Memory Consistency Conditions for Self-Assembly Programming

Aaron Sterling

Laboratory for Nanoscale Self-Assembly

Department of Computer Science

Iowa State University

Take-away message

- Self-assembling systems can be simulated by models of distributed shared memory.
- Types of error unique to algorithmic DNA self-assembly can be simulated by weak memory consistency conditions for those DSM models.
- Hence, the theory of memory consistency, and the theory of self-stabilization, can be productively applied to questions of algorithmic self-assembly.

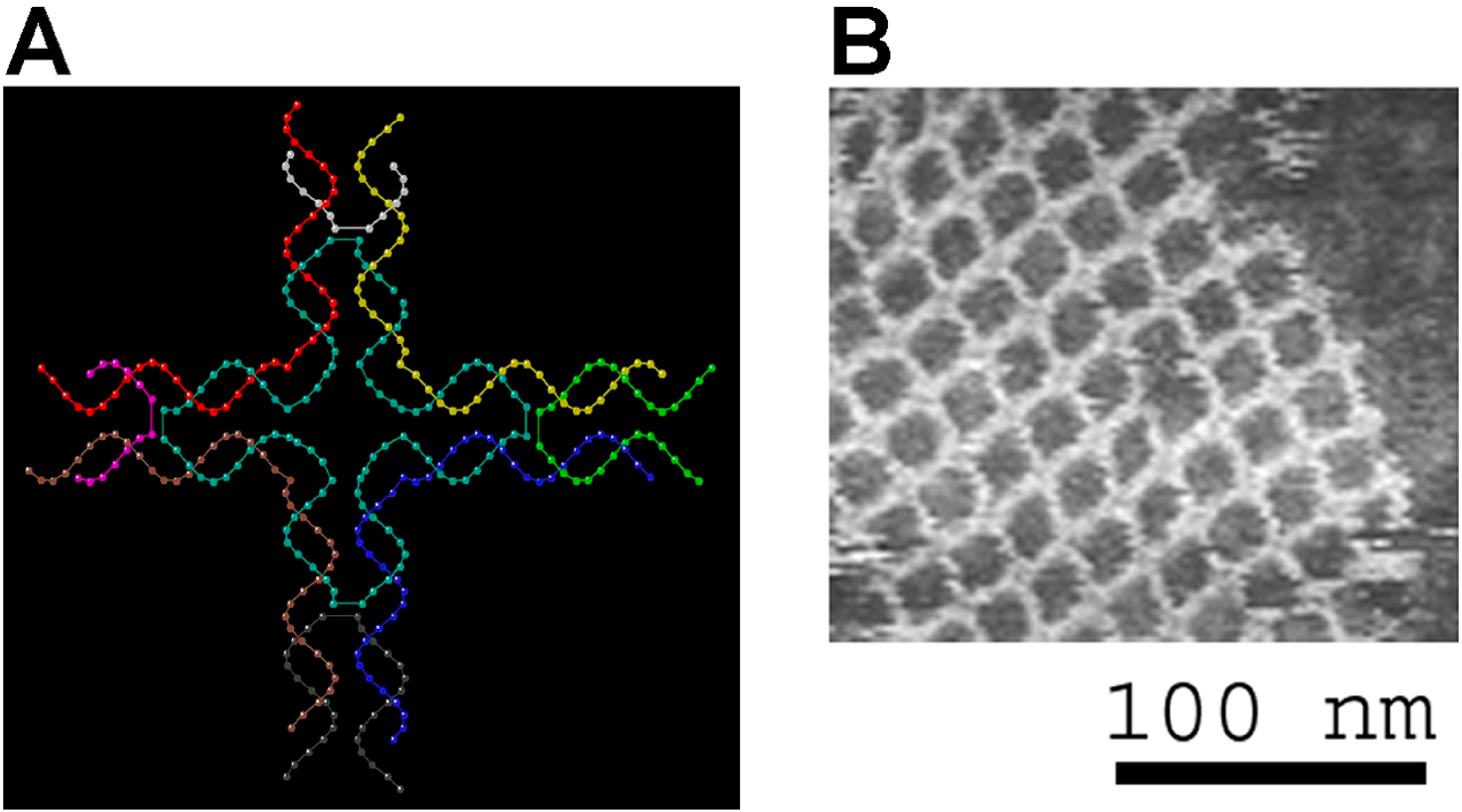
Take-away message

- The theory of multiprocessor architecture can be productively applied to biomolecular computing architecture!

Overview

- Introduction to algorithmic DNA self-assembly
- Introduction to distributed shared memory, and memory consistency conditions
- Sketch of reduction from self-assembly models to DSM models
- Two applications
- Conclusion, and preview of future work

