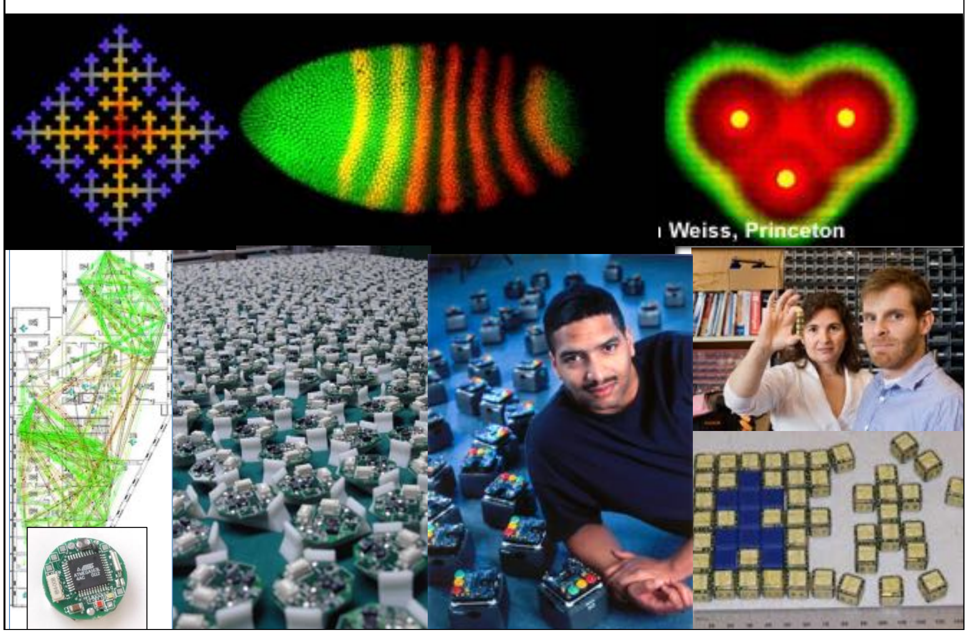
Global-to-Local Theory
CS289

"Spatial" Computers

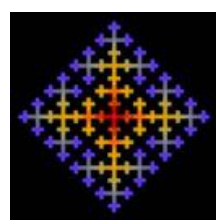


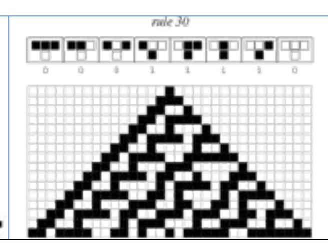
Why Global-to-local Theory?

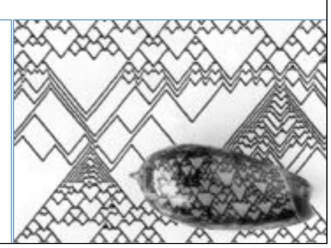
- Global-to-local compilers allows us to transform a class of global goals into local rules for individual agents
  - Robustness, scalability, provable..
- But they do not tell us what is *computable*
  - Local to Global is hard: e.g. Conway's Game of Life
  - But, Global to Local is possible: *Yamins, PhD 2008*

Cellular Automata

- Stanislaw Ulam and John von Neumann (1940s)
  - Simulate "discrete" biology & physics;
  - Self-replicating machines
- Conway's Game of Life (1970s)
  - A simple intuitive rule….amazing dynamic patterns!
  - Turing Complete! (2002)
- Wolfram, A New Kind of Science, 2002
  - Systematic classification of *all* 1D two-state CA rules





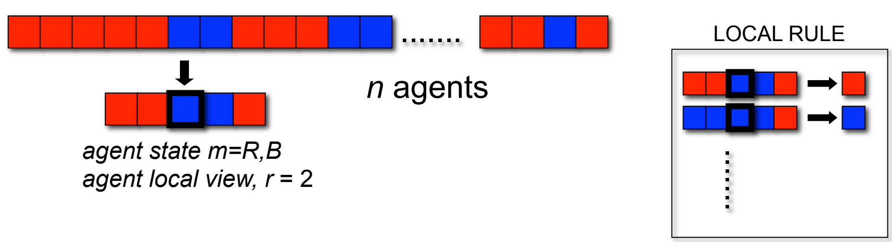


Why Global-to-local Theory?

