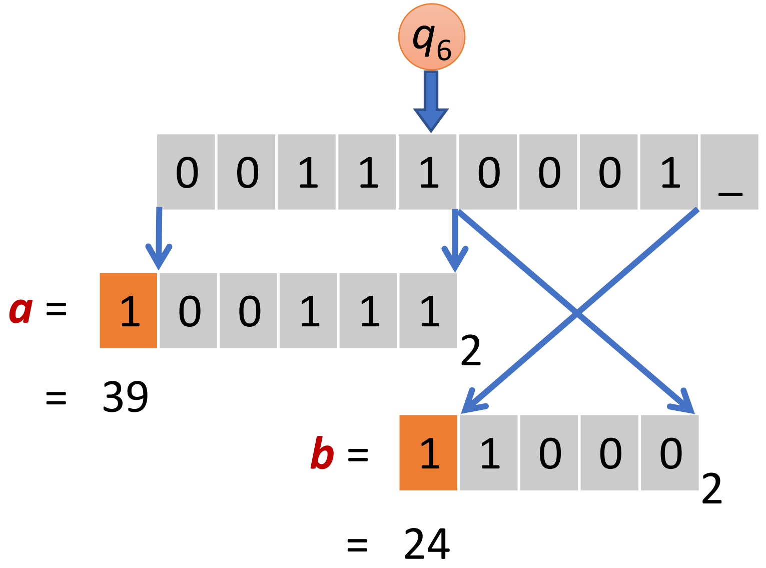
Assume Turing machine
- has a single blank on rightmost cell
- if rightmost blank overwritten, it grows a *new* blank cell to right



Interpret tape on each side of tape head as binary number; append new leading 1 to make this mapping 1-1, in case the binary string has no leading 1 already, since 00111_2, 0111_2, and 111_2 are all considered the number 7.

3-counter machines are Turing universal

Assume Turing machine
- has a single blank on rightmost cell
- if rightmost blank overwritten, it grows a *new* blank cell to right



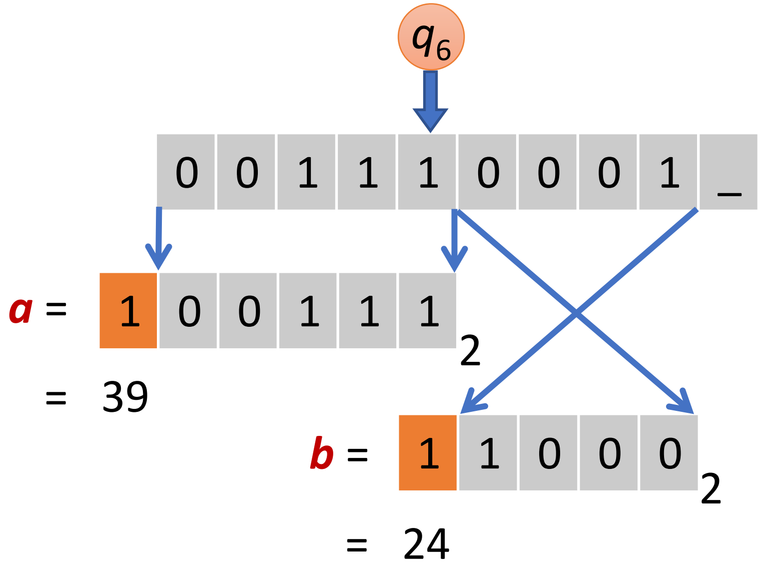
Need a third "work" counter ***c*** to help do the following operations on counters ***a*** and ***b***:

Turing machine operation	Counter machine implementation
read bit under tape head	
change bit under tape head	
move tape head right	
move tape head left	
test if tape head is on blank and if so, change it to 1	

Interpret tape on each side of tape head as binary number; append new leading 1 to make this mapping 1-1, in case the binary string has no leading 1 already, since 00111_2, 0111_2, and 111_2 are all considered the number 7.

3-counter machines are Turing universal

Assume Turing machine
- has a single blank on rightmost cell
- if rightmost blank overwritten, it grows a *new* blank cell to right



Need a third "work" counter ***c*** to help do the following operations on counters ***a*** and ***b***:

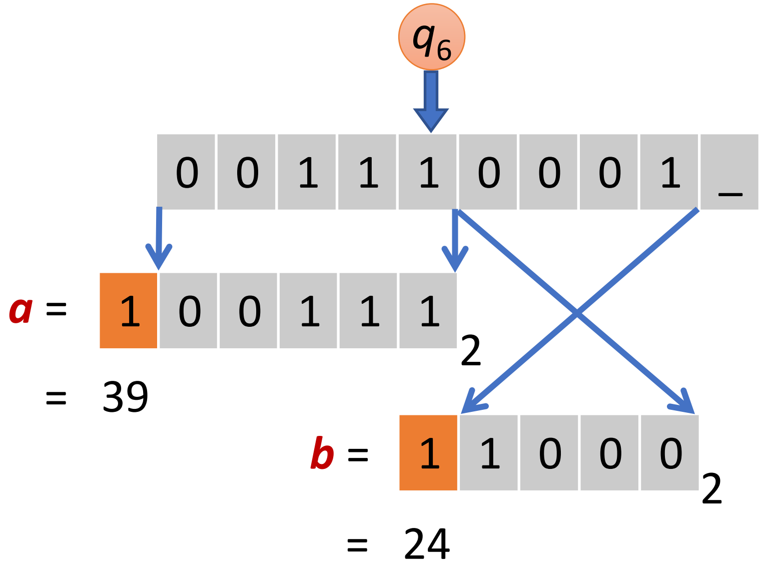
Turing machine operation	Counter machine implementation
read bit under tape head	is ***a*** odd?
change bit under tape head	
move tape head right	
move tape head left	
test if tape head is on blank and if so, change it to 1	

Interpret tape on each side of tape head as binary number; append new leading 1 to make this mapping 1-1, in case the binary string has no leading 1 already, since 00111_2, 0111_2, and 111_2 are all considered the number 7.

3-counter machines are Turing universal

Assume Turing machine

- has a single blank on rightmost cell
- if rightmost blank overwritten, it grows a *new* blank cell to right



Need a third "work" counter ***c*** to help do the following operations on counters ***a*** and ***b***:

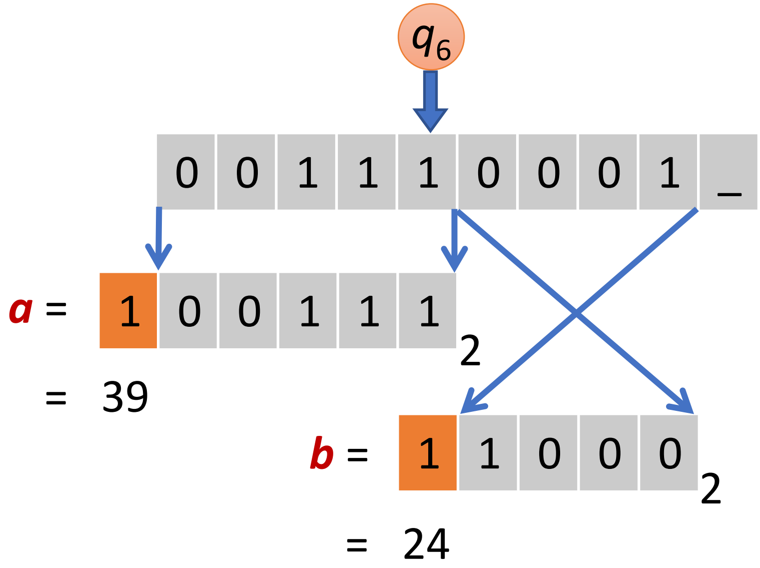
Turing machine operation	Counter machine implementation
read bit under tape head	is ***a*** odd?
change bit under tape head	inc/dec ***a***
move tape head right	
move tape head left	
test if tape head is on blank and if so, change it to 1	

Interpret tape on each side of tape head as binary number; append new leading 1 to make this mapping 1-1, in case the binary string has no leading 1 already, since 00111_2, 0111_2, and 111_2 are all considered the number 7.