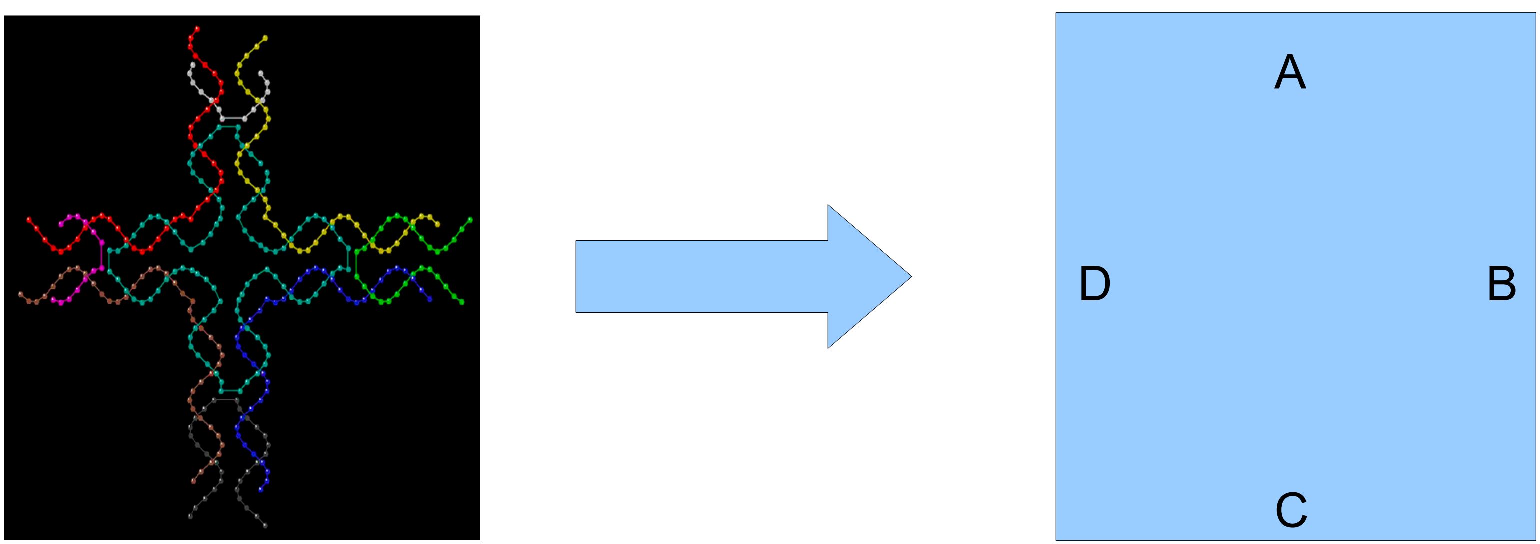
DNA self-assembly



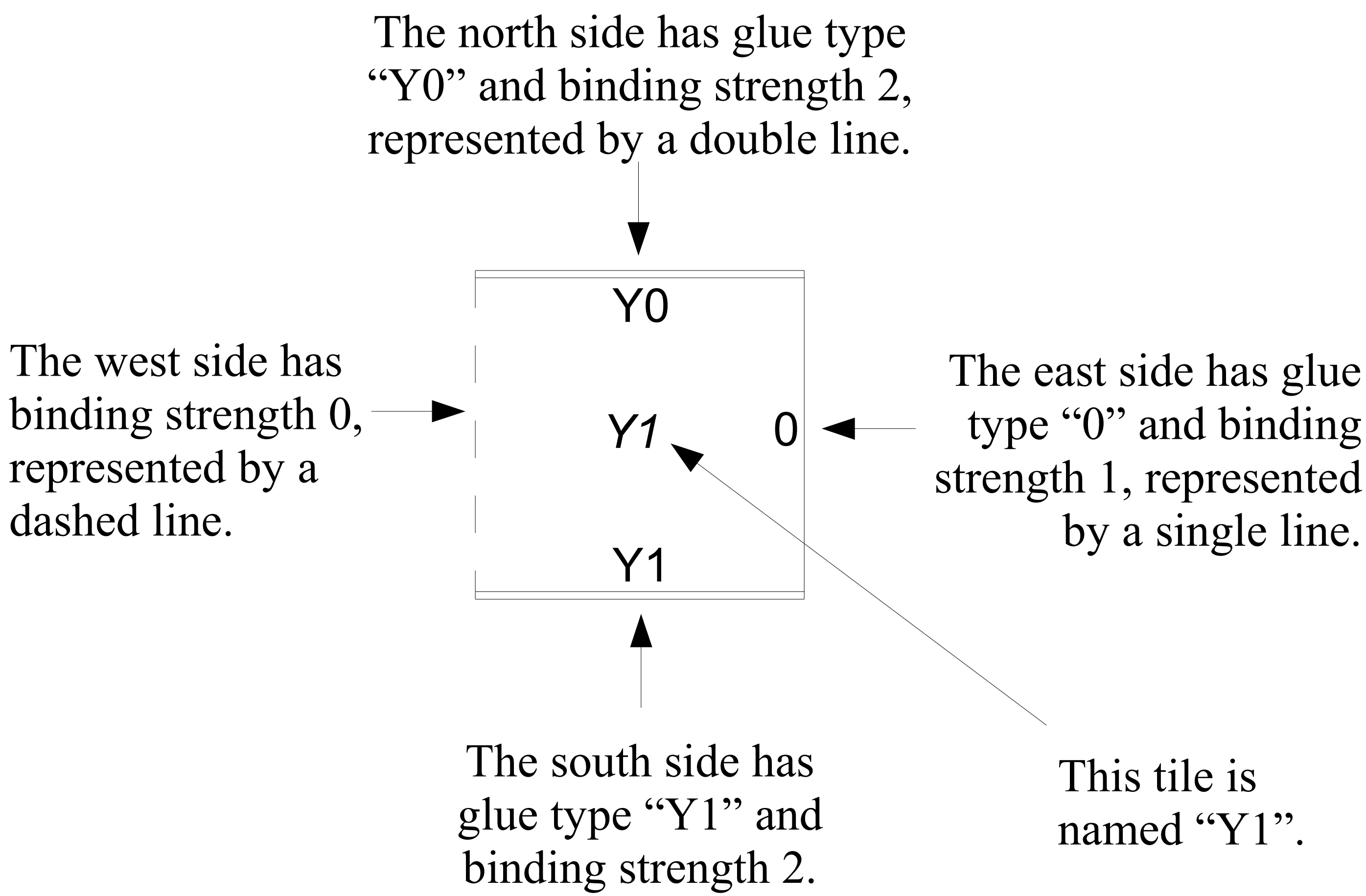
Source: Strong 2004

Algorithmic DNA self-assembly

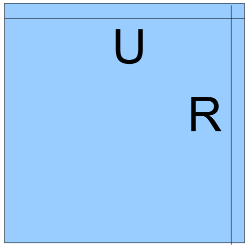
- Winfree's key insight [1995-8]:
  - DNA nanostructures with four “sticky ends” [Seeman] could be programmed by approximating them as a model of effectivized Wang tiling on the integer plane.



