A Self-Replicating System of Ribosome and Replisome Factories

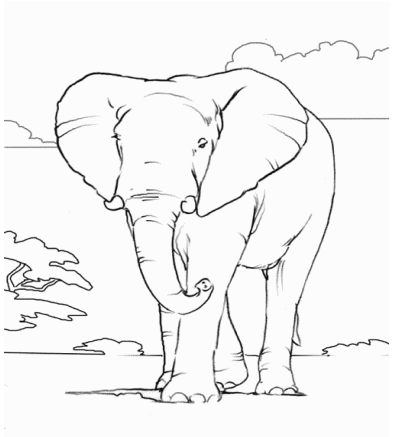
Lance R. Williams
Dept. of Computer Science
University of New Mexico

An Obscure Game invented by John von Neumann

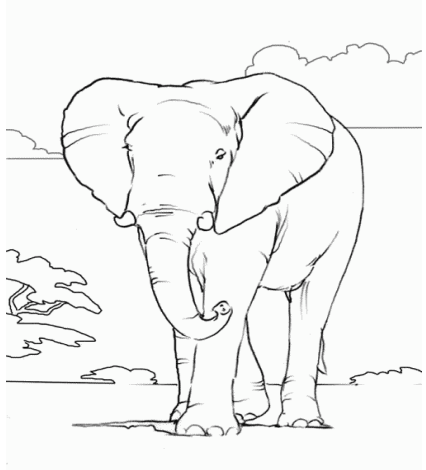
- Two parts
  - design a virtual world
  - design a self-replicating system with a non-trivial construction space in that world
- Two pitfalls
  - make the world too abstract
  - make the primitives too complex

