Automated Global-to-Local Programming in 1-D Spatial Multi-Agent Systems

Daniel Yamins and Radhika Nagpal
Harvard University
School of Engineering and Applied Sciences
Cambridge, MA 02138
yamins@fas.harvard.edu, rad@eecs.harvard.edu

ABSTRACT

A *spatial computer* is a distributed multi-agent system that is embedded in a geometric space. A key challenge is engineering local agent interaction rules that enable spatial computers to robustly achieve global computational tasks. This paper develops a principled approach to global-to-local programming, for pattern formation problems in a one-dimensional multi-agent model. We present theoretical analysis that addresses the existence, construction, and resource tradeoffs of robust local rule solutions to global patterns, and which together form a "global-to-local compiler".

1. SPATIAL COMPUTERS

A *spatial computer* is a distributed multi-agent system that is embedded in a geometric space. The agents' computational constraints are local: each agent has limited internal memory and processing power, and communicates only with neighboring agents. The systems' computational goals, however, are typically defined relative to the global spatial structure.

Real-world spatial computers abound. In biology, embryonic development is a prototypical example, where large numbers of identically-programmed agents interact to generate a spatially complex organism from an undifferentiated embryo [14] (fig. 1). In engineering, sensor networks distributed over large open regions or embedded in buildings react to dynamic global environments by collectively processing spatially localized data [12]. Mobile and reconfigurable robot swarms cooperate to achieve global spatial search, transport, and shape formation imperatives from innumerable local actions [8, 6, 10]. A "paintable computer" that could form image displays and storage devices on-the-fly would be an ultimate, if not yet accessible, form of spatial computing [2].

A key challenge for spatial computing is *programmable self-organization*. How does one design local interaction rules to achieve prespecified global goals? Despite the many compelling examples in nature, decentralized systems are difficult to reason about. Cascades of interaction produced by iterated local rules often behave in apparently unpredictable ways, and therefore are hard to design. This difficulty is compounded by the need for *robustness*, so that communications failures and perturbations that disturb equilibria are automatically repaired. Furthermore, some global tasks may not be solvable at all, given the local agents' computational and informational limitations.

Cite as: Automated Global-to-Local Programming in 1-D Spatial Multi-Agent Systems, D. Yamins and R. Nagpal, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. XXX-XXX.

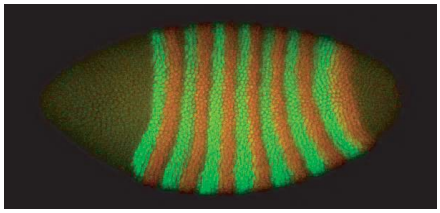


Figure 1: Embryo of the common fruit fly *Drosophila melanogaster* several hours post-fertilization. The colored stripes are a repeating pattern of spatially organized gene expression, roughly radially symmetric around the embryo's major axis. The stripes, which emerge from interactions between cells as the embryo grows, are developmental precursors of the major body components of the adult fly.

This paper addresses the problem of designing spatial computer programs in a principled way, for the case of pattern formation problems in a one-dimensional multi-agent model. In doing so, we resolve four subproblems:

1) An **existence** problem: We demonstrate a simple criterion, called *local checkability*, that any robustly solvable pattern must satisfy. If a pattern is not locally checkable, no robust local rule can self-organize it.

2) A **construction** problem: For all patterns that *are* locally checkable, we describe an algorithmically-derived local rule that generates it robustly. The mechanism this procedure exploits can be viewed as a *self-organizing Turing machine*.

3) A **resource** problem: The two main resource parameters of the model, the agent interaction radius and agent memory size, exist in a *radius-state resource tradeoff*. We describe algorithms for tuning along the continuum between large-radius/low-state and low-radius /high-state implementations. And,

4) A **description** problem: We exhibit a simple method, called *local feature invariance*, to allow programmers to describe solvable global patterns compactly.

By combining these four techniques, we are able to build a "Glocal-to-Local Compiler" – a procedure which takes as input a pattern and resource parameter limits, and whose output is a local rule generating the pattern within the specified limits.

The methods presented in this paper only explicitly address one-dimensional pattern formation (applying, for instance, to the case of fruit fly early embryonic development shown in fig. 1). However, the techniques naturally generalize to other spaces, as we discuss at the end.

Related Work

Our 1-D multi-agent model is similar to cellular automata. Studies of cellular automata go back over 50 years, from the

pioneering work of Ulam and von Neumann to the more recent attempts to classify automata rules, and their use in modeling real biological systems [11, 13]. Single rules, like Conway's Game of Life, have been shown to generate a variety of complex behaviors from varying initial conditions [3]. The main emphasis of these studies has been to understand the local-to-global connection: given a local agent rule, what patterns will it produce?