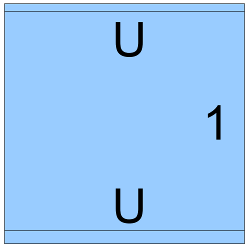
DNA tile self-assembly

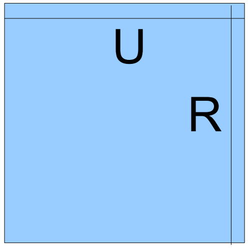


Programmable Self-Assembly

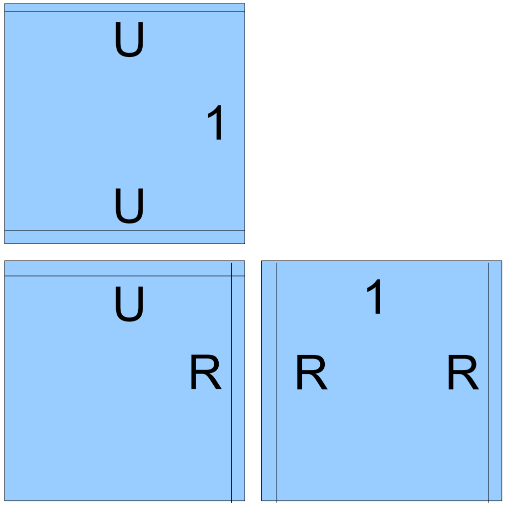


Programmable Self-Assembly

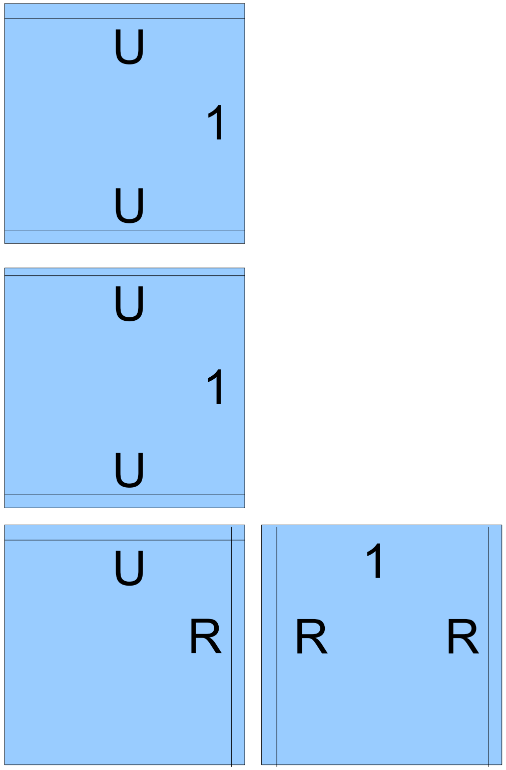




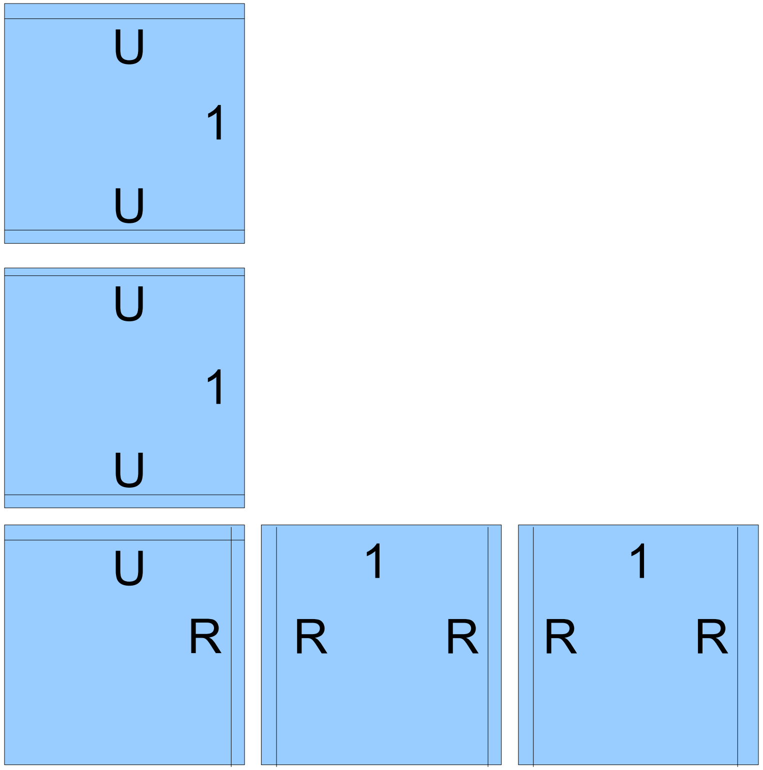
Programmable Self-Assembly



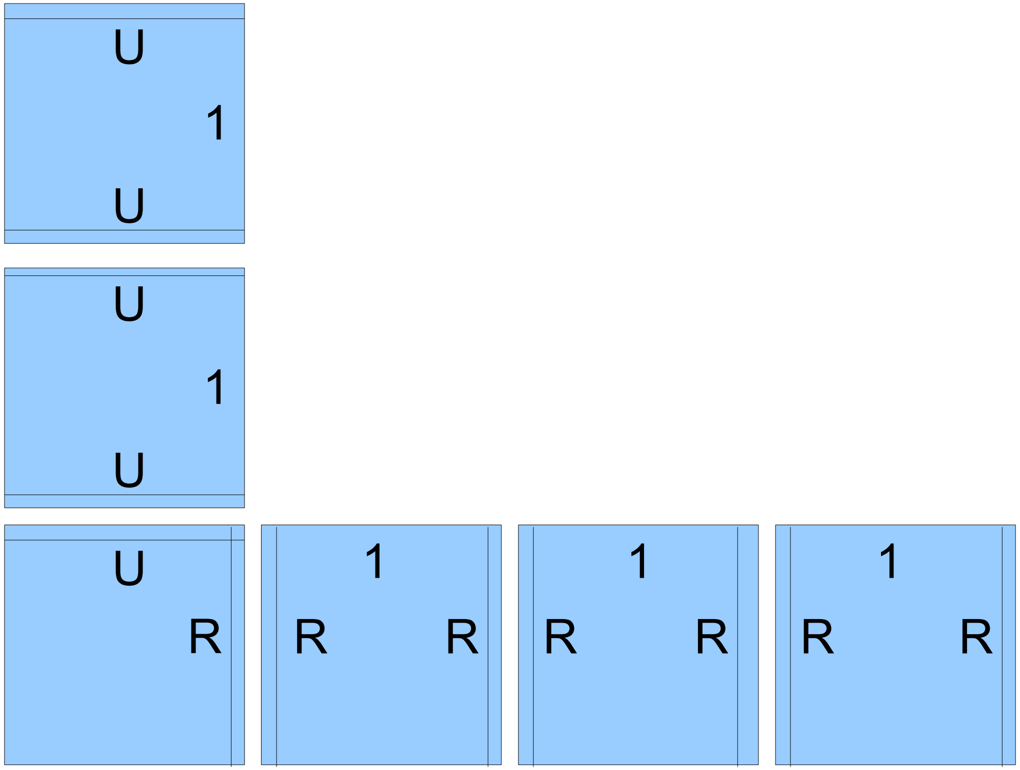
Programmable Self-Assembly



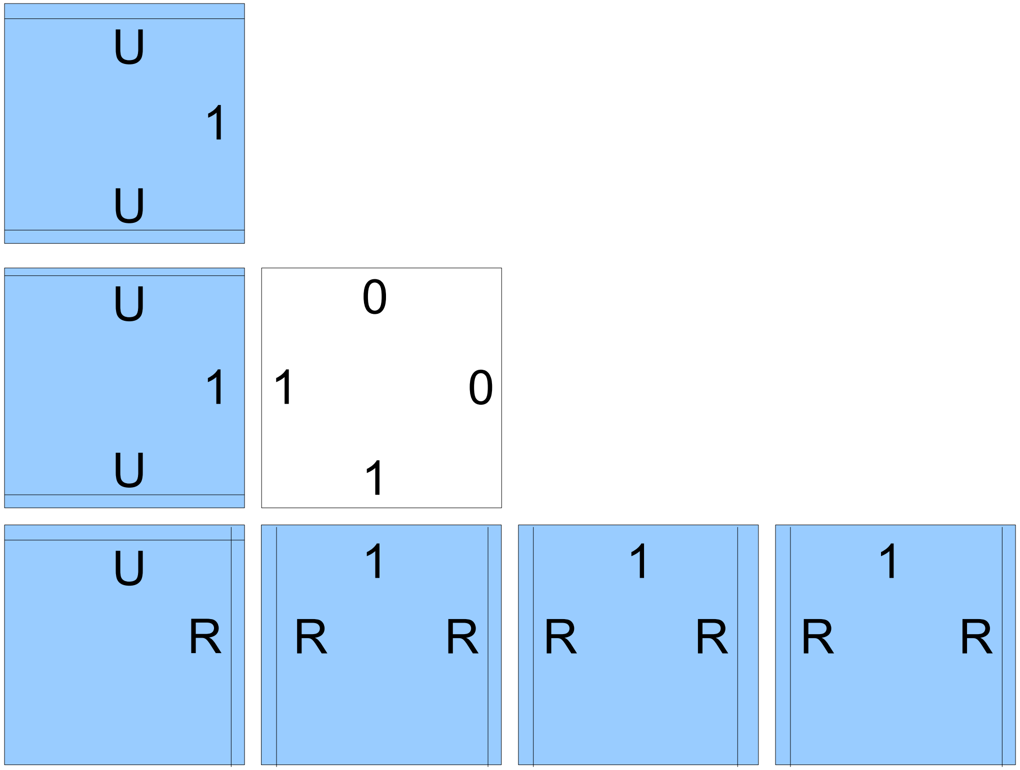
Programmable Self-Assembly