Non-Trivial Construction Space

programmable
constructor
a.k.a. elephant

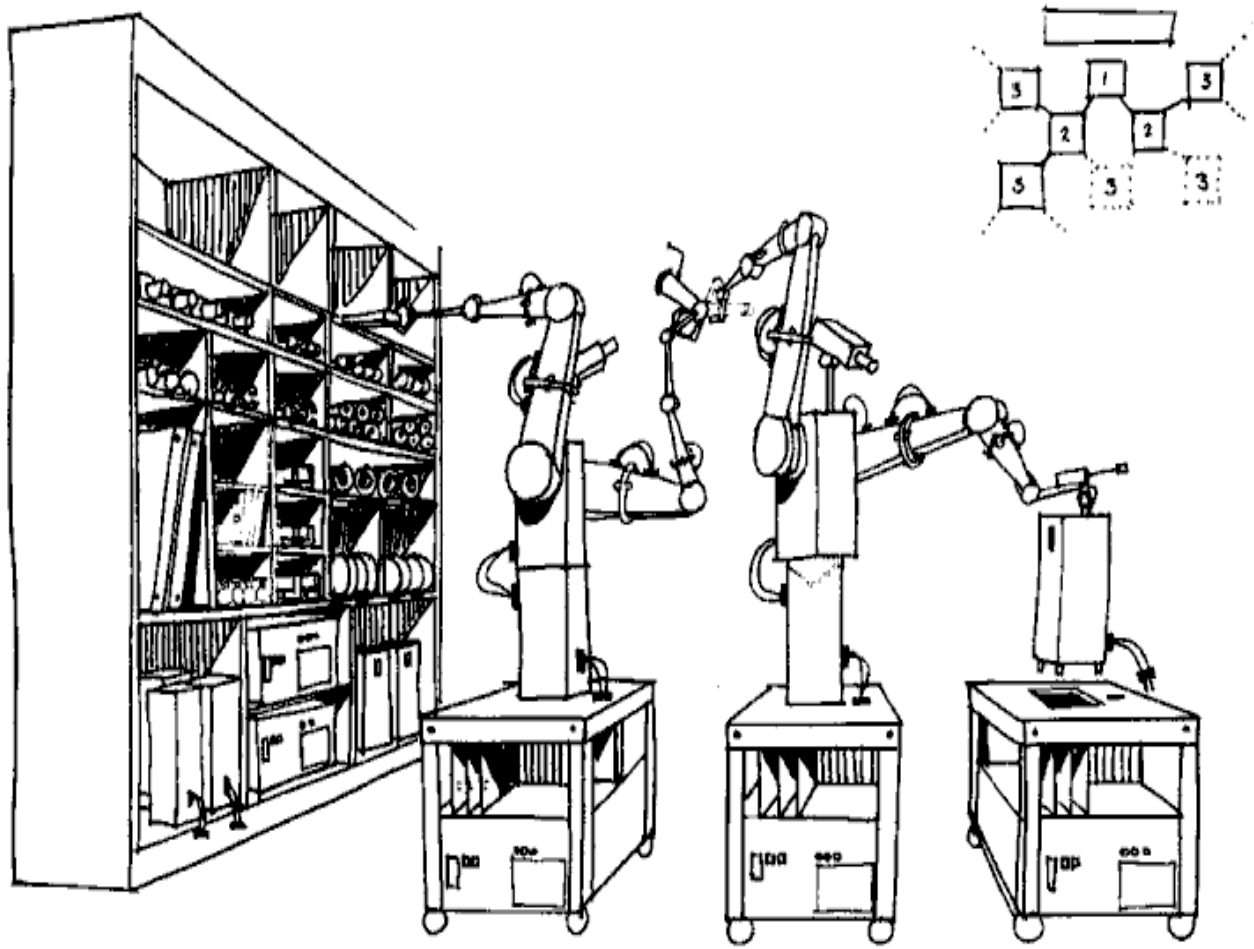


elephant DNA

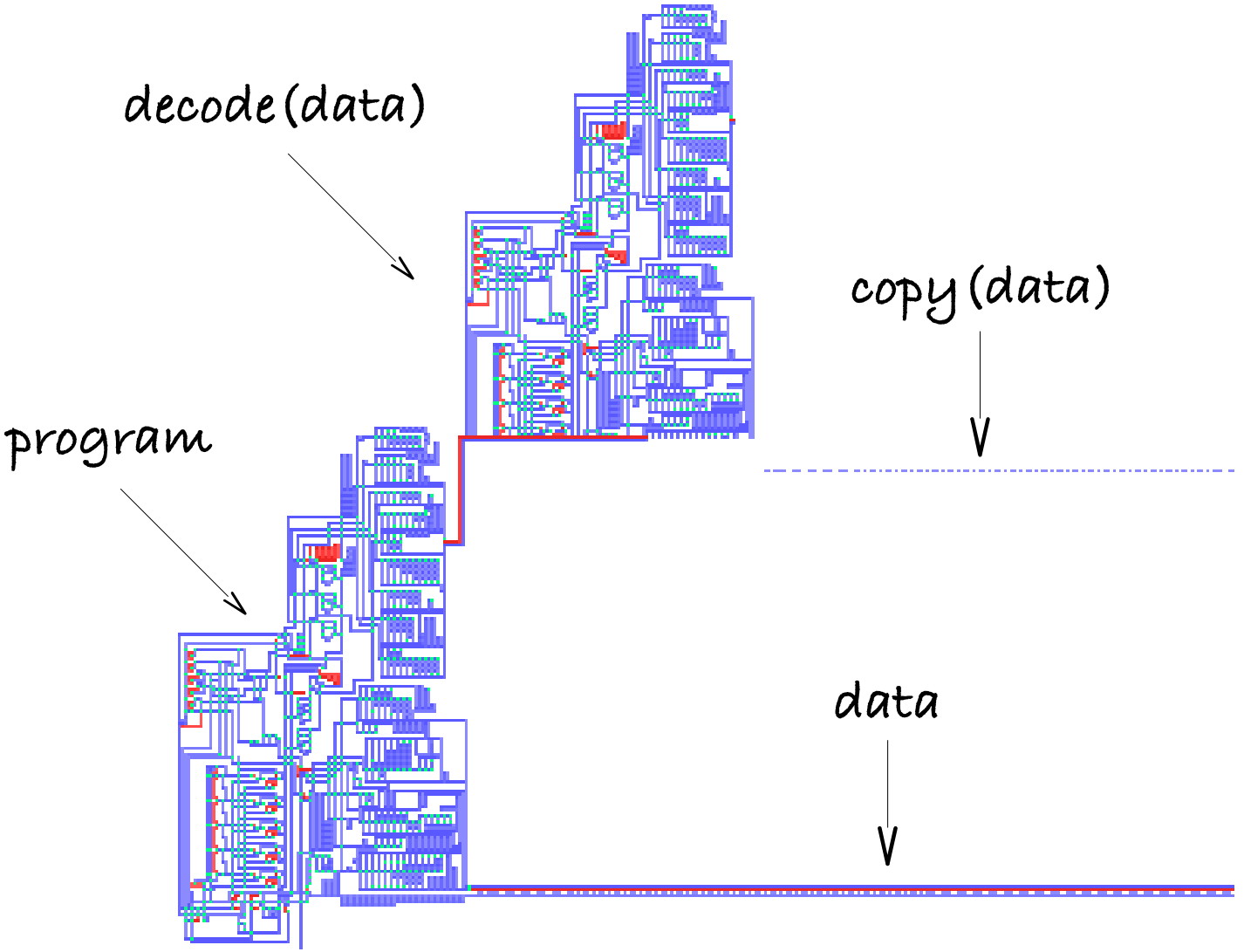


mammoth DNA

Complex Primitives



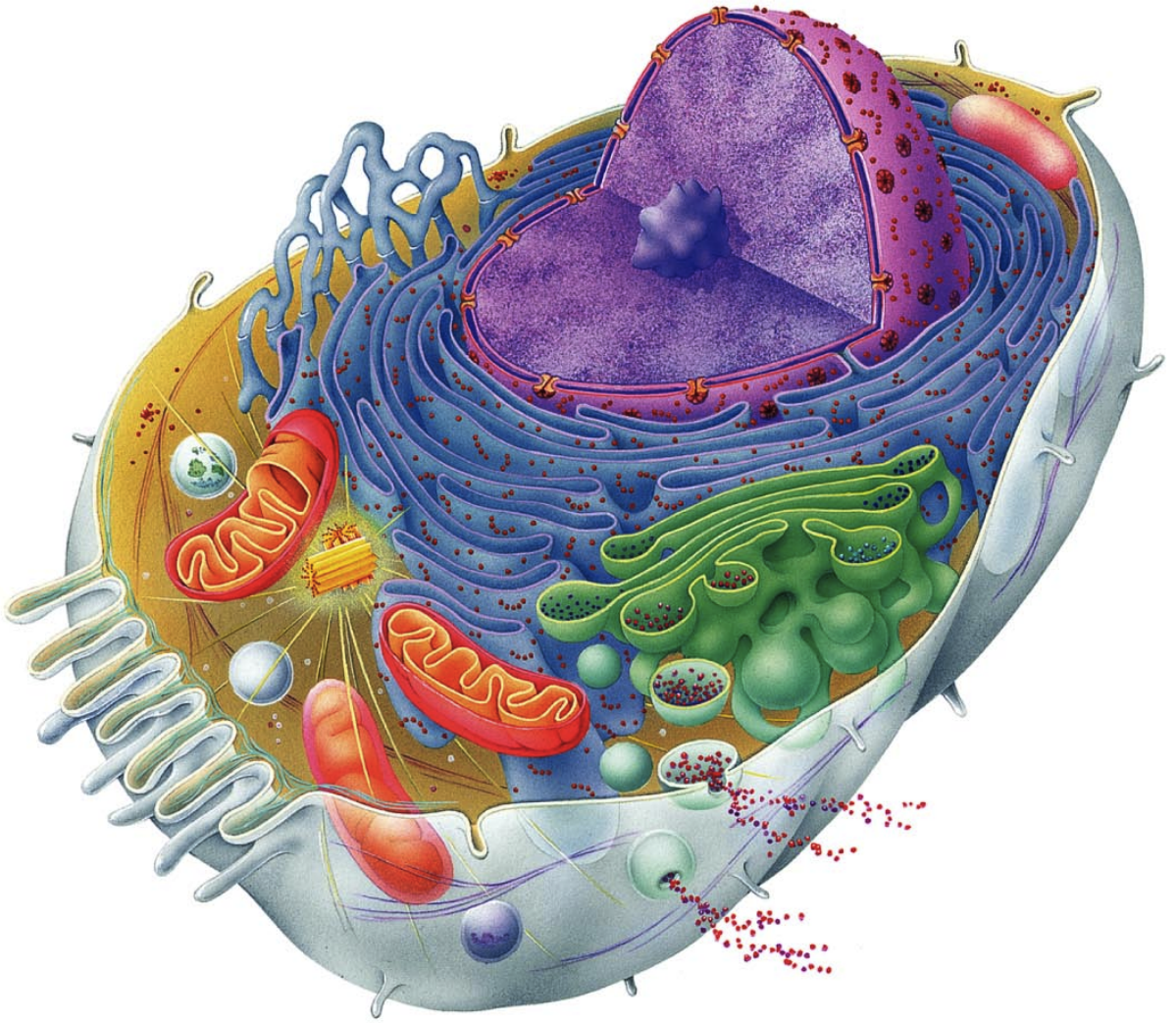
Von Neumann Replicator



Von Neumann Replicator

- *Span* – non-trivial construction space size
- *Modularity* – relative complexity of a system's parts and whole
- *Regularity* – reuse of complex parts
- *Parallelism* – concurrent construction subprocesses
- *Hierarchy* – complex parts comprised of simpler parts

Mechanism and Product of Evolution



Avoiding Excess Abstraction

- A *bespoke physics (Ackley '14)* is a software interface that
  - defines basic units embedded in space
  - dynamical laws that describe how the units move and interact
  - is subject to *meta-laws* including
    - *indefinite scalability*
    - *global-non-determinism.*
  - computations can be compiled to *asynchronous cellular automata (ACA).*

Avoiding Complex Primitives

- Use (admittedly) complex primitives to build parts that are *still more complex.*
- Interactions among parts result in the construction of more of these same parts.
- Systems like this are *strongly constructive (Dittrich et al. '01).*

Object-Oriented Combinator Chemistry

- An *artificial chemistry* is a dynamical system of constructable objects *(Fontana and Buss '96).*
- Object-Oriented Combinator Chemistry (OOCC)
  - An artificial chemistry with composition devices borrowed from the field of programming languages
    - Object-Oriented programming
      - association of programs with the data they operate on
    - Functional programming
      - programs comprised of combinators
  - A bespoke physics with an additional meta-law
    - C*onservation of mass*

Basic Units: Actors

- Actors occupy sites on a 2D grid.
- Computations progress when actors interact with other actors in their 8 neighborhoods.

Three Kinds of Actors

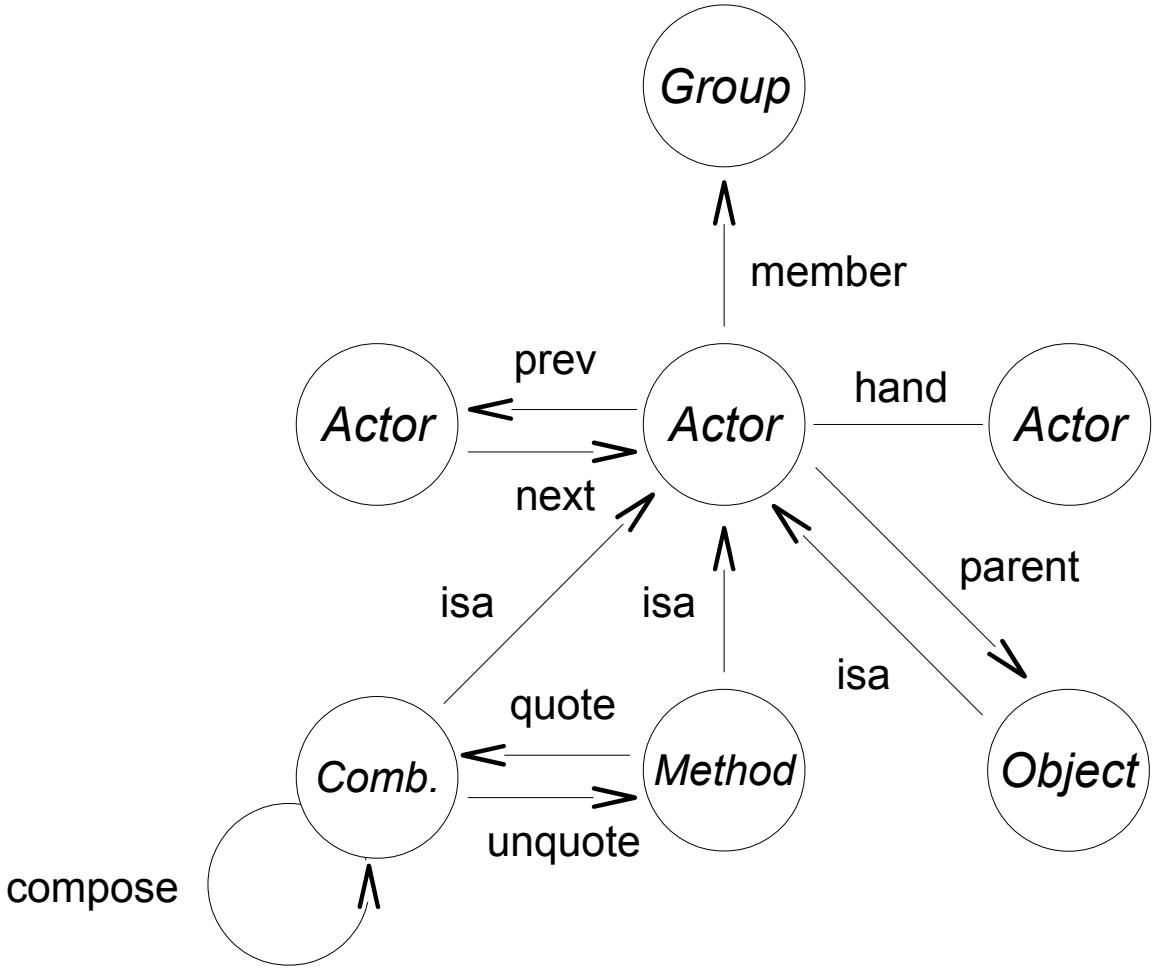
- Combinators

  - the building blocks of programs
- Methods
  - programs constructed from combinators
- Objects

  - can contain other actors (including other objects).