3-counter machines are Turing universal

Assume Turing machine

- has a single blank on rightmost cell
- if rightmost blank overwritten, it grows a *new* blank cell to right



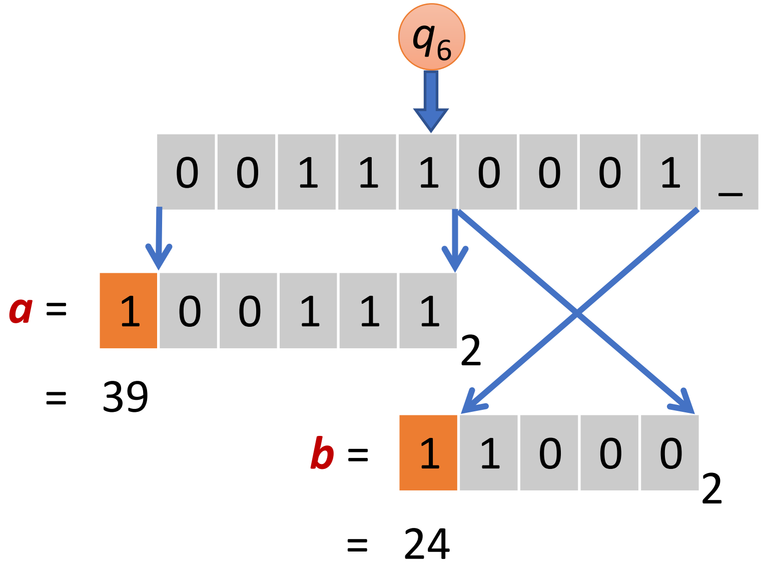
Need a third "work" counter $\boldsymbol{c}$ to help do the following operations on counters $\boldsymbol{a}$ and $\boldsymbol{b}$:

Turing machine operation	Counter machine implementation
read bit under tape head	is $\boldsymbol{a}$ odd?
change bit under tape head	inc/dec $\boldsymbol{a}$
move tape head right	set $\boldsymbol{a} = 2\boldsymbol{a}$ (+ 1) ; set $\boldsymbol{b} = \lfloor \boldsymbol{b}/2 \rfloor$
move tape head left	
test if tape head is on blank and if so, change it to 1	

Interpret tape on each side of tape head as binary number; append new leading 1 to make this mapping 1-1, in case the binary string has no leading 1 already, since 00111_2, 0111_2, and 111_2 are all considered the number 7.

3-counter machines are Turing universal

Assume Turing machine
- has a single blank on rightmost cell
- if rightmost blank overwritten, it grows a *new* blank cell to right



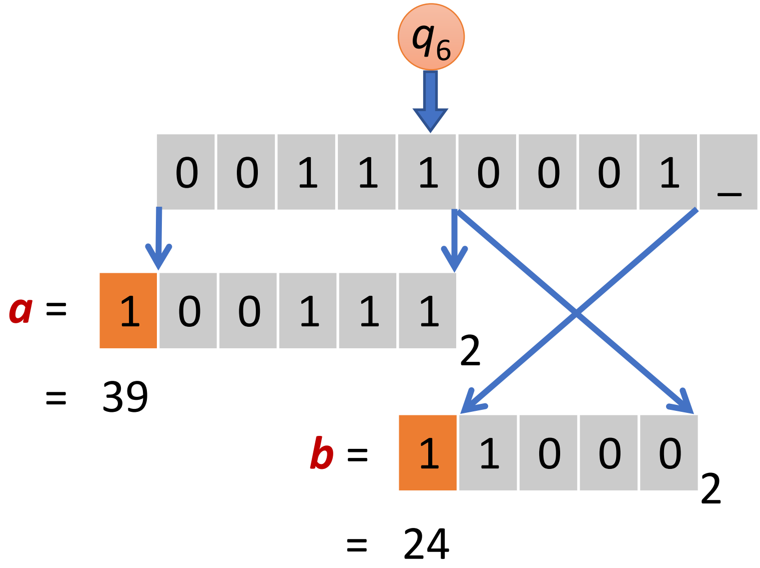
Need a third "work" counter ***c*** to help do the following operations on counters ***a*** and ***b***:

Turing machine operation	Counter machine implementation
read bit under tape head	is ***a*** odd?
change bit under tape head	inc/dec ***a***
move tape head right	set ***a*** = 2***a*** (+ 1) ; set ***b*** = ⌊***b***/2⌋
move tape head left	set ***b*** = 2***b*** (+ 1) ; set ***a*** = ⌊***a***/2⌋
test if tape head is on blank and if so, change it to 1	

Interpret tape on each side of tape head as binary number; append new leading 1 to make this mapping 1-1, in case the binary string has no leading 1 already, since 00111_2, 0111_2, and 111_2 are all considered the number 7.

3-counter machines are Turing universal

Assume Turing machine
- has a single blank on rightmost cell
- if rightmost blank overwritten, it grows a *new* blank cell to right



Need a third "work" counter ***c*** to help do the following operations on counters ***a*** and ***b***:

Turing machine operation	Counter machine implementation
read bit under tape head	is ***a*** odd?
change bit under tape head	inc/dec ***a***
move tape head right	set $\boldsymbol{a} = 2\boldsymbol{a}$ (+ 1) ; set $\boldsymbol{b} = \lfloor \boldsymbol{b}/2 \rfloor$
move tape head left	set $\boldsymbol{b} = 2\boldsymbol{b}$ (+ 1) ; set $\boldsymbol{a} = \lfloor \boldsymbol{a}/2 \rfloor$
test if tape head is on blank and if so, change it to 1	if $\boldsymbol{b}$=1 then set $\boldsymbol{a} = 2\boldsymbol{a} + 1$

Interpret tape on each side of tape head as binary number; append new leading 1 to make this mapping 1-1, in case the binary string has no leading 1 already, since 00111_2, 0111_2, and 111_2 are all considered the number 7.

1-counter machines are not Turing-universal… why?

2-counter machines are (sort of) Turing universal

[Minsky 1967, *Computation: Finite and Infinite Machines*]

2-counter machines are (sort of) Turing universal

[Minsky 1967, *Computation: Finite and Infinite Machines*]

- To represent counter values (a,b,c) in a single counter x, let $x = 2^a \cdot 3^b \cdot 5^c$ and $y = 0$.

2-counter machines are (sort of) Turing universal

[Minsky 1967, *Computation: Finite and Infinite Machines*]

- To represent counter values (a,b,c) in a single counter x, let $x = 2^a \cdot 3^b \cdot 5^c$ and $y = 0$.
  - To increment b, set $x = 3x$. (using y as a work counter)

2-counter machines are (sort of) Turing universal

[Minsky 1967, *Computation: Finite and Infinite Machines*]

- To represent counter values (a,b,c) in a single counter x, let $x = 2^a \cdot 3^b \cdot 5^c$ and $y = 0$.
  - To increment b, set $x = 3x$. (using y as a work counter)
  - To decrement a, set $x = \lfloor x/2 \rfloor$.

2-counter machines are (sort of) Turing universal

[Minsky 1967, *Computation: Finite and Infinite Machines*]

- To represent counter values (a,b,c) in a single counter x, let $x = 2^a \cdot 3^b \cdot 5^c$ and $y = 0$.
  - To increment b, set $x = 3x$. (using y as a work counter)
  - To decrement a, set $x = \lfloor x/2 \rfloor$.
  - To test if $c = 0$, test if $x \equiv 0 \bmod 5$.