In contrast, engineering emphasizes the inverse global-to-local question: given a pattern, which agent rules will robustly produce it? Recent work has shown that this problem can be tractable. Nagpal and others have created pattern formation languages for Amorphous computers, while Klavins, Kotay, and others have demonstrated robust self-assembly algorithms for modular robots [7, 5, 10]. This paper examines similar problems in a simpler model, but achieves more comprehensive results. We introduce a set of generic techniques – local checkability, self-organized Turing machines, resource tradeoff algorithms, local feature invariance – that aim to provide a theoretical underpinning for the future algorithm development.

2. THE 1-D MODEL

This section introduces a simple mathematical framework for describing one-dimensional distributed multi-agent systems and the patterns they can form. The main objects in our model are:

- *Configurations*, static snapshots of the overall multi-agent system, composed of agents, their internal states, and the underlying 1-D geometry.
- *Local Rules*, the locally-defined identical programs each agent runs.
- *The Timing Model*, the order in which agents update their state.
- *Patterns*, the spatially-organized structures that the agents are attempting to form.
- *Robust Solutions*, iterated local rules that self-organize given patterns, robust to initial conditions and timing. Finding robust solutions algorithmically is the central goal of this paper.

Configurations: A configuration is a labeled graph whose nodes represent agents, whose labels represent agent internal states, and whose edges represent the spatial relationships between the agents. Schematically:

in which the color-number correspondence represents agent internal states. To formalize this picture, we define a *configuration of size n over state set S* as a labelled graph

$$X = (V, E, A)$$

in which the vertex set $V = \{1, \ldots, n\}$ represents the agent nodes, the edge set $E = \{(1,2), (2,3), \ldots, (n-1, n)\}$ represents the linear 1-D spatial relationships, and $A : V \to S$, is a labeling assigning each node a state in S. The labeling of the nodes is the idea that each agent is one of the states in S at any given time. The fact that the edges of the graph are directed indicates that agents can distinguish left from right.

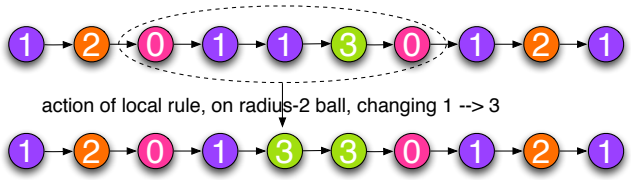
Local Neighborhoods: Given a configuration X and an agent $i \in V(X)$, we wish to define the "local view" around a in X, as a function of a choice of a *radius* parameter r. Formally, the *r-neighborhood in X at agent i*, denoted $B_r(i, X)$, is the subgraph of X whose nodes are at most distance r from i. The r-neighborhood of a "central agent" further than r hops from the ends of a configuration contains $2r + 1$ nodes, with a unique "center agent". The neighborhood of an agent closer than r hops to the left or right end contains fewer nodes and will have asymmetric local views, with fewer than $2r + 1$ nodes. Let $\mathcal{B}_{r,S}$ denote the set of all local radius-r neighborhoods. (Usually we'll drop the S from the subscript when it's obvious from context.)

Local Rules: Dynamics are generated by agent-based programs running identically on each of the agents, and drawing information only from other nearby agents. Formally, a *local rule of information radius r* is any function

$$F : \mathcal{B}_r \to S.$$

A local rule is just a look-up table by which an agent i takes local information from the r-neighborhood $B_r(i, X)$ around i, and chooses a new state to adopt as a function of what it sees. The inputs to the look up are local r-neighborhoods, so therefore the domain is $\mathcal{B}_r$. The outputs are the new states that the agents will adopt, so therefore the range is S. Any two central agents with the same r-local agent state sequence will have identical r-neighborhoods, and so be treated identically by F. End-agents can act differently, since their neighborhoods have different local geometry.

A local rule F acts on a configuration $X = (V, E, A)$ at agent $i \in X$ by changing the value of $A(i)$ from whatever it is to $F(B_{r(F)}(i, X))$. For example, suppose F is a radius-2 local rule such that on the neighborhood $b = (01\widehat{1}30)$, $F(b) = 3$. (The hat indicates the agent at the center of b.) Then the action of F on agent 5 from the configuration depicted above is:



The Timing Model: Given a configuration of agents and a local rule, there are many orders in which agents can act. An *asynchronous* action updates one agent at a time. A local rule can also act (partially or wholly) synchronously by having a *subset* of agents all update their states simulateously. For a configuration X, given $c \subset V(X)$, define $F(c, X)$ to be the configuration obtained by replacing the labels of each $i \in c$ with $F(B_{r(f)}(i, X))$. We define a *call sequence for a size-n configuration* to be an infinite sequence

$$c = (c_1, c_2, c_3, \ldots)$$

where each $c_j \subset V(X)$. The set c_j represents the agents that are called at timestep j.