Actor Datatype

$$Actor = Combinator \mid Method \mid \{Actor\}_0 \mid \cdots \mid \{Actor\}_3$$
$$Combinator = c_1 \mid \cdots \mid c_N \mid Combinator >\!\!\Rightarrow Combinator \mid Method^{-}$$
$$Method = Combinator^{+}$$



Information = Mass

- Primitive combinators have unit mass.
- The mass of a composite combinator is the sum of the mass of the combinators of which it is composed.
- The mass of an object is the sum of the masses of the actors it contains.

Movability

- Movable aggregates of complex automata introduced in *Arbib '66*
  - Aggregrate automata have increased *area.*
  - Arbitrarily large aggregates assumed to move *O(1)* distance in *O(1)* time.
- Object-Oriented Combinator Chemistry
  - Composed actors have increased *mass.*
  - Composite actor of mass *m* can be moved *O(1)* distance in *O(m)* time.

Dynamics: Diffusion

- Actors are subject to random 2D motion.
- An actor's *diffusion constant* is inversely proportional to its mass.
- This reflects cost of data transport in ACA substrate.

Bonds

- Actors can create *bonds* with other actors in their neighborhoods.
- Bonds are *relative addresses* which are
  - short
  - symmetric
  - updated as actors move
- The movement of actors is restricted by bonds.
- Bonds can be either *directed* or *undirected*.

Groups

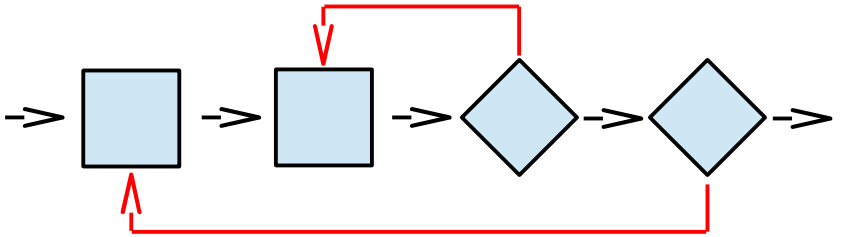
- Actors can join and quit *groups.*
- An actor is a member of exactly one group.
- Actors in a group
  - occupy a single site
  - diffuse as a unit
  - share a single finite time resource

Actors' Persistent States

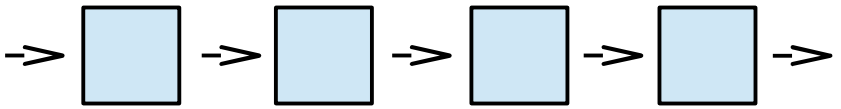
- Defined solely by
  - composition
  - containment
  - bonds
  - groups

Monadic Style

- Control idioms like iteration and backtracking require loops and function calls in conventional programming.
  - Both require address operands.



- No need for address operands in monadic code
  - Control is implicit in the datatype of the return value.
  - Programs exhibiting complex control idioms can be implemented by combinator sequences.



Non-Deterministic Choice

- Sets can be converted to *superpositions* using McCarthy's non-deterministic choice operator:

$$\text{amb}\,\{\,\} = \langle\,\rangle$$
$$\text{amb}\,\{x, y \ldots\} = \langle x, y \ldots\rangle$$

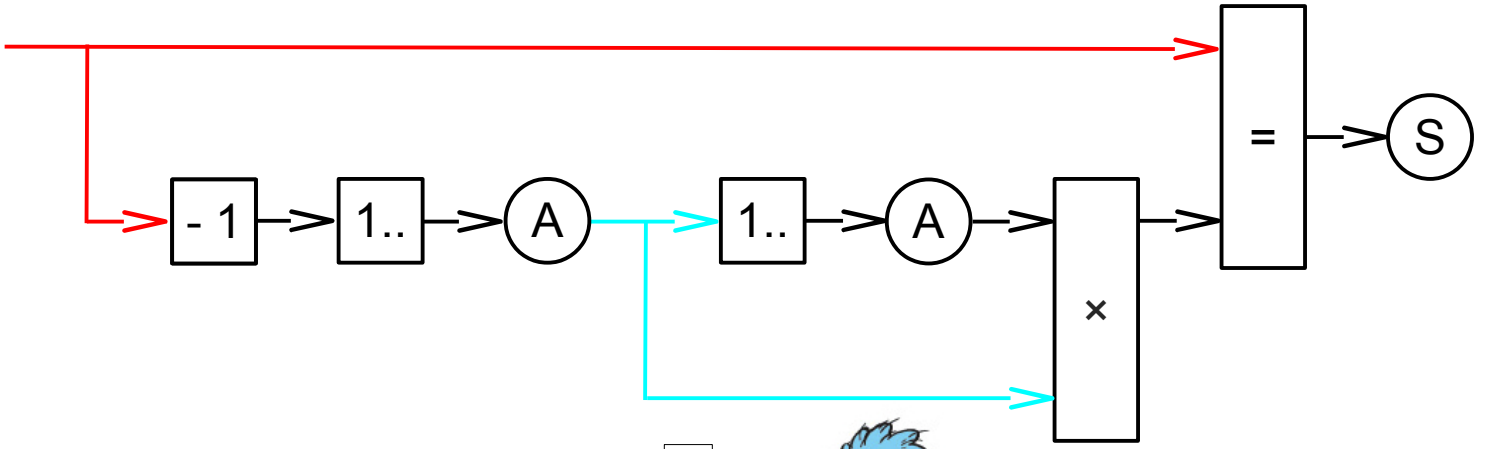
From Comprehensions to Combinators

$$\langle\, x \mid x \in \langle 1 \,..\, n-1 \rangle,\, y \in \langle 1 \,..\, x \rangle,\, x\,y = n \,\rangle$$

$\to \boxed{f'} \to :: \{a\} \to \langle\{a\}\rangle$

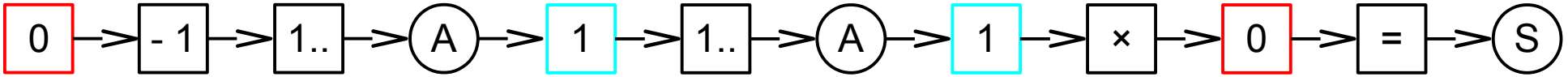
$\overset{\to}{\to} \boxed{g'} \to :: \{a\} \to \{a\} \to \langle\{a\}\rangle$