- Global-to-local compilers allows us to transform a class of global goals into local rules for individual agents
  - Robustness, scalability, provable..
- But they do not tell us what is *computable*
  - Local to Global is hard: e.g. Conway's Game of Life
  - But, Global to Local is possible: *Yamins, PhD 2008*

The Setup

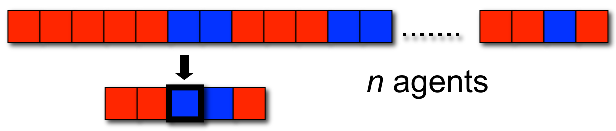
- 1D multi-agent system (like cellular automata)



-

The Setup

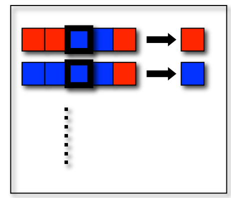
- 1D multi-agent system (like cellular automata)



n agents

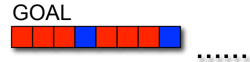
agent state m=R,B
agent local view, r = 2

LOCAL RULE

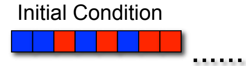


- **Goal:**
  - Self-organize target pattern
  - *Scalable to more agents*
  - *Any initial condition*
  - *Any timing model*

GOAL



Initial Condition



Theoretical Underpinnings

- **Local Checkability**
  - Given agent model (r, m)
  - Can you design a "voting" scheme such that if every agent says 1, then the global pattern is in goal space.
  - *Necessary and Sufficient**
- **If no check exists => no solution exists**
  - Can use to to prove minimal requirements
- **If a check is available =>**
  - Can automatically produce a local rule,
    but with slightly larger radius (R=2r+2)
  - Provably correct, robust to asynchrony, self-repairing

Lets do an example

- Goal Pattern: 000100010001.....0001

1) Design and prove correct a **local check scheme for r=2**
2) Prove that **no local check** scheme can be designed for r=1
3) How would you add state (change pattern) to make r=1 possible?
4) Make a **local rule of radius r=4** for the original pattern

5) Prove there **is no local check of finite radius** for the half-n-half pattern (0^n1^n) pattern

Goal Pattern: 000100010001.....

Local check scheme for r=2

Left case: 000 0001

Right case: 001 and 0001

Middle case: 00100 00010 01000 10001

No local check for r=1: You need to accept 000, but then all zeros would be accepted

Local Rule Construction for r=4

Always possible to make a local rule of length R=2*r + 2

Method is to make a "left-side" local rule (here, we do r=4 on left side)

Special cases on left side

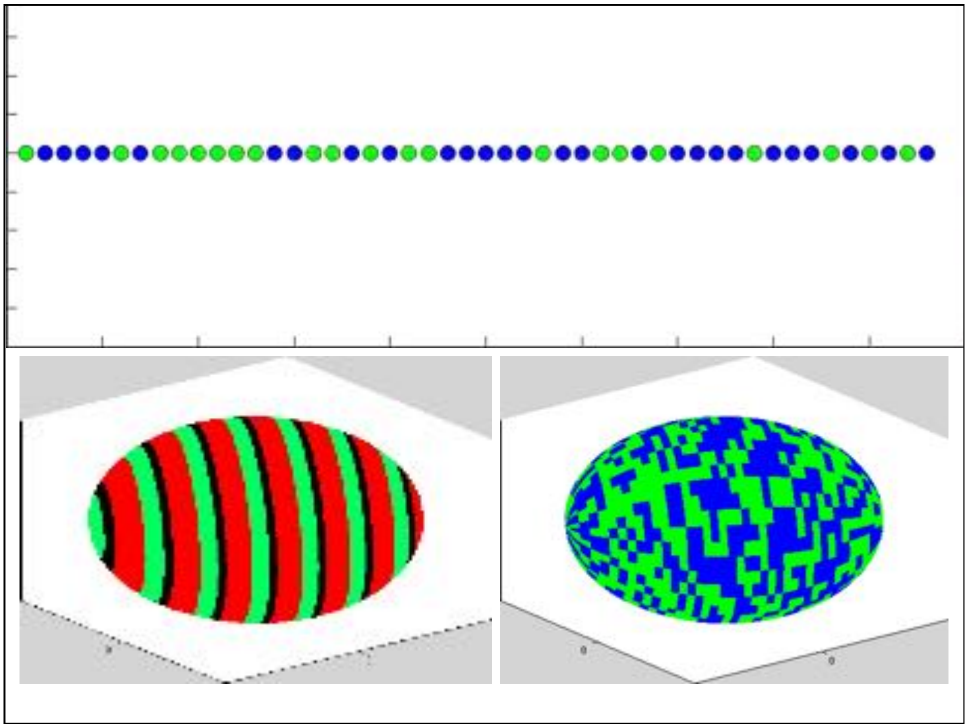
- * => 0
- 0* => 00
- 00* => 000
- 000* => 0001

General cases

- 0001* => 00010
- 0010* => 00100
- 0100* => 01000
- 1000* => 10001

In this case,
You cannot do a left-side rule for r=2
Because 00* is ambiguous
00* could be 001 or 000