2-counter machines are (sort of) Turing universal

[Minsky 1967, *Computation: Finite and Infinite Machines*]

- To represent counter values (a,b,c) in a single counter x, let $x = 2^a \cdot 3^b \cdot 5^c$ and $y = 0$.
  - To increment b, set $x = 3x$. (using y as a work counter)
  - To decrement a, set $x = \lfloor x/2 \rfloor$.
  - To test if $c = 0$, test if $x \equiv 0 \bmod 5$.
- To start with $a = n$ and $b = c = 0$, start with $x = 2^n \cdot 3^0 \cdot 5^0 = 2^n$.

2-counter machines are (sort of) Turing universal

[Minsky 1967, *Computation: Finite and Infinite Machines*]

- To represent counter values (a,b,c) in a single counter x, let $x = 2^a \cdot 3^b \cdot 5^c$ and $y = 0$.
  - To increment b, set $x = 3x$. (using y as a work counter)
  - To decrement a, set $x = \lfloor x/2 \rfloor$.
  - To test if $c = 0$, test if $x \equiv 0 \bmod 5$.
- To start with $a = n$ and $b = c = 0$, start with $x = 2^n \cdot 3^0 \cdot 5^0 = 2^n$.
- If $f{:}\mathbb{N} \to \mathbb{N}$ is any computable function, this machine can start with x=2^n and halt with x=$2^{f(n)}$.

2-counter machines are (sort of) Turing universal

[Minsky 1967, *Computation: Finite and Infinite Machines*]

- To represent counter values (a,b,c) in a single counter x, let $x = 2^a \cdot 3^b \cdot 5^c$ and $y = 0$.
  - To increment b, set $x = 3x$. (using y as a work counter)
  - To decrement a, set $x = \lfloor x/2 \rfloor$.
  - To test if $c = 0$, test if $x \equiv 0 \bmod 5$.
- To start with $a = n$ and $b = c = 0$, start with $x = 2^n \cdot 3^0 \cdot 5^0 = 2^n$.
- If $f: \mathbb{N} \to \mathbb{N}$ is any computable function, this machine can start with $x=2^n$ and halt with $x=2^{f(n)}$.
- Caveat about encoding: there is no 2-counter machine that starts with $x=n$ and halts with $x=2^n$.

> [Schroeppel 1972, *A Two Counter Machine Cannot Calculate* 2^N]
> "Theorem: Any counter machine can be simulated by a 2-counter machine, provided an obscure coding is accepted for the input and output."

2-counter machines are (sort of) Turing universal

[Minsky 1967, *Computation: Finite and Infinite Machines*]

- To represent counter values (a,b,c) in a single counter x, let $x = 2^a \cdot 3^b \cdot 5^c$ and $y = 0$.
  - To increment b, set $x = 3x$. (using y as a work counter)
  - To decrement a, set $x = \lfloor x/2 \rfloor$.
  - To test if $c = 0$, test if $x \equiv 0 \bmod 5$.
- To start with $a = n$ and $b = c = 0$, start with $x = 2^n \cdot 3^0 \cdot 5^0 = 2^n$.
- If $f : \mathbb{N} \to \mathbb{N}$ is any computable function, this machine can start with $x=2^n$ and halt with $x=2^{f(n)}$.
- Caveat about encoding: there is no 2-counter machine that starts with $x=n$ and halts with $x=2^n$.

  [Schroeppel 1972, *A Two Counter Machine Cannot Calculate 2^N*]
  "Theorem: Any counter machine can be simulated by a 2-counter machine, provided an obscure coding is accepted for the input and output."

  - 2-counter machines can do universal computation on *encoded* inputs (n encoded as 2^n), but they *cannot compute the encoding/decoding* themselves.

2-counter machines are (sort of) Turing universal

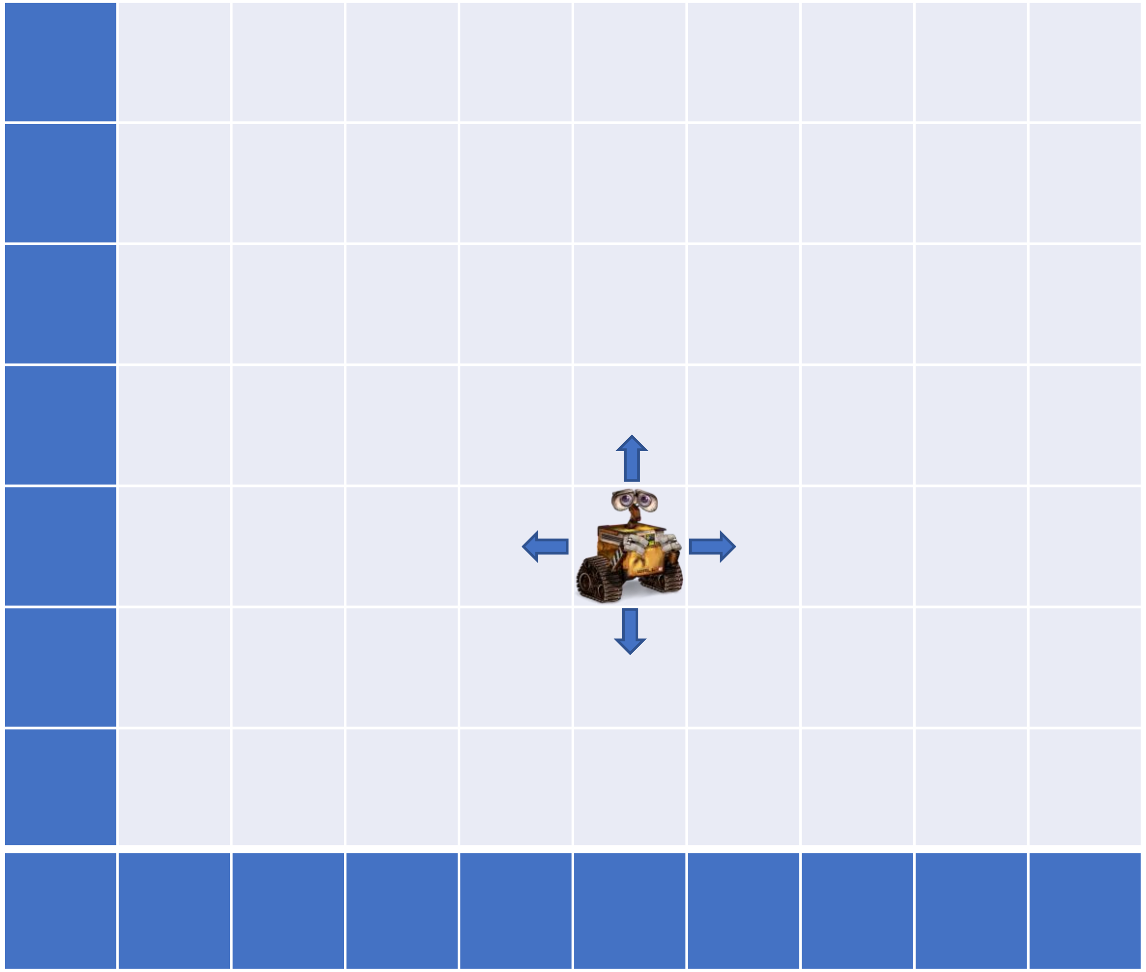
[Minsky 1967, *Computation: Finite and Infinite Machines*]

- To represent counter values (a,b,c) in a single counter x, let $x = 2^a \cdot 3^b \cdot 5^c$ and $y = 0$.
  - To increment b, set $x = 3x$. (using y as a work counter)
  - To decrement a, set $x = \lfloor x/2 \rfloor$.
  - To test if $c = 0$, test if $x \equiv 0 \bmod 5$.
- To start with $a = n$ and $b = c = 0$, start with $x = 2^n \cdot 3^0 \cdot 5^0 = 2^n$.
- If $f: \mathbb{N} \to \mathbb{N}$ is any computable function, this machine can start with $x=2^n$ and halt with $x=2^{f(n)}$.
- <u>Caveat about encoding</u>: there is no 2-counter machine that starts with $x=n$ and halts with $x=2^n$.