desugaring / unifying data types

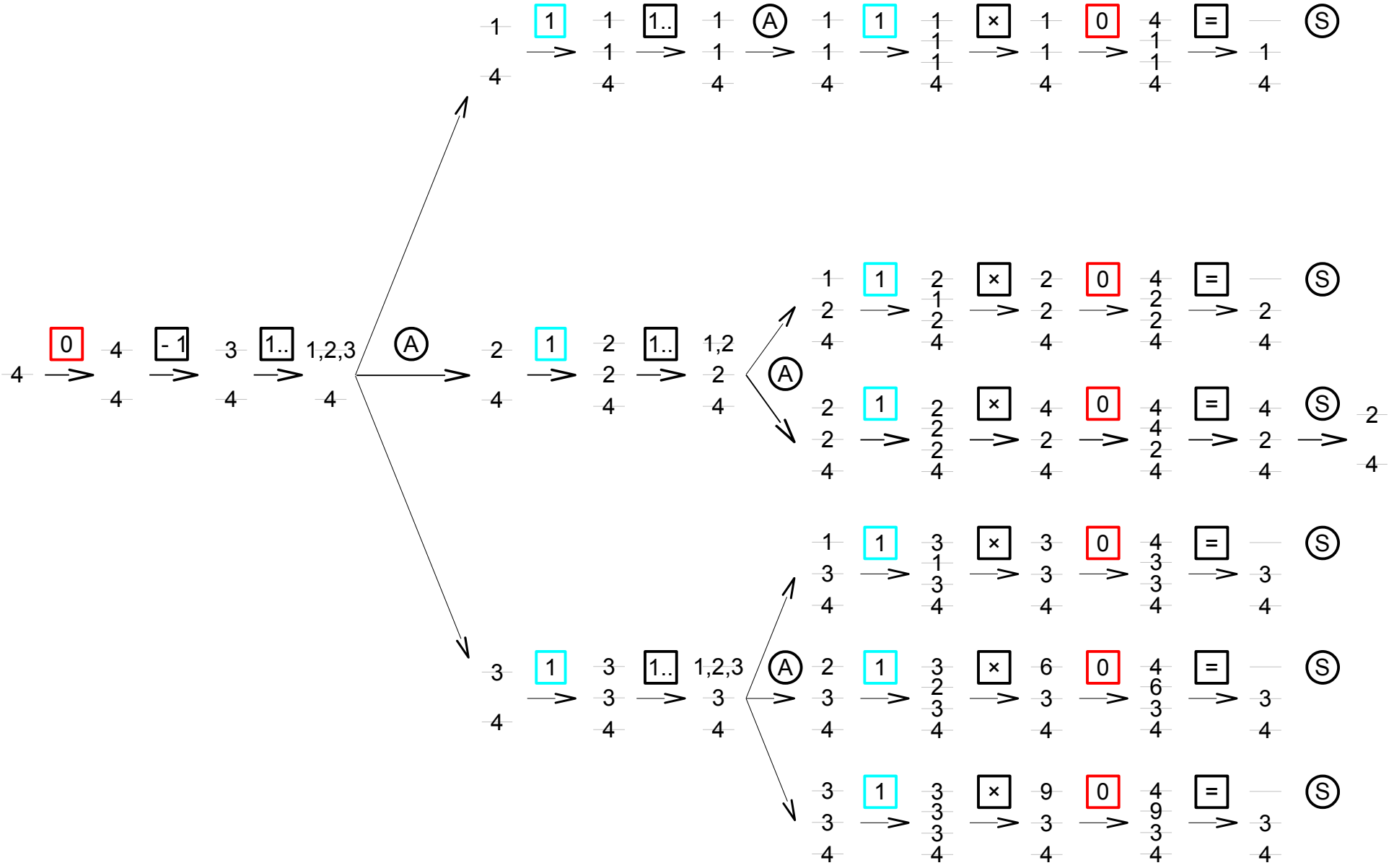


compiling

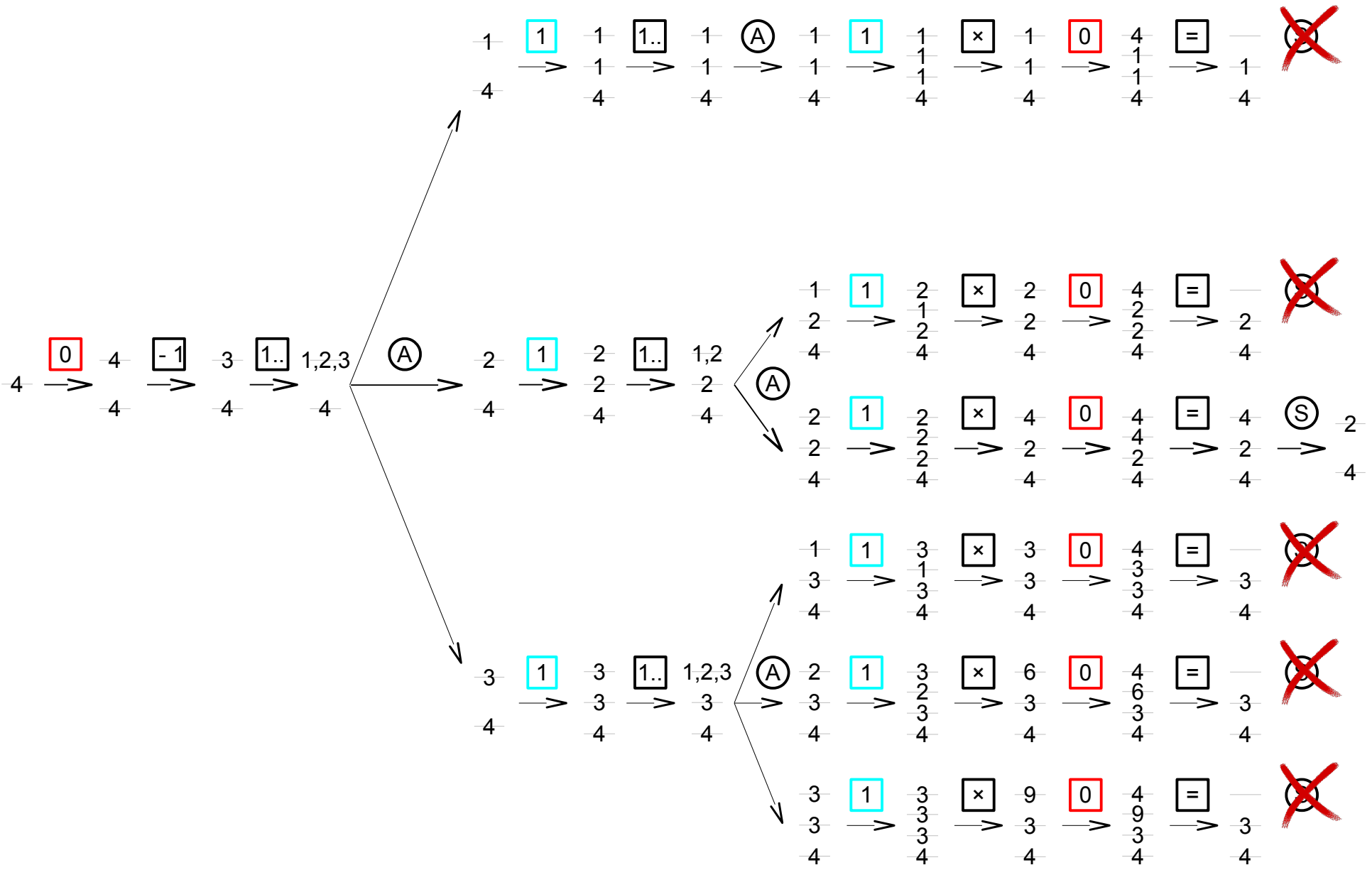
$\to \boxed{f''} \to :: [\{a\}] \to \langle[\{a\}]\rangle$



Non-deterministic Evaluation



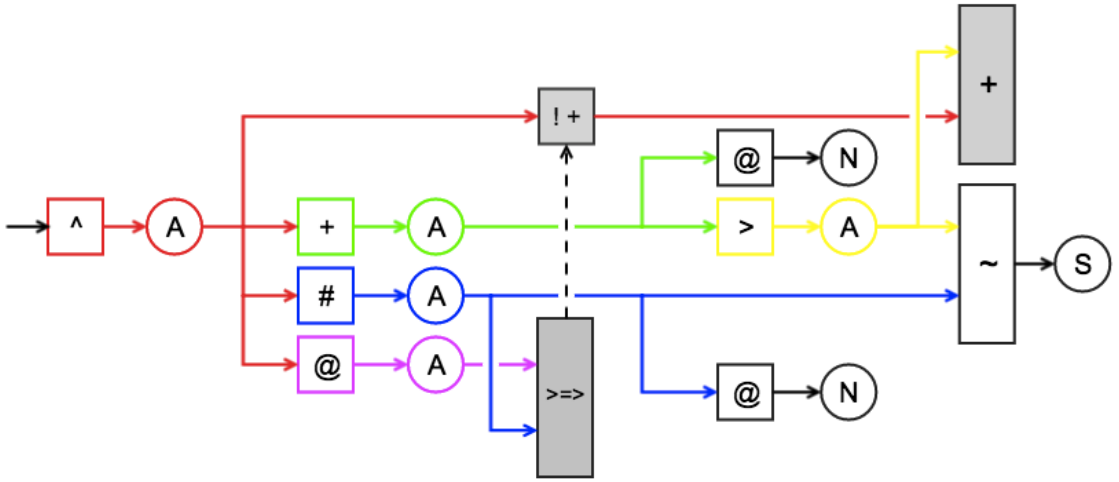
Non-deterministic Evaluation



Primitive Combinators

- `amb` introduces non-determinism.
- `bonds, members, contents, neighbors` reference actors using bonds, groups, containment and neighborhood.
- `similar, same, different` compare actors' identities and types.
- `some` and `none` allow methods to conditionally succeed or fail.
- `grab, drop, join, quit, compose, unquote` change actors' persistent states.