Example, try this initial condition: 1000 0010 0000 00000……



Yamin's Global-to-Local Compiler

Local Checkability-based Compiler

Input:

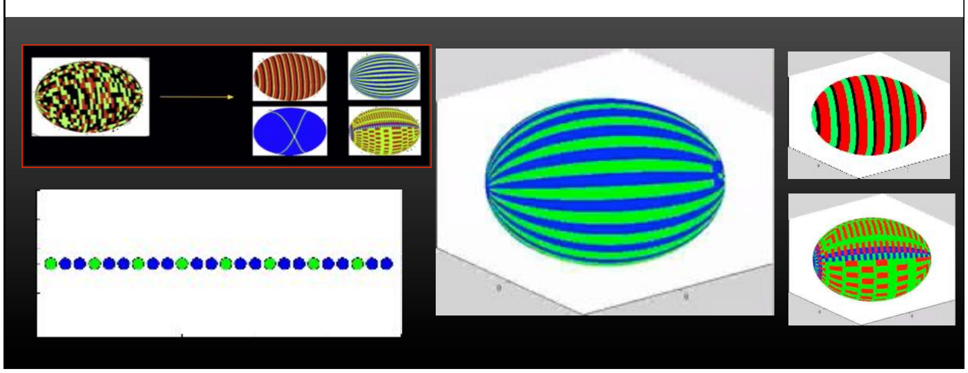
Target pattern, desired agent state and radius

Output:

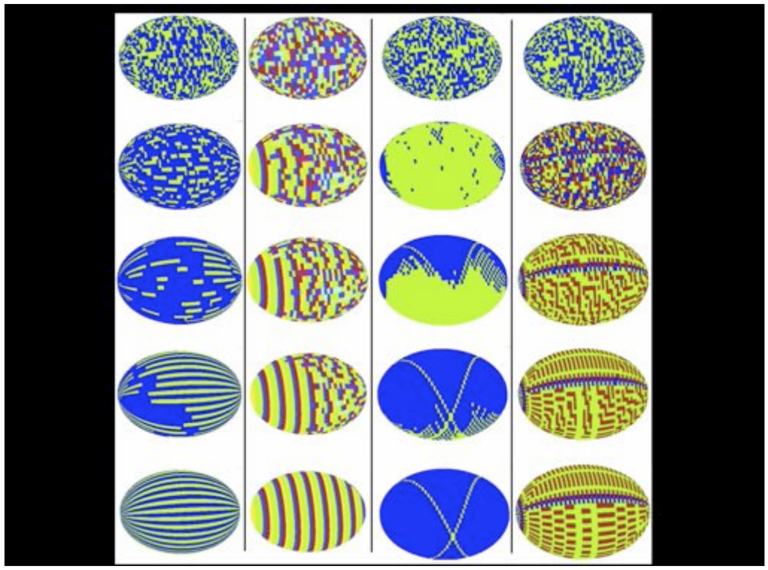
Compiler tries to derive a local check scheme

Either doesn't exist (suggest minimum radius needed)

Or Local Rule (scalable, asynchronous, self-repairing)



Compiler generated patterns

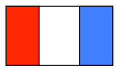


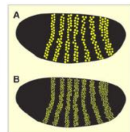
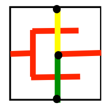
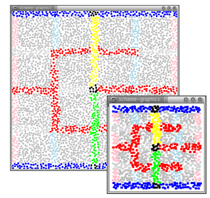
Some Thoughts

- So far we have tackled 1D systems. Can we generalize the ideas to other agent models?
- Open Questions:
  - More complex patterns
    - E.g. Majority vote (Melanie Mitchell)
  - More complex spaces
    - 3D cellular automata*: Lattice Swarms! (Th&B)*
  - Approximate (high-probability) solutions

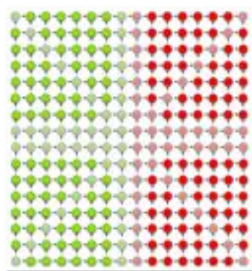
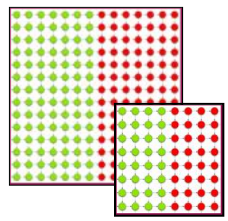
The Curious Case of 2D Proportional Patterns

Proportional Patterns are Interesting



In 1D (line), no solution exists with fixed state and radius
But, in 2D (square), can solve with finite state and radius!

Theoretical Underpinnings

- Reason *theoretically* about *intuitive* things
  - How one can tradeoff state and radius
  - Why some things are harder than others
  - Why some things take longer than others
  - How simple patterns can be combined to make complex ones
  - Why 1D patterns are like Strings (relation to grammars)
  - Why global-to-local is possible in CAs, whereas local-to-global may be so complex....