  [Schroeppel 1972, *A Two Counter Machine Cannot Calculate 2^N*]
  "<u>Theorem</u>: Any counter machine can be simulated by a 2-counter machine, provided an obscure coding is accepted for the input and output."

  - 2-counter machines can do universal computation on *encoded* inputs (n encoded as 2^n), but they *cannot compute the encoding/decoding* themselves.
  - However, the fact that 2-counter machines can simulate arbitrary 3-counter machines implies that the Halting Problem for 2-counter machines is undecidable.

2-counter machines: Finite automata robots on the plane



Finite automaton occupying a point $(x,y) \in \mathbb{N}^2$.

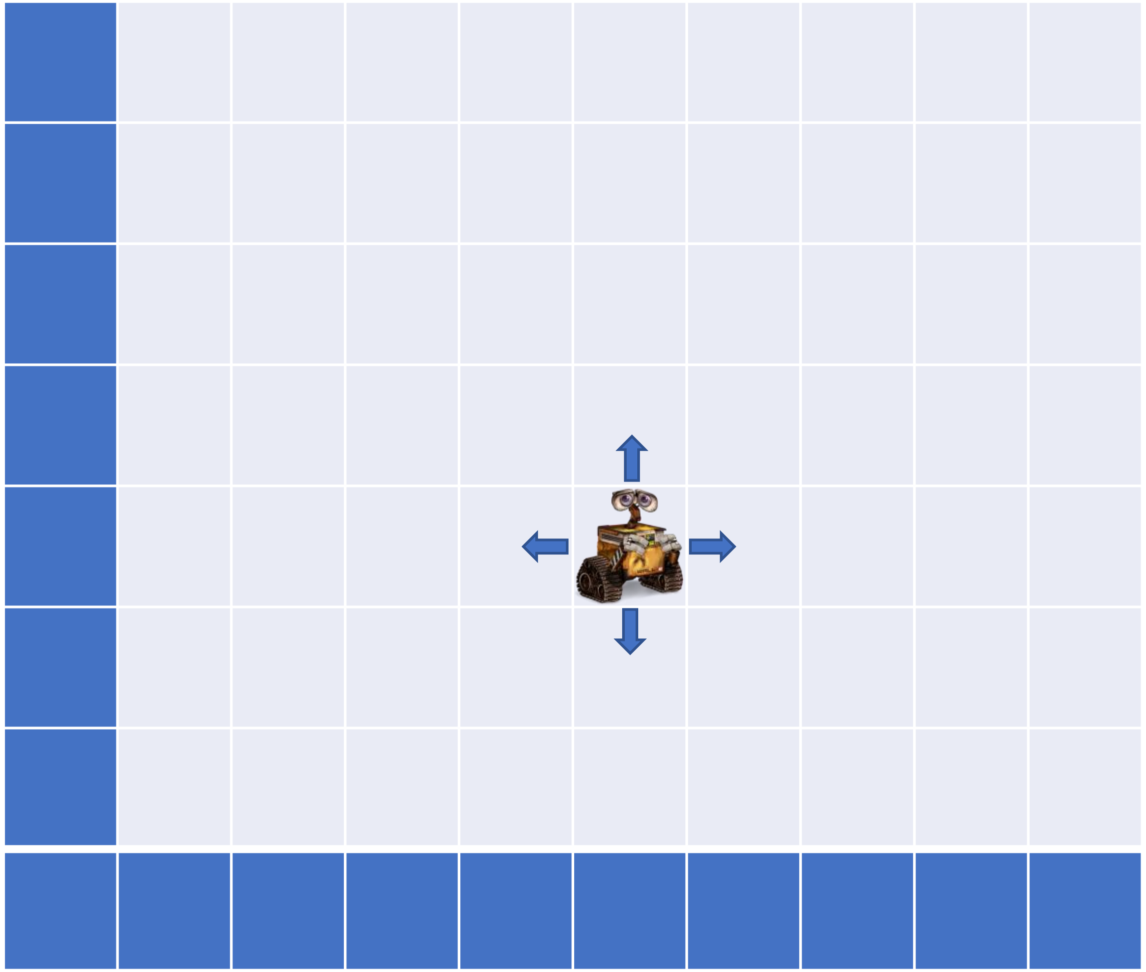
It cannot write anything, or see anything.

It can sense if it is touching the southern wall, or western wall (or both).

It can move north, south, east, or west based on its current state and 2 "wall bits", and of course change state:

$\delta: S \times \{\text{wall, no wall}\}^2 \to S \times \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$

2-counter machines: Finite automata robots on the plane



Finite automaton occupying a point $(x,y) \in \mathbb{N}^2$.

It cannot write anything, or see anything.

It can sense if it is touching the southern wall, or western wall (or both).

It can move north, south, east, or west based on its current state and 2 “wall bits”, and of course change state:

$\delta: S \times \{\text{wall, no wall}\}^2 \rightarrow S \times \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$

There is an automaton *A* so that this problem is undecidable: given $(x,y) \in \mathbb{N}^2$, if started at (x,y), will *A* ever visit the lower-left corner?

CRNs can simulate counter machines with probability < 1

[Soloveichik, Cook, Winfree, Bruck, Natural Computing 2008, Angluin, Aspnes, Eisenstat, DISC 2006, Distributed Computing 2008]

CRNs can simulate counter machines with probability < 1

Counter machine:
r = input *n*, start line 1

[Soloveichik, Cook, Winfree, Bruck, Natural Computing 2008, Angluin, Aspnes, Eisenstat, DISC 2006, Distributed Computing 2008]

CRNs can simulate counter machines with probability < 1

Counter machine:
r = input *n*, start line 1

CRN:
initial state $\{n\ R, 1\ L_1\}$

[Soloveichik, Cook, Winfree, Bruck, Natural Computing 2008, Angluin, Aspnes, Eisenstat, DISC 2006, Distributed Computing 2008]

CRNs can simulate counter machines with probability < 1

Counter machine:
r = input *n*, start line 1

1) **inc** r
2) **dec** r if zero goto 1
3) **inc** s
4) **dec** s if zero goto 2

CRN:
initial state $\{n\ R, 1\ L_1\}$

[Soloveichik, Cook, Winfree, Bruck, Natural Computing 2008, Angluin, Aspnes, Eisenstat, DISC 2006, Distributed Computing 2008]

CRNs can simulate counter machines with probability < 1

Counter machine:
r = input *n*, start line 1

1) **inc** r
2) **dec** r if zero goto 1
3) **inc** s
4) **dec** s if zero goto 2

CRN:
initial state $\{n\ R, 1\ L_1\}$

1) $L_1 \to L_2 + R$

[Soloveichik, Cook, Winfree, Bruck, Natural Computing 2008, Angluin, Aspnes, Eisenstat, DISC 2006, Distributed Computing 2008]

CRNs can simulate counter machines with probability < 1

Counter machine:
r = input n, start line 1

1) **inc** r
2) **dec** r if zero goto 1
3) **inc** s
4) **dec** s if zero goto 2

CRN:
initial state $\{n\ R, 1\ L_1\}$

1) $L_1 \to L_2 + R$
2) $L_2 + R \to L_3$

[Soloveichik, Cook, Winfree, Bruck, Natural Computing 2008, Angluin, Aspnes, Eisenstat, DISC 2006, Distributed Computing 2008]

CRNs can simulate counter machines with probability < 1

Counter machine:
r = input *n*, start line 1

1) **inc** r
2) **dec** r if zero goto 1
3) **inc** s
4) **dec** s if zero goto 2