Visual Spock / Spock / Spasm



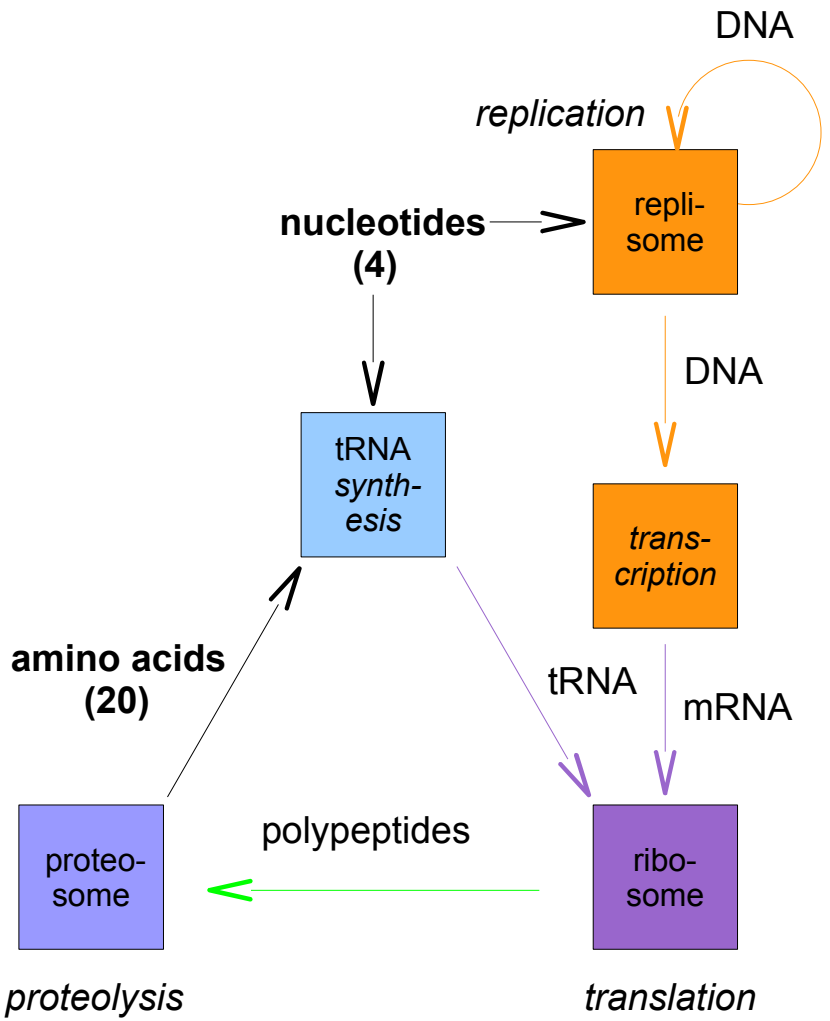

```
ribE = \m -> do {
  p <- amb =<< parents =<< m;
  q <- amb =<< others p;
  none =<< contents q;
  r <- amb =<< nexts q;
  n <- amb =<< neighbors p;
  some =<< similar n r;
  none =<< bonds n;
  c <- amb =<< contents p;
  compose c n;
  join r =<< quit p
}
```

```
parents >=> amb >=> x0 >=> others >=> amb >=> x1 >=> contents >=> none
  >=> x1 >=> nexts >=> amb >=> x0 >=> neighbors >=> amb >=> x2 >=> x3
  >=> similar >=> some >=> x3 >=> bonds >=> none >=> x0 >=> contents
  >=> amb >=> x3 >=> x4 >=> compose >=> x0 >=> quit >=> x2 >=> join
```

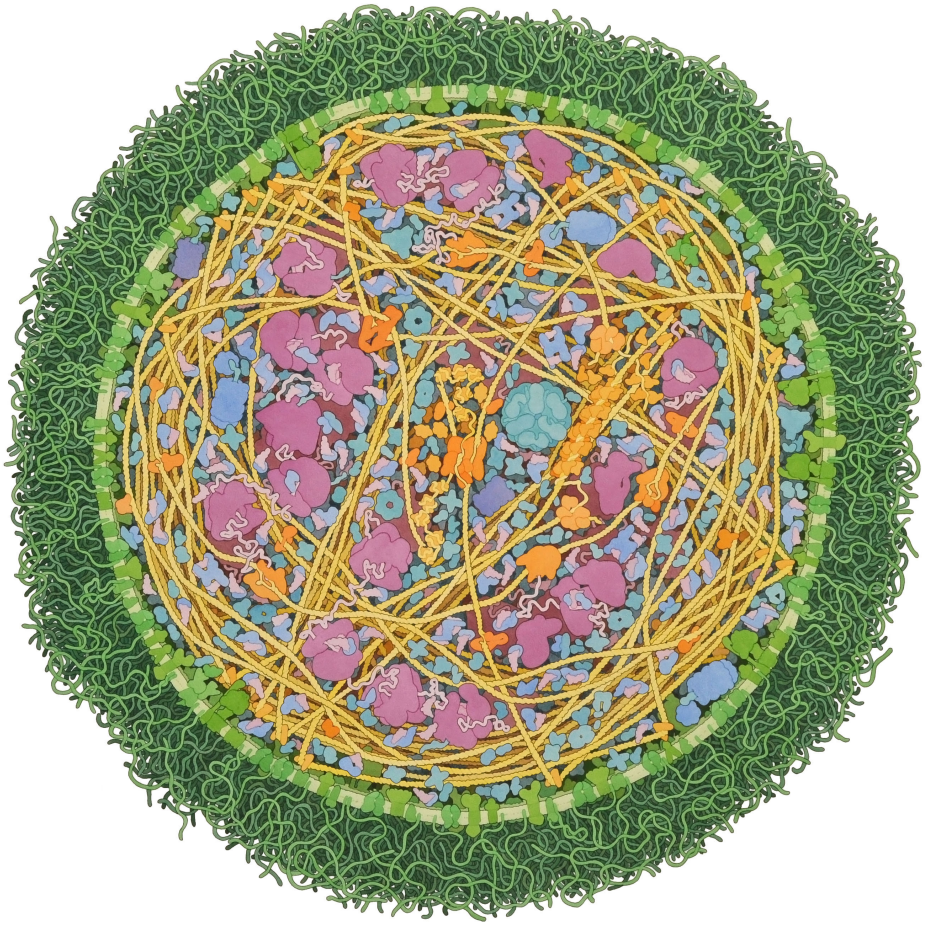
Time = Energy

- Methods require different amounts of time to do their work.
- Actors are serviced at rates which are the inverses of these times.
- Actors which use more time are serviced less often.
- Constant rate of energy use per site per unit time in ACA substrate.