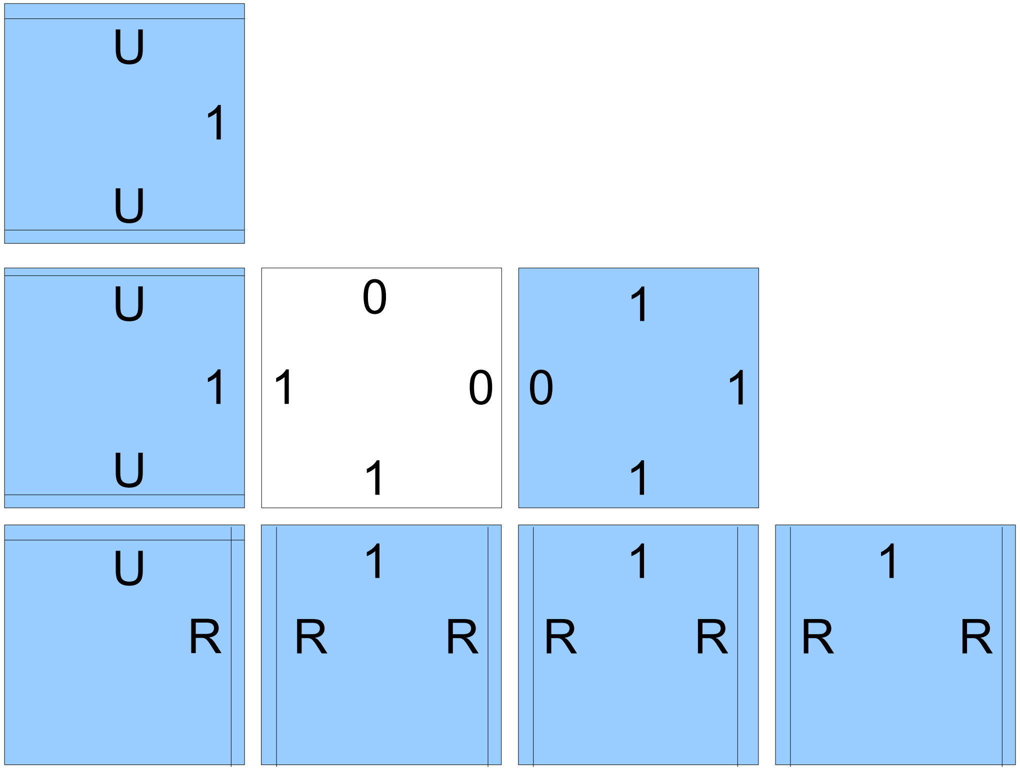
Programmable Self-Assembly



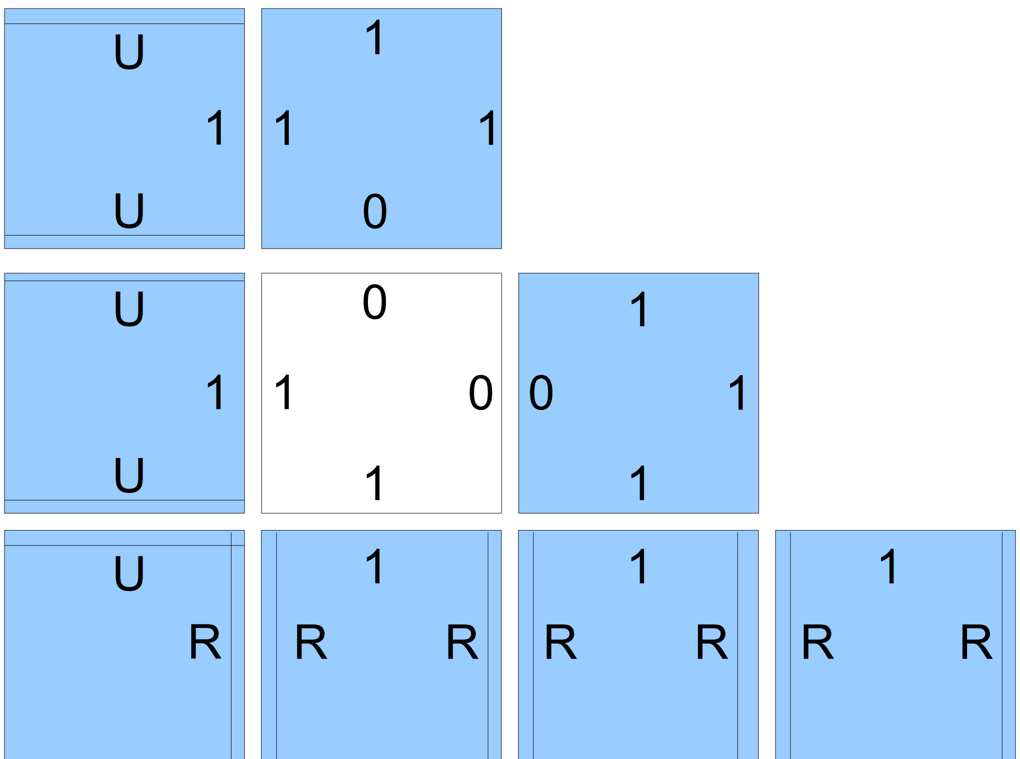
Programmable Self-Assembly



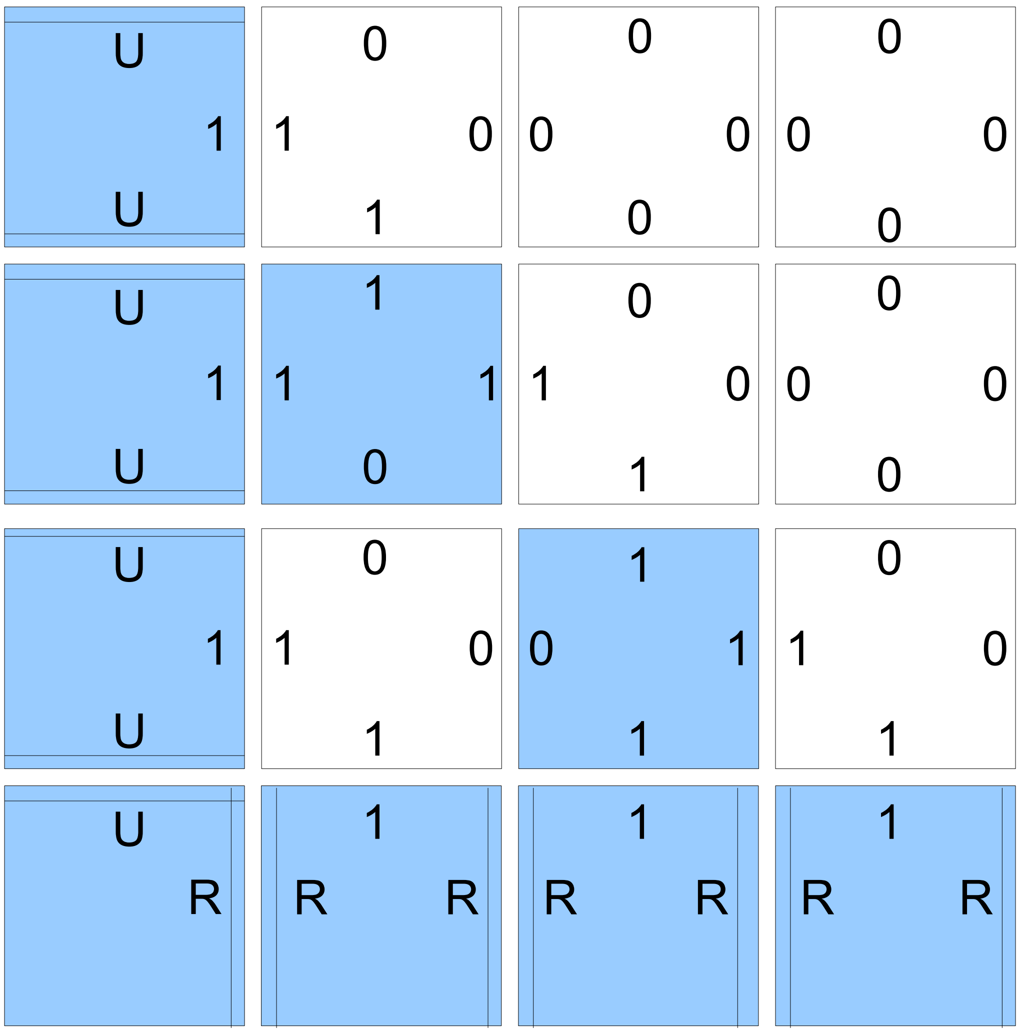
Programmable Self-Assembly



Programmable Self-Assembly

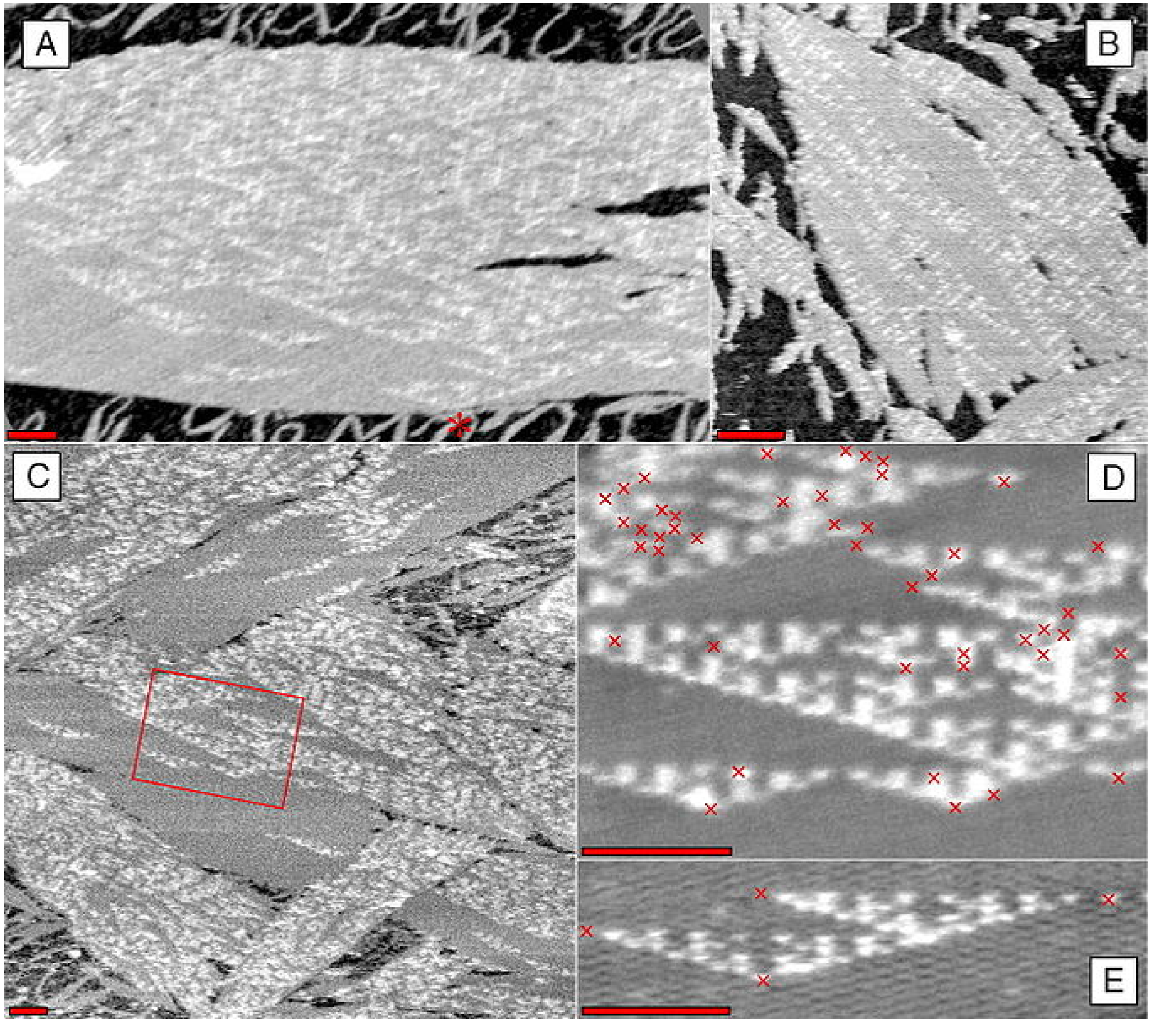


Programmable Self-Assembly



This set of eight tiles computes exclusive-or (addition mod 2), and is colored only if the output of the function is 1.

Programmable Self-Assembly



Rothemund et al., 2004

Tile Assembly Models

- Abstract Tile Assembly Model (aTAM) [Rothemund and Winfree]
  - Nondeterministic and error-free
  - At each time step, one tile is placed nondeterministically at the frontier
- Kinetic Tile Assembly Model (kTAM) [Winfree]
  - Probabilistic and error-permitting
  - At each time step, tiles on the frontier can bind or dissociate, with probabilities based on rate equations from chemical kinetics

Fundamental Questions

- Necessary and sufficient conditions to produce a unique terminal assembly
- Fault tolerance / error correction

Fundamental Questions

- Necessary and sufficient conditions to produce a unique terminal assembly
  - Local determinism [Soloveichik and Winfree]
- Fault tolerance / error correction
  - Proofreading [ChenGoel], [Soloveichik *et al*.]
  - Protected Tile Mechanism [Fujibiyashi *et al*.]