CRN:
initial state $\{n\ R, 1\ L_1\}$

1) $L_1 \rightarrow L_2 + R$
2) $L_2 + R \rightarrow L_3$

[Soloveichik, Cook, Winfree, Bruck, Natural Computing 2008, Angluin, Aspnes, Eisenstat, DISC 2006, Distributed Computing 2008]

CRNs can simulate counter machines with probability < 1

Counter machine:
r = input *n*, start line 1

1) **inc** r
2) **dec** r if zero goto 1
3) **inc** s
4) **dec** s if zero goto 2

CRN:
initial state $\{n\, R, 1\, L_1\}$

1) $L_1 \to L_2 + R$
2) $L_2 + R \to L_3$; $L_2 \to L_1$

[Soloveichik, Cook, Winfree, Bruck, Natural Computing 2008, Angluin, Aspnes, Eisenstat, DISC 2006, Distributed Computing 2008]

CRNs can simulate counter machines with probability < 1

Counter machine:
r = input *n*, start line 1

1) **inc** r
2) **dec** r if zero goto 1
3) **inc** s
4) **dec** s if zero goto 2

CRN:
initial state $\{n\ R, 1\ L_1\}$

1) $L_1 \rightarrow L_2 + R$
2) $L_2 + R \rightarrow L_3$; $L_2 \rightarrow L_1$
3) $L_3 \rightarrow L_4 + S$
4) $L_4 + S \rightarrow L_5$; $L_4 \rightarrow L_2$

[Soloveichik, Cook, Winfree, Bruck, Natural Computing 2008, Angluin, Aspnes, Eisenstat, DISC 2006, Distributed Computing 2008]

CRNs can simulate counter machines with probability < 1

Counter machine:
r = input n, start line 1

1) **inc** r
2) **dec** r if zero goto 1
3) **inc** s
4) **dec** s if zero goto 2

CRN:
initial state $\{n\ R, 1\ L_1\}$

1) $L_1 \to L_2 + R$
2) $L_2 + R \to L_3$; $L_2 \to L_1$
3) $L_3 \to L_4 + S$
4) $L_4 + S \to L_5$; $L_4 \to L_2$

Error occurs when R is present, but reaction $L_2 \to L_1$ occurs instead of $L_2 + R \to L_3$.
Semantic effect on register machine: when $r > 0$, it may jump from line 2 to 1 without decrementing.
There's a positive probability of error; how to reduce it? Need to <u>slow down</u> $L_2 \to L_1$.

[Soloveichik, Cook, Winfree, Bruck, <u>Natural Computing</u> 2008, Angluin, Aspnes, Eisenstat, <u>DISC</u> 2006, <u>Distributed Computing</u> 2008]

CRNs can simulate counter machines with probability < 1

Counter machine:
r = input *n*, start line 1

1) **inc** r
2) **dec** r if zero goto 1
3) **inc** s
4) **dec** s if zero goto 2

CRN:
initial state $\{n\ R, 1\ L_1\}$

1) $L_1 \rightarrow L_2 + R$
2) $L_2 + R \rightarrow L_3$; $L_2 \rightarrow L_1$
3) $L_3 \rightarrow L_4 + S$
4) $L_4 + S \rightarrow L_5$; $L_4 \rightarrow L_2$

Need to be **very** slow!

Error occurs when R is present, but reaction $L_2 \rightarrow L_1$ occurs instead of $L_2 + R \rightarrow L_3$.
Semantic effect on register machine: when $r > 0$, it may jump from line 2 to 1 without decrementing.
There's a positive probability of error; how to reduce it? Need to slow down $L_2 \rightarrow L_1$.

[Soloveichik, Cook, Winfree, Bruck, Natural Computing 2008, Angluin, Aspnes, Eisenstat, DISC 2006, Distributed Computing 2008]

Problem with adjusting rate constant to slow down reactions for achieving Turing-universal computation

Could make rate constant *k* very small

- If correct reaction r_c: $L_2+R \longrightarrow L_3$ has rate constant 1, how small should k be to achieve Pr[r_i occurs instead of r_c] = Pr[error] = ε?
- rate of $r_c = \lambda_c = \#L_2 \cdot \#R/v = \#R/v \geq 1/v$
- rate of $r_i = \lambda_i = k \cdot \#L_2 = k$
- $\Pr[\text{error}] = \lambda_i \;/\; (\lambda_i + \lambda_c) \;\leq\; k \;/\; (k + 1/v)$
- For Pr[error] = ε, set $k = \varepsilon \;/\; (v - v\varepsilon) \approx \varepsilon/v$

Problems with simulation scheme so far

Problems with simulation scheme so far

1. Adjusting rate constants means designing new chemicals.

Problems with simulation scheme so far

1. Adjusting rate constants means designing new chemicals.
   - Easier to adjust counts of existing molecules than to design new ones.

Problems with simulation scheme so far

1. Adjusting rate constants means designing new chemicals.
    - Easier to adjust counts of existing molecules than to design new ones.
2. Pr[error in <u>any</u> time step] increases for longer computations.

Problems with simulation scheme so far

1. Adjusting rate constants means designing new chemicals.
   - Easier to adjust counts of existing molecules than to design new ones.
2. Pr[error in <u>any</u> time step] increases for longer computations.
   - By union bound we can only say Pr[error in any time step] ≤ $\varepsilon \cdot t$ (t = running time), so to achieve total error probability ≤ δ over all the computation requires setting $\varepsilon \le \delta/\mathrm{t}$, i.e., $k \le \delta/(t \cdot v)$.

Problems with simulation scheme so far

1. Adjusting rate constants means designing new chemicals.
   - Easier to adjust counts of existing molecules than to design new ones.
2. Pr[error in <u>any</u> time step] increases for longer computations.
   - By union bound we can only say Pr[error in any time step] ≤ $\varepsilon \cdot t$ (t = running time), so to achieve total error probability ≤ δ over all the computation requires setting $\varepsilon \le \delta/\mathrm{t}$, i.e., $k \le \delta/(t \cdot v)$.
   - <u>Universal</u> computation requires that we can simulate a program without knowing in advance how many steps it will take.

Problems with simulation scheme so far

1. Adjusting rate constants means designing new chemicals.
    - Easier to adjust counts of existing molecules than to design new ones.
2. Pr[error in <u>any</u> time step] increases for longer computations.
    - By union bound we can only say Pr[error in any time step] $\leq \varepsilon \cdot t$ (t = running time), so to achieve total error probability $\leq \delta$ over all the computation requires setting $\varepsilon \leq \delta/\mathrm{t}$, i.e., $k \leq \delta/(t \cdot v)$.
    - <u>Universal</u> computation requires that we can simulate a program without knowing in advance how many steps it will take.
3. Reducing error slows down the computation “significantly”.

Problems with simulation scheme so far

1. Adjusting rate constants means designing new chemicals.
   - Easier to adjust counts of existing molecules than to design new ones.
2. Pr[error in <u>any</u> time step] increases for longer computations.
   - By union bound we can only say Pr[error in any time step] $\leq \varepsilon \cdot t$ (t = running time), so to achieve total error probability $\leq \delta$ over all the computation requires setting $\varepsilon \leq \delta/\mathrm{t}$, i.e., $k \leq \delta/(t \cdot v)$.
   - <u>Universal</u> computation requires that we can simulate a program without knowing in advance how many steps it will take.
3. Reducing error slows down the computation "significantly".
   - halving rate constant k decreases Pr[error] by half, but doubles expected running time of all jump steps

Problems with simulation scheme so far

1. Adjusting rate constants means designing new chemicals.
   - Easier to adjust counts of existing molecules than to design new ones.
2. Pr[error in <u>any</u> time step] increases for longer computations.
   - By union bound we can only say Pr[error in any time step] $\leq \varepsilon \cdot t$ (t = running time), so to achieve total error probability $\leq \delta$ over all the computation requires setting $\varepsilon \leq \delta/t$, i.e., $k \leq \delta/(t \cdot v)$.
   - <u>Universal</u> computation requires that we can simulate a program without knowing in advance how many steps it will take.
3. Reducing error slows down the computation "significantly".
   - halving rate constant k decreases Pr[error] by half, but doubles expected running time of all jump steps
4. Register machines are exponentially slower than Turing machines.