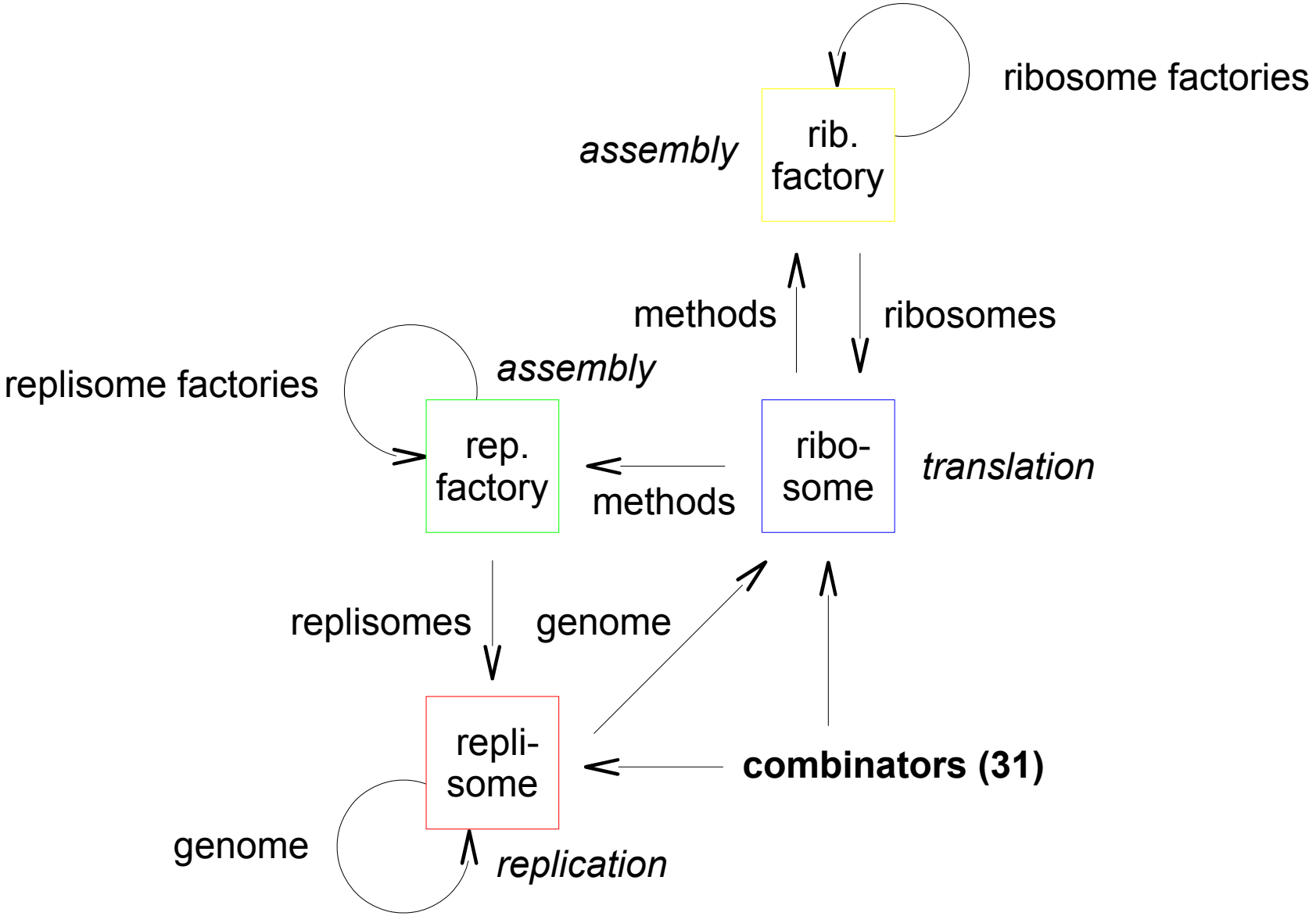
Fundamental Dogma of Molecular Biology



Mycoplasma mycoides



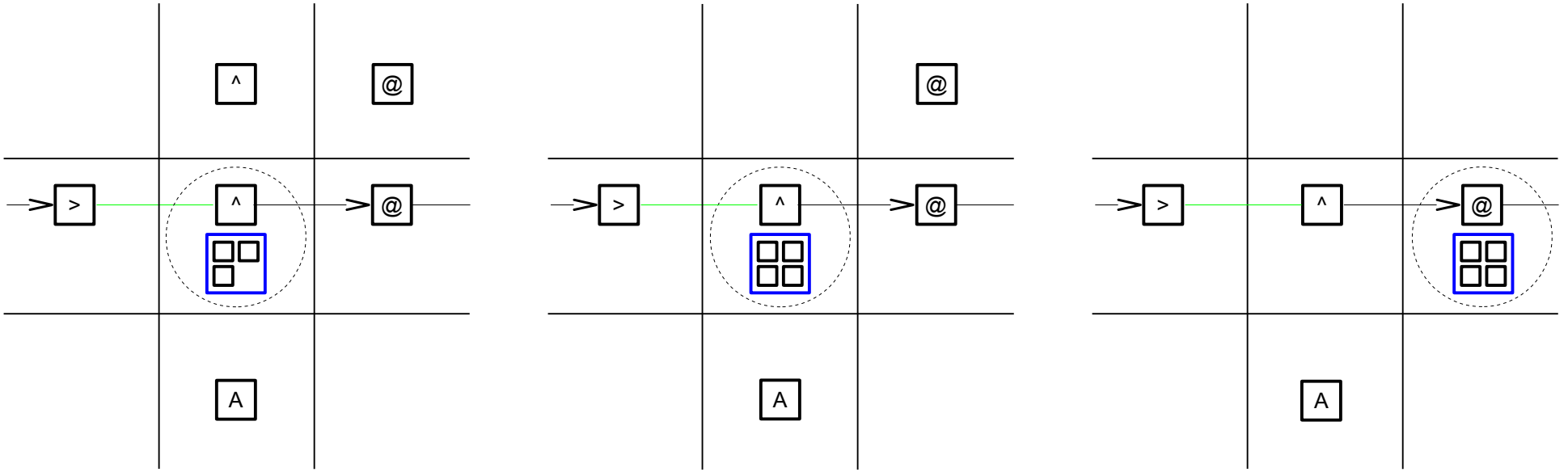
Self-Replicating System of Ribosome and Replisome Factories



Self-Replicating System of Ribosome and Replisome Factories

- *Span* – non-trivial construction space
- *Modularity* – relative complexity of a system's parts and whole
- *Regularity* – reuse of complex parts
- *Parallelism* – concurrent construction subprocesses
- *Hierarchy* – complex parts comprised of simpler parts

Computational Ribosome



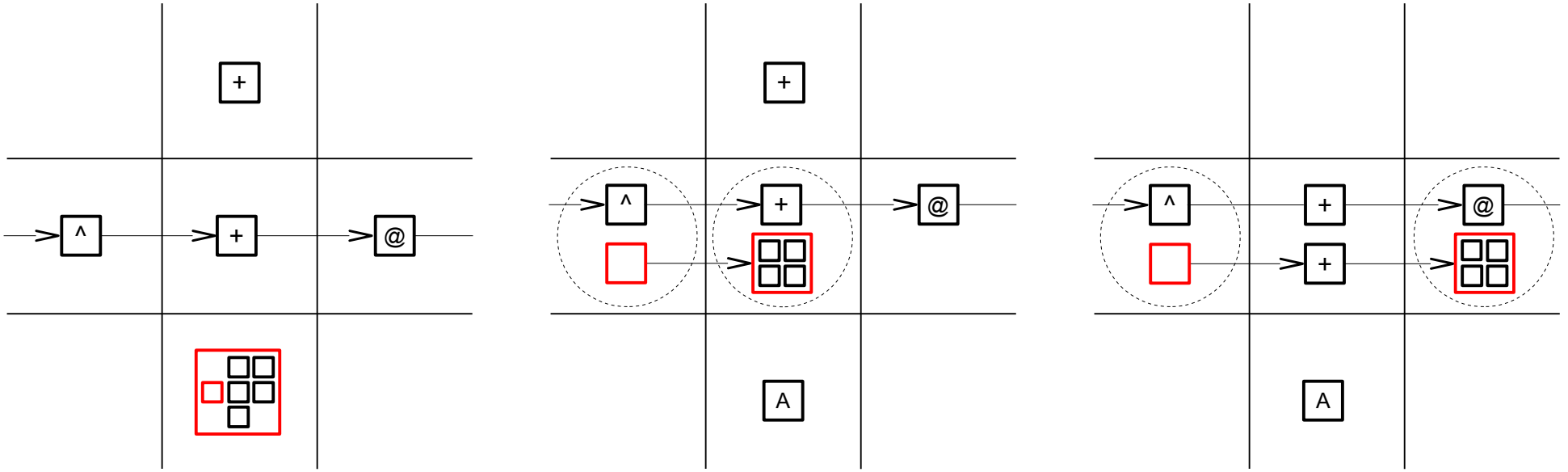
$$P + R + \sum_P \sum_C h(p,c)\, c \rightarrow P + R' + \mathrm{ribA} + \sum_P E_p$$

$$P = |\,{}_{i=1}^{|P|} >_{j=1}^{|G_i|} c_i(j)$$

$$R = \{\mathrm{ribA},\ \mathrm{ribI},\ \mathrm{ribE},\ \mathrm{ribT}\,\}_0$$

$$E_i = (\ >\!\!\Rightarrow_{j=1}^{|G_i|}\ c_i(j)\ )^+$$

Computational Replisome

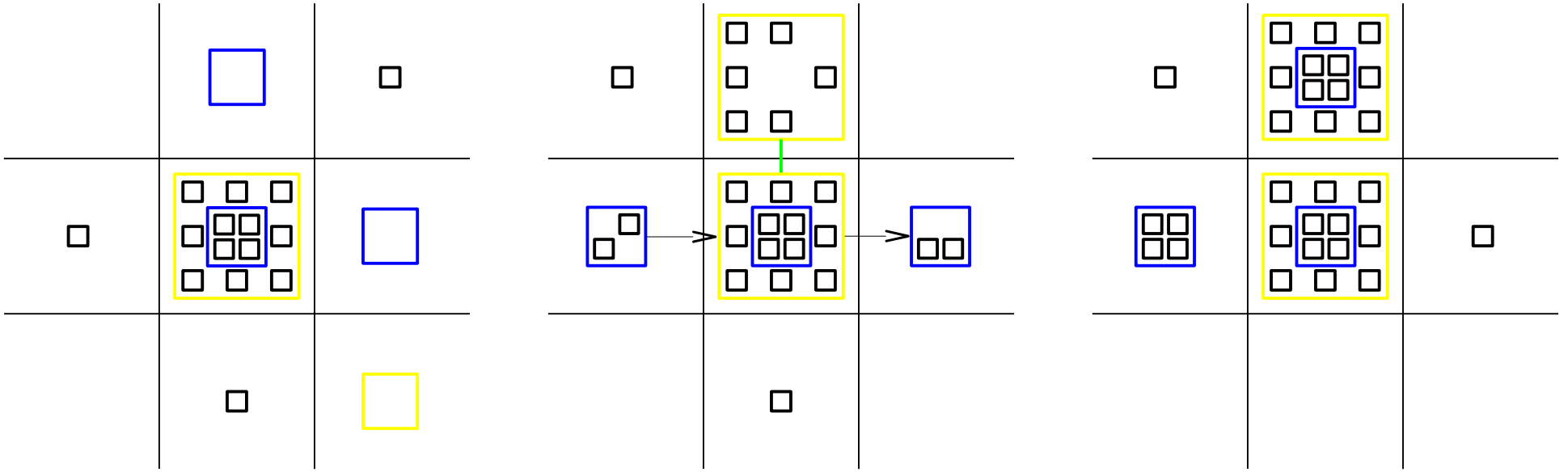


$$P + Q + \sum_P \sum_C h(p, c)\, c \rightarrow 2\, P + Q' + \mathrm{repA} + \{\,\}_2$$

$$P = |\,{}_{i=1}^{|P|} >_{j=1}^{|G_i|} c_i(j)$$

$$Q = \{\mathrm{repA},\ \mathrm{repE},\ \mathrm{repF},\ \mathrm{repY},\ \mathrm{repZ},\ \{\,\}_2\,\}_2$$