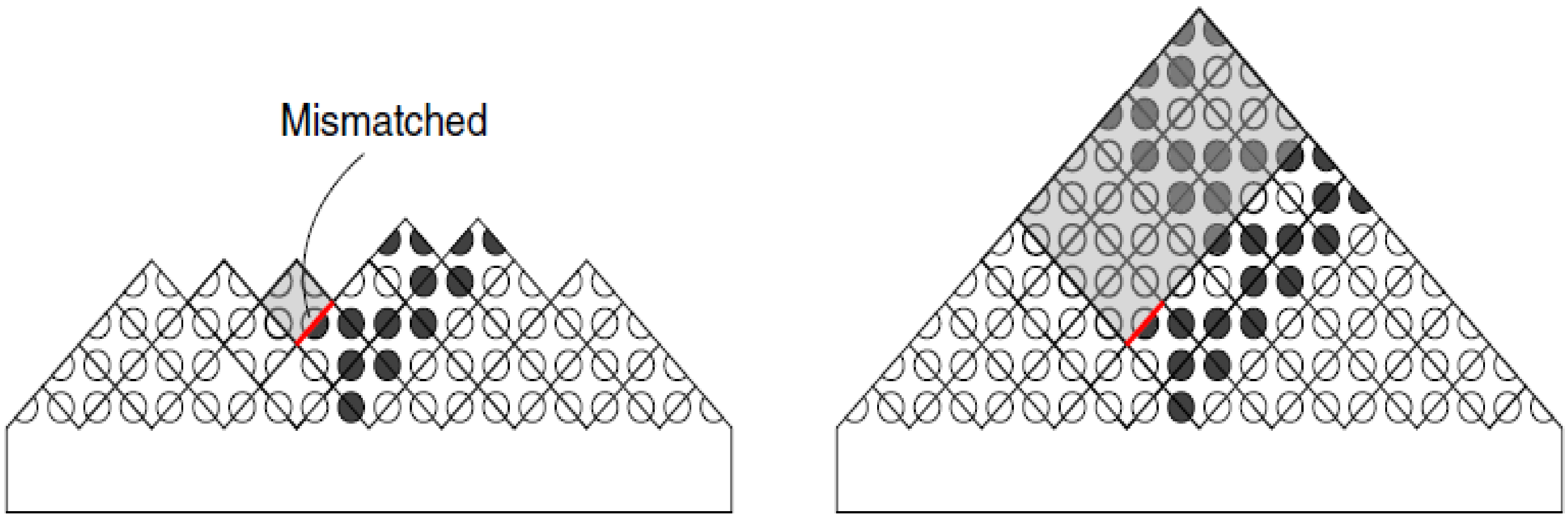
Tile Assembly as Distributed System

- Each agent has only local knowledge
- Behavior is asynchronous
- Goal is to build a global structure using only local rules
- Researchers in distributed systems have been designing algorithms for fault-tolerance for thirty years.

Tile Assembly as Distributed System

- Each agent has only local knowledge
- Behavior is asynchronous
- Goal is to build a global structure using only local rules
- Researchers in distributed systems have been designing algorithms for fault-tolerance for thirty years.
- Binding errors in self-assembly are fundamentally different from faults in previously-studied distributed systems.

Tile Binding Errors



A mismatched tile is trapped in the assembly before it can dissociate. [Fujibayashi *et al*.]

Binding Errors as Inconsistent Registers

- Metaphor: an agent approaches the assembly, “asks” whether the bonds in a location are correct, “hears” incorrectly what the bonds are, and binds at that location.

Binding Errors as Inconsistent Registers

- Metaphor: an agent approaches the assembly, “asks” whether the bonds in a location are correct, “hears” incorrectly what the bonds are, and binds at that location.
- Mathematics: simulate tile assembly systems with systems of distributed processors. The registers of these processors can be faulty, *i.e.*, can return inconsistent values, to model binding errors

Memory Consistency Conditions

- Multiprocessor programming, and architecture theory, have dealt with this type of problem for years.
- Just because a processor “writes” to a register, the register may not return that value. For example, the value may be in a cache, to be written to the register later.
- Programmers want guarantees of consistency; designers of architecture and compilers want flexibility for optimization.

Memory Consistency Conditions

- A memory consistency model specifies the allowable behavior of memory.

Memory Consistency Conditions

- A memory consistency model specifies the allowable behavior of memory.
- *Sequentially consistent*: operations of all processors executed in sequential order, and ops of each processor appear in the order specified by its program.

Memory Consistency Conditions

- A memory consistency model specifies the allowable behavior of memory.
- *Sequentially consistent*: operations of all processors executed in sequential order, and ops of each processor appear in the order specified by its program.
- *Causally consistent*: for each processor, the ops of that processor plus all writes known to that processor appear in a total order that respects potential causality.

Examples

Causally consistent, not sequentially consistent:

p_1: $WRITE_x(0)$ $READ_x(1)$

p_2: $WRITE_x(1)$ $READ_x(0)$

Examples

Causally consistent, not sequentially consistent:

p_1: $WRITE_x(0)$ $READ_x(1)$

p_2: $WRITE_x(1)$ $READ_x(0)$

Not causally consistent:

p_1: $WRITE_x(0)$ $WRITE_x(1)$

p_2: $READ_x(1)$ $WRITE_y(2)$

p_3: $READ_y(2)$ $READ_x(0)$

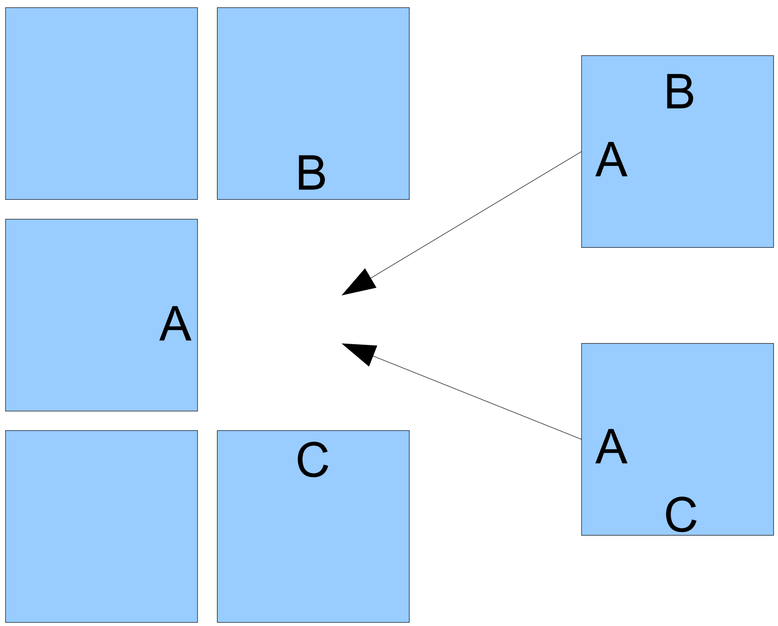
Simulation Theorem #1

- *Theorem*: There exists a class of causally consistent distributed processors that simulates the aTAM.
- “Simulate” means that each processor acts like a location on the assembly surface, and takes on a different state for each possible tile type, or “EMPTY” to simulate the absence of a tile.

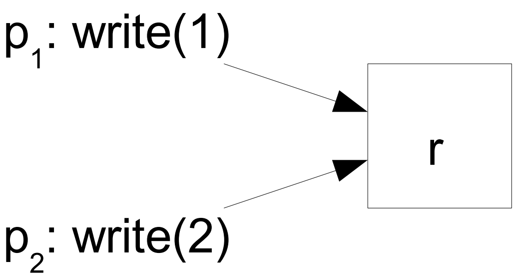
Application #1

- A tile assembly system is *locally deterministic* if, for any location in the assembly, a unique tile type can be placed legally in that assembly, given the neighboring tiles that were placed there previously in the assembly sequence.
- A multiprocessor program is *concurrent-write free* if no legal program execution permits conflicting writes to the same register.

Application #1



Not locally deterministic



Not concurrent-write free

Application #1

- *Theorem*: *T* is a locally deterministic tile assembly system iff it can be simulated by a concurrent-write free program on a system of distributed processors whose behavior is entirely determined by local binding rules.

Application #1

- *Theorem*: Local determinism iff simulation is concurrent-write free.
- *Consequence*: Programming language techniques (like types) to ensure concurrent-write freedom will also enforce local determinism when compiling tile assembly systems.

Application #1

- *Theorem*: Local determinism iff simulation is concurrent-write free.
- *Consequence*: Programming language techniques (like types) to ensure concurrent-write freedom will also enforce local determinism when compiling tile assembly systems.
- *Consequence*: Heuristics to check failure of concurrent-write freedom can be applied to self-assembly programming.

Simulation Theorem #2

- GWO ("global write-read-write order") is the condition that there is global agreement on the order of any two writes, when a processor can prove it has read one before the other.
- *Theorem*: There exists a class of GWO-consistent distributed processors that simulates the kTAM.

Simulation Theorem #2

- *Theorem*: There exists a class of GWO-consistent distributed processors that simulates the kTAM.

Simulation Theorem #2

- *Theorem*: There exists a class of GWO-consistent distributed processors that simulates the kTAM.
- *Intuition*: In the kTAM, future bonds are causally related to past bonds. So writes are globally causally related, even though there is no guarantee that registers will return the values written.

Simulation Theorem #2

- *Theorem*: There exists a class of GWO-consistent distributed processors that simulates the kTAM.
- *Intuition*: In the kTAM, future bonds are causally related to past bonds. So writes are globally causally related, even though there is no guarantee that registers will return the values written.
- *Note*: This is the first “natural” distributed system shown to obey GWO, and not anything stronger. Errors in, *e.g.*, sensor networks or silicon architecture, are fundamentally different.

Application #2

- *Self-stabilizing system*: starting from any initial state, it is guaranteed to converge to a "legitimate" state.

Application #2

- *Self-stabilizing system*: starting from any initial state, it is guaranteed to converge to a "legitimate" state.
- *Theorem*: There exists a polynomial-time algorithm that, given a locally deterministic *T* for the kTAM, outputs a self-healing, proofreading tile assembly system *T'*.

Application #2

- *Self-stabilizing system*: starting from any initial state, it is guaranteed to converge to a "legitimate" state.
- *Theorem*: There exists a polynomial-time algorithm that, given a locally deterministic T for the kTAM, outputs a self-healing, proofreading tile assembly system T'.
- *Note*: This was already known, though not published in this general form. The new contribution is the proof technique of self-stabilization.

Future Work

- “The greatest promise [of algorithmic DNA self-assembly] may lie in applications where DNA nanostructure templates have been used to assemble other inorganic components and functional groups.” [Nanofabrication by DNA self-assembly, Li *et al*.]

Future Work

- “The greatest promise [of algorithmic DNA self-assembly] may lie in applications where DNA nanostructure templates have been used to assemble other inorganic components and functional groups.” [Nanofabrication by DNA self-assembly, Li *et al*.]
  - Models of “mixed media” self-assembly

Future Work

"For the forseeable future, self-assembly has to deal with a significantly higher defect rate than etching and similar methods; this presumably has to be dealt with at the algorithmic level. Thus we need a theory of fault-tolerant assembly, as well as new fault-tolerant algorithms and architectures for these models." [The Computational Worldview and the Sciences, Arora *et al*.]

Future Work

"For the forseeable future, self-assembly has to deal with a significantly higher defect rate than etching and similar methods; this presumably has to be dealt with at the algorithmic level. Thus we need a theory of fault-tolerant assembly, as well as new fault-tolerant algorithms and architectures for these models." [The Computational Worldview and the Sciences, Arora *et al*.]

- Self-stabilizing algorithms for self-assembling agents with binding errors

Thank you!