Problems with simulation scheme so far

1. Adjusting rate constants means designing new chemicals.
   - Easier to adjust counts of existing molecules than to design new ones.
2. Pr[error in any time step] increases for longer computations.
   - By union bound we can only say Pr[error in any time step] $\leq \varepsilon \cdot t$ (t = running time), so to achieve total error probability $\leq \delta$ over all the computation requires setting $\varepsilon \leq \delta/t$, i.e., $k \leq \delta/(t \cdot v)$.
   - Universal computation requires that we can simulate a program without knowing in advance how many steps it will take.
3. Reducing error slows down the computation “significantly”.
   - halving rate constant k decreases Pr[error] by half, but doubles expected running time of all jump steps
4. Register machines are exponentially slower than Turing machines.
5. To store b bits, we need $\Omega(2^b)$ molecules.

Problems with simulation scheme so far

1. Adjusting rate constants means designing new chemicals.
   - Easier to adjust counts of existing molecules than to design new ones.
2. Pr[error in <u>any</u> time step] increases for longer computations.
   - By union bound we can only say Pr[error in any time step] $\leq \varepsilon \cdot t$ (t = running time), so to achieve total error probability $\leq \delta$ over all the computation requires setting $\varepsilon \leq \delta/t$, i.e., $k \leq \delta/(t \cdot v)$.
   - <u>Universal</u> computation requires that we can simulate a program without knowing in advance how many steps it will take.
3. Reducing error slows down the computation "significantly".
   - halving rate constant k decreases Pr[error] by half, but doubles expected running time of all jump steps
4. Register machines are exponentially slower than Turing machines.
5. To store b bits, we need $\Omega(2^b)$ molecules.

- Problem 5 is fundamental in CRNs: they necessarily store a "unary" encoding of any integer.

Problems with simulation scheme so far

1. Adjusting rate constants means designing new chemicals.
   - Easier to adjust counts of existing molecules than to design new ones.
2. Pr[error in <u>any</u> time step] increases for longer computations.
   - By union bound we can only say Pr[error in any time step] $\leq \varepsilon \cdot t$ (t = running time), so to achieve total error probability $\leq \delta$ over all the computation requires setting $\varepsilon \leq \delta/\mathrm{t}$, i.e., $k \leq \delta/(t \cdot v)$.
   - <u>Universal</u> computation requires that we can simulate a program without knowing in advance how many steps it will take.
3. Reducing error slows down the computation "significantly".
   - halving rate constant k decreases Pr[error] by half, but doubles expected running time of all jump steps
4. Register machines are exponentially slower than Turing machines.
5. To store b bits, we need $\Omega(2^b)$ molecules.

- Problem 5 is fundamental in CRNs: they necessarily store a "unary" encoding of any integer.
- **Theorem(ish)**: There is a CRN solving problems 1–4.

Problems with simulation scheme so far

1. Adjusting rate constants means designing new chemicals.
   - Easier to adjust counts of existing molecules than to design new ones.
2. Pr[error in <u>any</u> time step] increases for longer computations.
   - By union bound we can only say Pr[error in any time step] $\leq \varepsilon \cdot t$ (t = running time), so to achieve total error probability $\leq \delta$ over all the computation requires setting $\varepsilon \leq \delta/t$, i.e., $k \leq \delta/(t \cdot v)$.
   - <u>Universal</u> computation requires that we can simulate a program without knowing in advance how many steps it will take.
3. Reducing error slows down the computation "significantly".
   - halving rate constant k decreases Pr[error] by half, but doubles expected running time of all jump steps
4. Register machines are exponentially slower than Turing machines.
5. To store b bits, we need $\Omega(2^b)$ molecules.

- Problem 5 is fundamental in CRNs: they necessarily store a "unary" encoding of any integer.
- **Theorem(ish)**: There is a CRN solving problems 1–4.
- We'll see how to solve problems 1–3 by simulating a register machine more efficiently.

Problems with simulation scheme so far

1. Adjusting rate constants means designing new chemicals.
   - Easier to adjust counts of existing molecules than to design new ones.
2. Pr[error in any time step] increases for longer computations.
   - By union bound we can only say Pr[error in any time step] $\leq \varepsilon \cdot t$ (t = running time), so to achieve total error probability $\leq \delta$ over all the computation requires setting $\varepsilon \leq \delta/t$, i.e., $k \leq \delta/(t \cdot v)$.
   - Universal computation requires that we can simulate a program without knowing in advance how many steps it will take.
3. Reducing error slows down the computation "significantly".
   - halving rate constant k decreases Pr[error] by half, but doubles expected running time of all jump steps
4. Register machines are exponentially slower than Turing machines.
5. To store *b* bits, we need $\Omega(2^b)$ molecules.

- Problem 5 is fundamental in CRNs: they necessarily store a "unary" encoding of any integer.
- **Theorem(ish)**: There is a CRN solving problems 1–4.
- We'll see how to solve problems 1–3 by simulating a register machine more efficiently.
- To handle Problem 4, see [Soloveichik, Cook, Winfree, Bruck, *Computation with Finite Stochastic Chemical Reaction Networks*, NaCo 2008]

How to slow down reaction $L_2 \longrightarrow L_1$?

How to slow down reaction $L_2 \longrightarrow L_1$?

Use a clock:

$1\ C_1,\ 1\ F,\ n\ B$

How to slow down reaction $L_2 \longrightarrow L_1$?

Use a clock:

$1\ C_1$, $1\ F$, $n\ B$

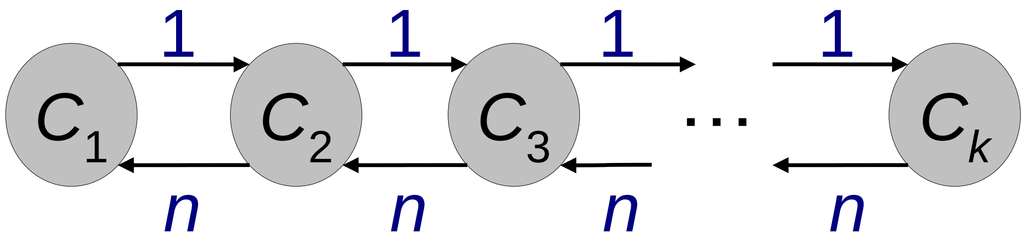
$$F + C_1 \rightarrow F + C_2 \qquad\qquad B + C_2 \rightarrow B + C_1$$
$$F + C_2 \rightarrow F + C_3 \qquad\qquad B + C_3 \rightarrow B + C_2$$
$$\vdots$$

How to slow down reaction $L_2 \longrightarrow L_1$?

Use a clock:

$1\ C_1,\ 1\ F,\ n\ B$

$$F + C_1 \to F + C_2 \qquad B + C_2 \to B + C_1$$
$$F + C_2 \to F + C_3 \qquad B + C_3 \to B + C_2$$
$$\vdots$$



reverse-biased random walk

How to slow down reaction $L_2 \longrightarrow L_1$?