Ribosome-factory



$$F_R + 2\{\,\}_0 + \{\,\}_1 + 2\sum_R E_r + \sum_F E_f \rightarrow 2\,F_R + R$$

$$F_R = \{\mathrm{facP},\ \mathrm{facN},\ \mathrm{facH},\ \mathrm{facU}, \mathrm{facV},\ \mathrm{facX},\ \mathrm{facY},\ \mathrm{facZ},\ R\,\}_1$$

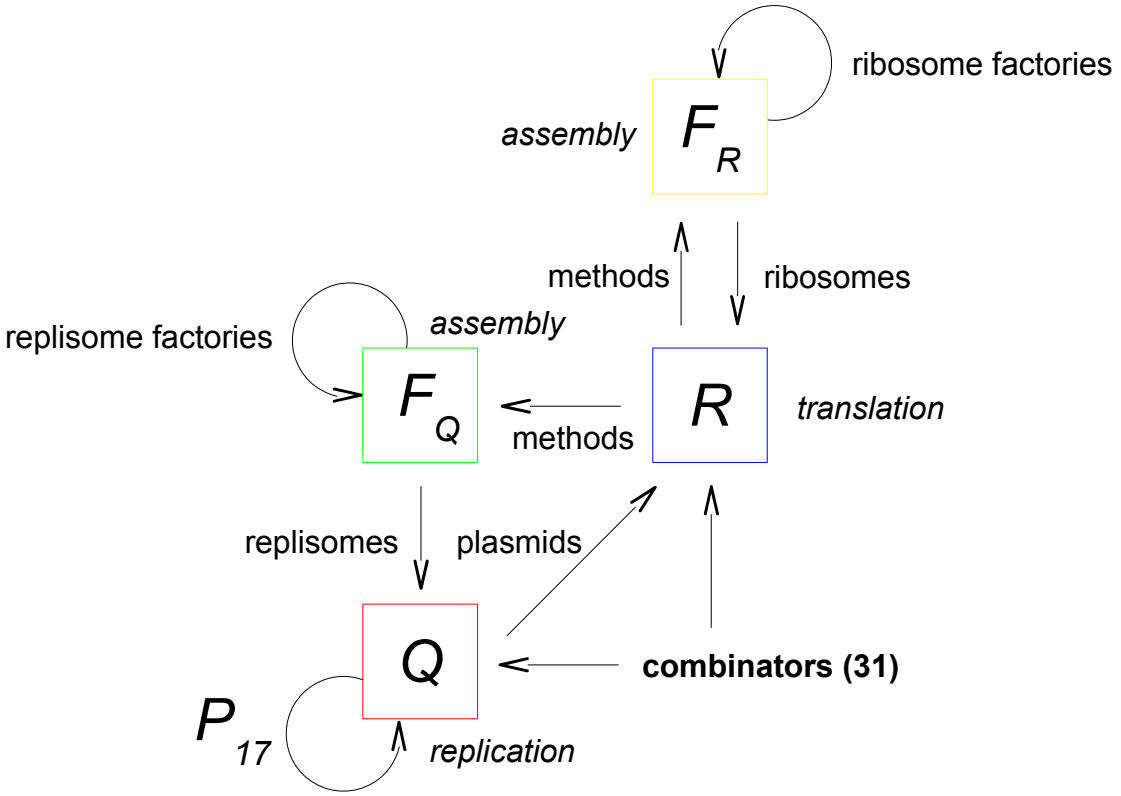
$$R = \{\mathrm{ribA},\ \mathrm{ribI},\ \mathrm{ribE},\ \mathrm{ribT}\,\}_0$$

$$E_i = (\ \gg\!\!\Rightarrow_{j=1}^{|G_i|}\ c_i(j)\ )^+$$

Self-Replicating System of Ribosome and Replisome Factories

$$P_{17} + F_Q + F_R + R + 2\{\,\}_0 + \{\,\}_1 + 4\{\,\}_2 + \{\,\}_3 + 3\sum\nolimits_{P_{17}} \sum\nolimits_{C} h(p,c)\, c \rightarrow$$

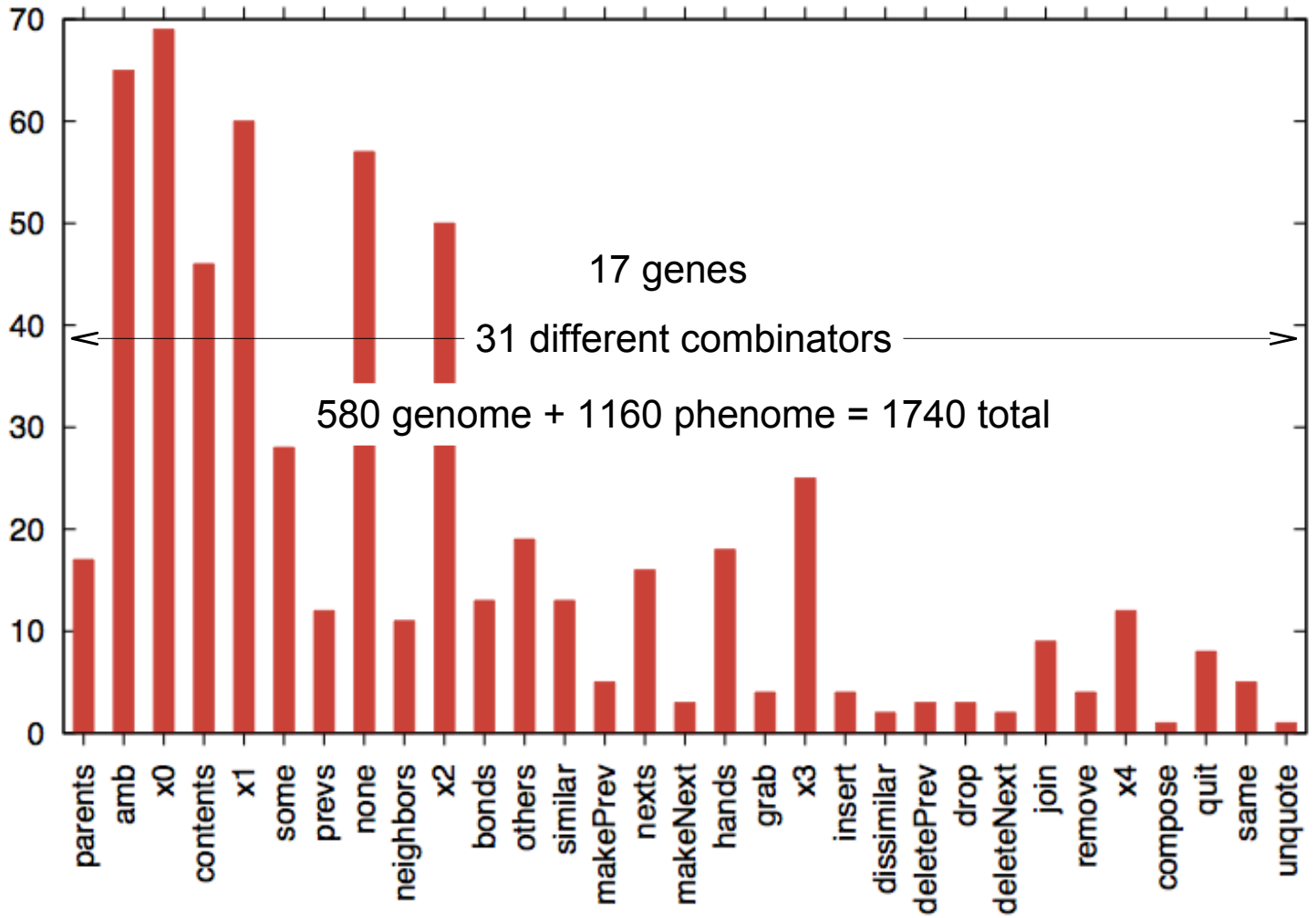
$$2\,P_{17} + 2\,F_Q + 2\,F_R + Q' + \mathrm{repA} + \{\,\}_2 + R + R' + \mathrm{ribA}$$