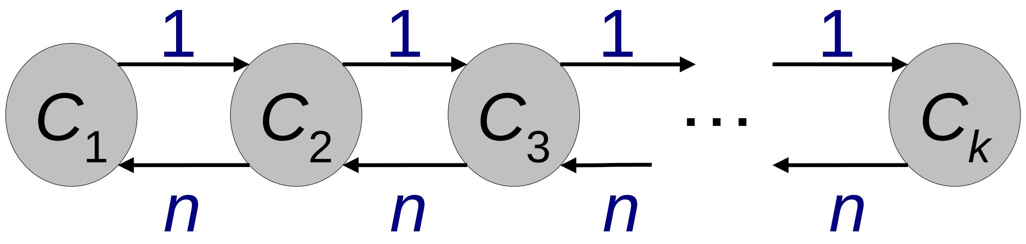
Use a clock:

$1\ C_1$, $1\ F$, $n\ B$

$$F + C_1 \rightarrow F + C_2$$
$$F + C_2 \rightarrow F + C_3$$

$$B + C_2 \rightarrow B + C_1$$
$$B + C_3 \rightarrow B + C_2$$

$$\vdots$$



reverse-biased random walk

C_k appears after expected time $\approx n^{k-1}$

How to slow down reaction $L_2 \longrightarrow L_1$?

Use a clock:

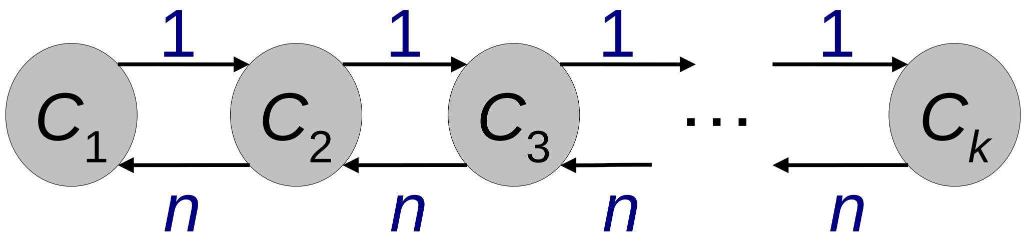
$1\ C_1$, $1\ F$, $n\ B$

$$C_k + L_2 \longrightarrow C_1 + L_1$$

$$F + C_1 \rightarrow F + C_2$$
$$F + C_2 \rightarrow F + C_3$$

$$B + C_2 \rightarrow B + C_1$$
$$B + C_3 \rightarrow B + C_2$$

$$\vdots$$



reverse-biased random walk

C_k appears after expected time $\approx n^{k\text{-}1}$

How to slow down reaction $L_2 \rightarrow L_1$?

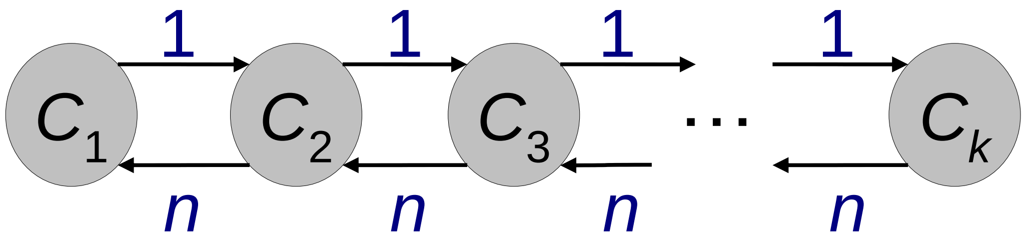
Use a clock:
$1\ C_1,\ 1\ F,\ n\ B$

$$\boxed{C_k + L_2 \rightarrow C_1 + L_1}$$

$$F + C_1 \rightarrow F + C_2$$
$$F + C_2 \rightarrow F + C_3$$

$$B + C_2 \rightarrow B + C_1$$
$$B + C_3 \rightarrow B + C_2$$

$$\vdots$$



reverse-biased random walk

C_k appears after expected time $\approx n^{k-1}$

E[time for $L_2 + R \rightarrow L_3$] $\leq n$

How to handle the three problems

Recall three problems we claimed we would solve:

1. Adjusting rate constants means designing new chemicals.
2. Pr[error in <u>any</u> time step] increases for longer computations.
3. Reducing error slows down the computation “significantly”.

How to handle the three problems

Recall three problems we claimed we would solve:

1. Adjusting rate constants means designing new chemicals.
2. Pr[error in <u>any</u> time step] increases for longer computations.
3. Reducing error slows down the computation “significantly”.

<u>Problem 1</u>: Now all rate constants = 1.

How to handle the three problems

Recall three problems we claimed we would solve:

1. Adjusting rate constants means designing new chemicals.
2. Pr[error in <u>any</u> time step] increases for longer computations.
3. Reducing error slows down the computation “significantly”.

<u>Problem 1</u>: Now all rate constants = 1.

<u>Problem 2</u>: How to make Pr[error in any time step] < ε, no matter how long the computation goes?

How to handle the three problems

Recall three problems we claimed we would solve:

1. Adjusting rate constants means designing new chemicals.
2. Pr[error in <u>any</u> time step] increases for longer computations.
3. Reducing error slows down the computation “significantly”.

<u>Problem 1</u>: Now all rate constants = 1.

<u>Problem 2</u>: How to make Pr[error in any time step] < ε, no matter how long the computation goes?

$$F + C_1 \to F + C_2 \qquad B + C_2 \to B + C_1$$
$$F + C_2 \to F + C_3 \qquad B + C_3 \to B + C_2$$

Two competing reactions, r_i incorrect, and r_c correct:

r_i: $C_k + L_2 \to C_1 + L_1$

r_c: $L_2 + R \to L_3$

How to handle the three problems

Recall three problems we claimed we would solve:

1. Adjusting rate constants means designing new chemicals.
2. Pr[error in <u>any</u> time step] increases for longer computations.
3. Reducing error slows down the computation "significantly".

<u>Problem 1</u>: Now all rate constants = 1.

<u>Problem 2</u>: How to make Pr[error in any time step] < ε, no matter how long the computation goes?

$$F + C_1 \to F + C_2 \qquad B + C_2 \to B + C_1$$
$$F + C_2 \to F + C_3 \qquad B + C_3 \to B + C_2$$

Two competing reactions, r_i incorrect, and r_c correct:

r_i: $C_k + L_2 \to C_1 + L_1$

r_c: $L_2 + R \to L_3$

If both possible, worst case is #R=1, whereas #C_k = 0 or 1.

How to handle the three problems

Recall three problems we claimed we would solve:

1. Adjusting rate constants means designing new chemicals.
2. Pr[error in any time step] increases for longer computations.
3. Reducing error slows down the computation “significantly”.

Problem 1: Now all rate constants = 1.

Problem 2: How to make Pr[error in any time step] < ε, no matter how long the computation goes?

$$F + C_1 \rightarrow F + C_2 \qquad B + C_2 \rightarrow B + C_1$$
$$F + C_2 \rightarrow F + C_3 \qquad B + C_3 \rightarrow B + C_2$$

Two competing reactions, r_i incorrect, and r_c correct:

r_i: $C_k + L_2 \rightarrow C_1 + L_1$

r_c: $L_2 + R \rightarrow L_3$

If both possible, worst case is $\#R=1$, whereas $\#C_k = 0$ or 1.

$\Pr[r_i] = \Pr[\#C_k = 1] \leq 1/n^k$, where n = $\#B$.

How to handle the three problems

Recall three problems we claimed we would solve:

1. Adjusting rate constants means designing new chemicals.
2. Pr[error in <u>any</u> time step] increases for longer computations.
3. Reducing error slows down the computation "significantly".

<u>Problem 1</u>: Now all rate constants = 1.

<u>Problem 2</u>: How to make Pr[error in any time step] < ε, no matter how long the computation goes?

$$F + C_1 \rightarrow F + C_2 \qquad B + C_2 \rightarrow B + C_1$$
$$F + C_2 \rightarrow F + C_3 \qquad B + C_3 \rightarrow B + C_2$$

Two competing reactions, r_i incorrect, and r_c correct:

r_i: $C_k + L_2 \rightarrow C_1 + L_1$

r_c: $L_2 + R \rightarrow L_3$

If both possible, worst case is $\#R$=1, whereas $\#C_k$ = 0 or 1.

$\Pr[r_i] = \Pr[\#C_k = 1] \leq 1/n^k$, where $n = \#B$.

Setting $k = 2$, $\Pr[r_i] \leq 1/n^2$.

How to handle the three problems

Recall three problems we claimed we would solve:

1. Adjusting rate constants means designing new chemicals.
2. Pr[error in <u>any</u> time step] increases for longer computations.
3. Reducing error slows down the computation "significantly".

<u>Problem 1</u>: Now all rate constants = 1.

<u>Problem 2</u>: How to make Pr[error in any time step] < ε, no matter how long the computation goes?

$$F + C_1 \longrightarrow F + C_2 \qquad B + C_2 \longrightarrow B + C_1$$
$$F + C_2 \longrightarrow F + C_3 \qquad B + C_3 \longrightarrow B + C_2$$

Two competing reactions, r_{i} incorrect, and r_{c} correct:

r_{i}: $C_k + L_2 \rightarrow C_1 + L_1$

r_{c}: $L_2 + R \rightarrow L_3$

If both possible, worst case is $\#R$=1, whereas $\#C_k$ = 0 or 1.

Pr[r_{i}] = Pr[$\#C_k$ = 1] ≤ $1/n^k$, where n = $\#B$.

Setting k = 2, Pr[r_{i}] ≤ $1/n^2$.

<u>Solution</u>: increase B after every decrement and jump:

r_{i}: $C_k + L_2 \rightarrow C_1 + L_1 + B$

r_{c}: $L_2 + R \rightarrow L_3 + B$

How to handle the three problems

Recall three problems we claimed we would solve:

1. Adjusting rate constants means designing new chemicals.
2. Pr[error in <u>any</u> time step] increases for longer computations.
3. Reducing error slows down the computation "significantly".

<u>Problem 1</u>: Now all rate constants = 1.

<u>Problem 2</u>: How to make Pr[error in any time step] < ε, no matter how long the computation goes?

$F + C_1 \rightarrow F + C_2$

$F + C_2 \rightarrow F + C_3$

$B + C_2 \rightarrow B + C_1$

$B + C_3 \rightarrow B + C_2$

Two competing reactions, r_i incorrect, and r_c correct:

r_i: $C_k + L_2 \rightarrow C_1 + L_1$

r_c: $L_2 + R \rightarrow L_3$

If both possible, worst case is $\#R=1$, whereas $\#C_k = 0$ or 1.

$\Pr[r_i] = \Pr[\#C_k = 1] \leq 1/n^k$, where n = #B.

Setting $k = 2$, $\Pr[r_i] \leq 1/n^2$.

<u>Solution</u>: increase B after every decrement and jump:

r_i: $C_k + L_2 \rightarrow C_1 + L_1 + B$

r_c: $L_2 + R \rightarrow L_3 + B$

So Pr[r_i ever occurs when it shouldn't] $\leq \sum_{n=1}^{\infty} 1/n^2 = \pi^2/6$.

How to handle the three problems

Recall three problems we claimed we would solve:
1. Adjusting rate constants means designing new chemicals.
2. Pr[error in any time step] increases for longer computations.
3. Reducing error slows down the computation "significantly".

Problem 1: Now all rate constants = 1.

Problem 2: How to make Pr[error in any time step] < ε, no matter how long the computation goes?

$$F + C_1 \rightarrow F + C_2 \qquad B + C_2 \rightarrow B + C_1$$
$$F + C_2 \rightarrow F + C_3 \qquad B + C_3 \rightarrow B + C_2$$

Two competing reactions, r_i incorrect, and r_c correct:

r_i: $C_k + L_2 \rightarrow C_1 + L_1$

r_c: $L_2 + R \rightarrow L_3$

If both possible, worst case is #R=1, whereas #C_k = 0 or 1.

Pr[r_i] = Pr[#C_k = 1] ≤ $1/n^k$, where n = #B.

Setting k = 2, Pr[r_i] ≤ $1/n^2$.

Solution: increase B after every decrement and jump:

r_i: $C_k + L_2 \rightarrow C_1 + L_1 + B$

r_c: $L_2 + R \rightarrow L_3 + B$

So Pr[r_i ever occurs when it shouldn't] ≤ $\sum_{n=1}^{\infty} 1/n^2 = \pi^2/6$.

Still not a great probability bound, but we can scale that to any constant error probability ε by setting starting value of B:

How to handle the three problems

Recall three problems we claimed we would solve:

1. Adjusting rate constants means designing new chemicals.
2. Pr[error in <u>any</u> time step] increases for longer computations.
3. Reducing error slows down the computation “significantly”.

<u>Problem 1</u>: Now all rate constants = 1.

<u>Problem 2</u>: How to make Pr[error in any time step] < ε, no matter how long the computation goes?

$$F + C_1 \to F + C_2 \qquad B + C_2 \to B + C_1$$
$$F + C_2 \to F + C_3 \qquad B + C_3 \to B + C_2$$

Two competing reactions, r_i incorrect, and r_c correct:

r_i: $C_k + L_2 \to C_1 + L_1$

r_c: $L_2 + R \to L_3$

If both possible, worst case is #R=1, whereas #C_k = 0 or 1.

Pr[r_i] = Pr[#C_k = 1] ≤ $1/n^k$, where n = #B.

Setting k = 2, Pr[r_i] ≤ $1/n^2$.

<u>Solution</u>: increase B after every decrement and jump:

r_i: $C_k + L_2 \to C_1 + L_1 + B$

r_c: $L_2 + R \to L_3 + B$

So Pr[r_i ever occurs when it shouldn’t] ≤ $\sum_{n=1}^{\infty} 1/n^2 = \pi^2/6$.

Still not a great probability bound, but we can scale that to any constant error probability ε by setting starting value of B:

For ε = 1/100, set initial #B = 102, since $\sum_{n=102}^{\infty} 1/n^2 < 0.01$.

How to handle the three problems

Recall three problems we claimed we would solve:
1. Adjusting rate constants means designing new chemicals.
2. Pr[error in any time step] increases for longer computations.
3. Reducing error slows down the computation "significantly".

Problem 1: Now all rate constants = 1.

Problem 2: How to make Pr[error in any time step] < ε, no matter how long the computation goes?

$$F + C_1 \rightarrow F + C_2 \qquad B + C_2 \rightarrow B + C_1$$
$$F + C_2 \rightarrow F + C_3 \qquad B + C_3 \rightarrow B + C_2$$

Two competing reactions, r_i incorrect, and r_c correct:

r_i: $C_k + L_2 \rightarrow C_1 + L_1$

r_c: $L_2 + R \rightarrow L_3$

If both possible, worst case is #R=1, whereas #C_k = 0 or 1.

Pr[r_i] = Pr[#C_k = 1] ≤ $1/n^k$, where n = #B.

Setting k = 2, Pr[r_i] ≤ $1/n^2$.

Solution: increase B after every decrement and jump:

r_i: $C_k + L_2 \rightarrow C_1 + L_1 + B$

r_c: $L_2 + R \rightarrow L_3 + B$

So Pr[r_i ever occurs when it shouldn't] ≤ $\sum_{n=1}^{\infty} 1/n^2$ = $\pi^2/6$.

Still not a great probability bound, but we can scale that to any constant error probability ε by setting starting value of B:

For ε = 1/100, set initial #B = 102, since $\sum_{n=102}^{\infty} 1/n^2 < 0.01$.

Problem 3: Also solved! i.e., halving error probability no longer doubles computation time (*derivation not shown*)