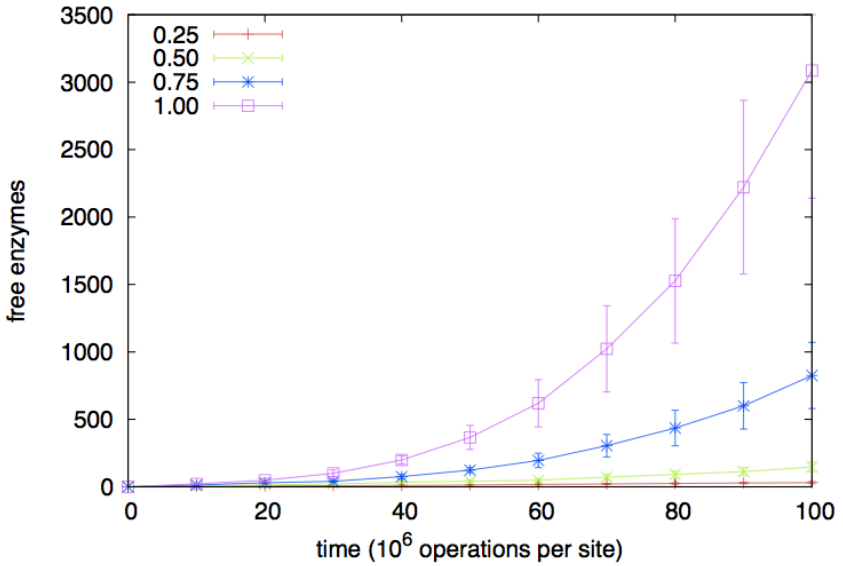
$$P_{17} = ribA \mid ribI \mid ribE \mid ribT \mid repA \mid repE \mid repF \mid repY \mid repZ$$

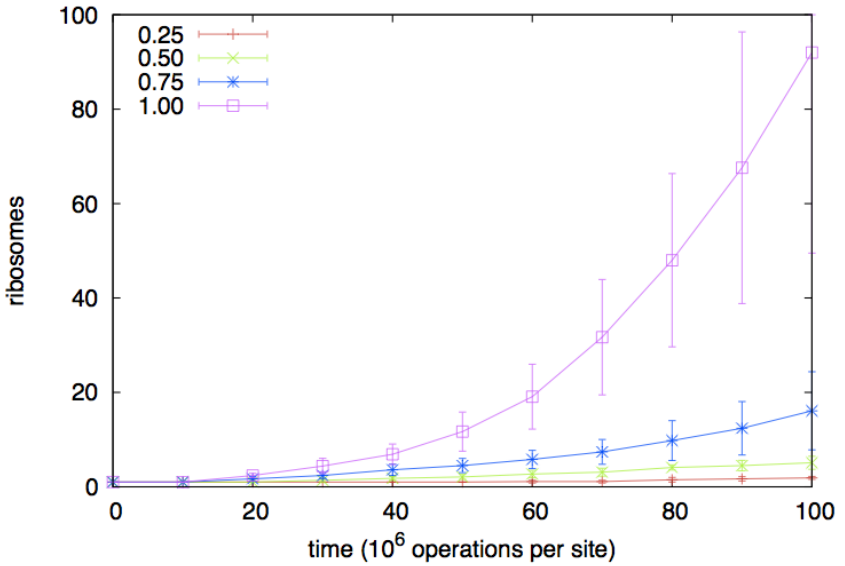
$$\mid facP \mid facN \mid facH \mid facU \mid facV \mid facX \mid facY \mid facZ$$

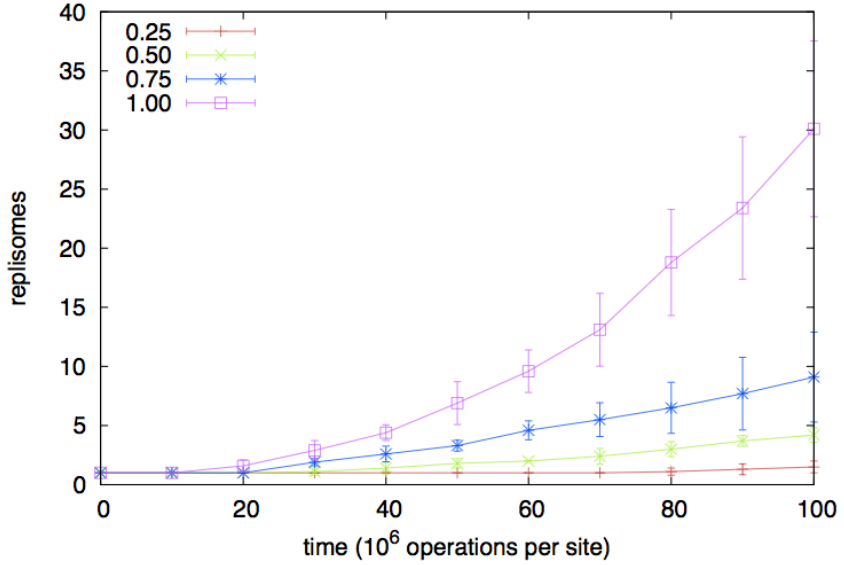
Self-Replicating System of Ribosome and Replisome Factories

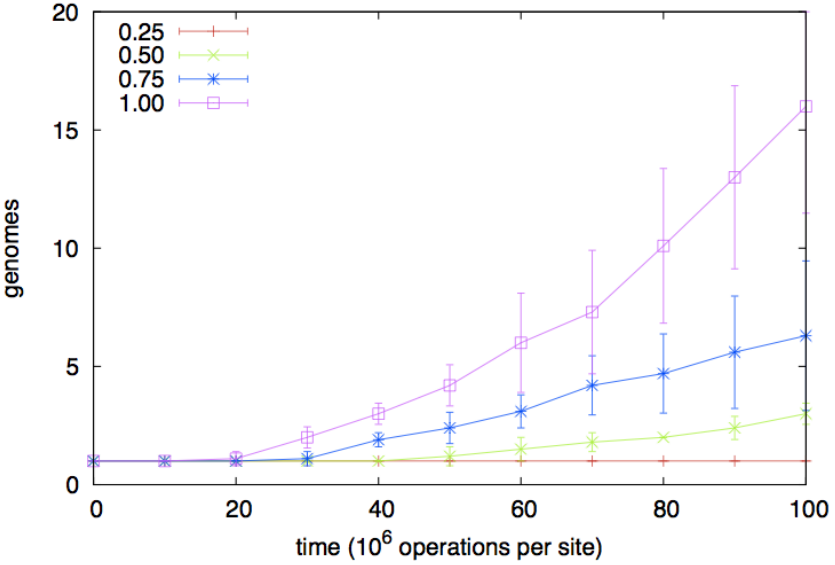


Populations vs. Time





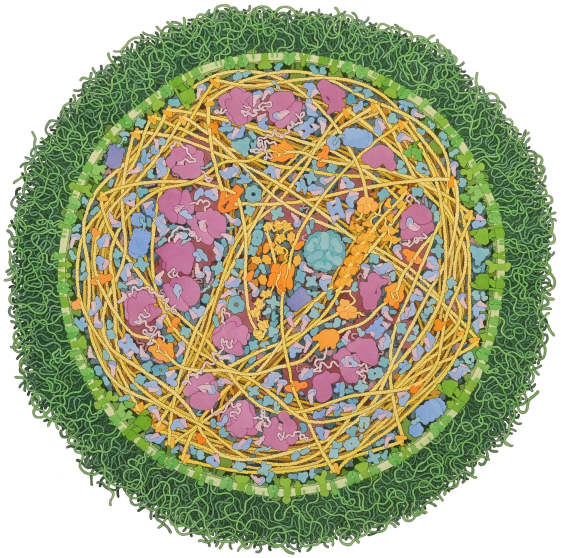




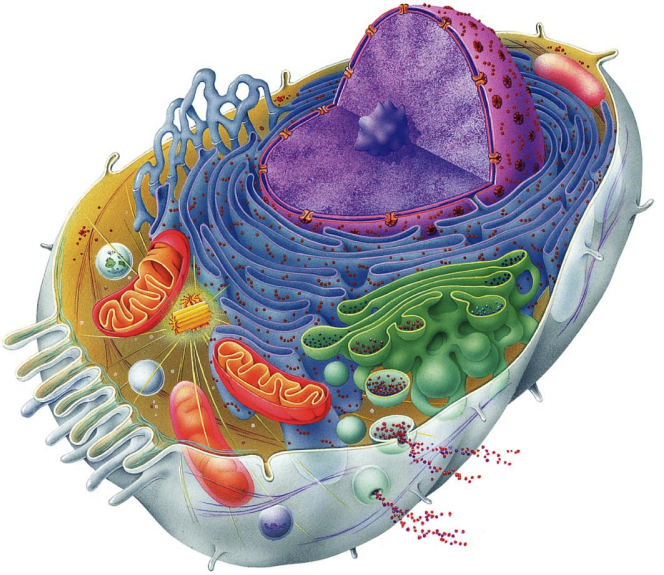
Near Term Goals

- Cell membrane
  - import / export
  - growth
  - binary fission
- Differential gene expression rates
- Homeostasis
- Cell cycle

Long Term Goal



evolution



Conclusion

- A novel artificial chemistry with composition devices borrowed from modern programming languages
- A self-replicating system modeled in part on the living cell

This work powered by Haskell