In this paper, we work over a "live timing model" which includes both synchronous and asynchronous call sequences. A call sequence c is *live* if it calls every agent infinitely many times, meaning intuitively that no agent is completely excluded. Let $\mathcal{S}_n$ denote the set of all live call sequences for size-n configurations, and $\mathcal{S} = \cup_n \mathcal{S}_n$.

Given a local rule F, an initial configuration X_0, and a call sequence c, a trajectory arises by each agent iteratively applying F in the order given by c. Formally, the *trajectory of local rule F starting at X_0 generated by call sequence c* is the sequence $\{F_c^n(X_0) | n \in \mathbb{N}\}$, where

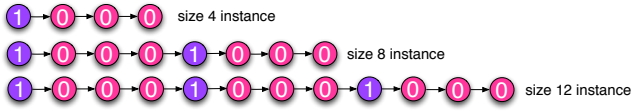
$$F_c^n(X_0) = F(c_n, F_c^{n-1}(X_0))$$

and $F_c^0(X_0) = X_0$. If the trajectory converges to a unique final fixed configuration y, then we say that the trajectory

has a *well-defined limit*, and write $y = \lim_n F_c^n(X)$.

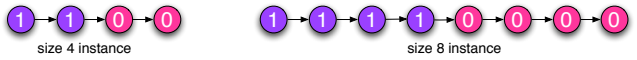
Pattern Goals: The "computational goal" of the spatial computers studied here is pattern formation. Intuitively, a pattern is an arrangement of states that is ordered in some fashion. This order can be described most simply by defining patterns as sets of configurations. Formally, let $\mathbb{C}_S$ be the set of all configurations with state set S. Then a *pattern* over state set S is any subset $T \subset \mathbb{C}_S$. The elements of T are the *instances* of the pattern.

Two broad classes of patterns that are of interest are *repeat patterns* and *proportionate patterns*. Consider a length-4 configuration of agents, the left-most of which has state 1, followed by three agents in state 0. By repeatedly concatening this segment with itself we obtain a version of the 1000 pattern at every size that is a multiple of 4:



Given any finite state sequence q, the general repeat pattern is $T_q = \{q^n | n \in \mathbb{N}\}$ for any finite m-ary string q. The pattern T_q only has instances at sizes that are multiples of the size of the generating unit. The fruit fly embryonic gene pattern shown in fig. 1 is a repeat pattern.

In contrast, consider the "half pattern" $T_{1/2}$ consisting of configurations of the form 1^n0^n, for each n:



$T_{1/2}$ is an example of a proportionate patterns, characterized by having a subpattern (or a change in background pattern) appear at a specific fractional value along an otherwise (piecewise) uniform structure.

Robust Solutions: Intuitively, to "robustly solve" a pattern, a local rule must generate trajectories that always converge to configurations consistent with that pattern. Formally, a local rule F is a *robust solution to pattern* T if for all initial conditions X_0 and live call sequenes $c \in \mathbb{C}_{|X_0|}$, the limit of the trajectory generated by F starting at X under c is well-defined and an element of T whenever T contains at least one configuration of size n. Symbolically,

$$\lim_{n\to\infty} F_s^n(X) \in T \text{ whenever } T \cap \mathbb{C}_n \neq \emptyset.$$

We impose the condition that $T \cap \mathbb{C}_n \neq \emptyset$ because it would be unfair to expect a rule that cannot add or remove agents to push into T a configuration whose size is wrong.

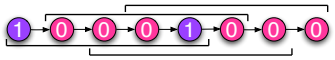
3. LOCAL CHECKABILITY: EXISTENCE

In this section, we describe a simple "necessary condition" that any pattern must meet to be robustly solvable.

Let T be a pattern. Assuming that T has at least one instance X of size n, the definition of a local rule F being a solution to pattern T requires that

$$\lim_{n\to\infty} F_c^n(X) = Y$$

for some fixed configuration $Y \in T$. The local neighborhoods around the agents in Y must therefore all be fixed points of F. These local neighborhoods overlap, forming a mutually interlocking configuration-wide stop state. For example, if a local rule F with radius 2 is a solution to the T_{1000} pattern, then in this figure:



all of the bracketed 2-neighborhoods must be fixed states of F. That is, $F(10\widehat{00}01)$, $F(00\widehat{00}01)$, $F(01\widehat{00}0) = 0$ and $F(00\widehat{1}00) = 1$.

These considerations suggest that for T to admit a robust solution, it must be possible to find a coherent set of locally-specifiable stop states as a subset of T. This constrains the patterns T that are robustly solvable. To see this formally, we define the notion of *local checkability*.

Recall the definition of $\mathcal{B}_r$, the set of local neighborhoods of radius r with states in the set S. Consider a binary-valued function Θ on $\mathcal{B}_r$, i.e.

$$\Theta : \mathcal{B}_r \to \{0, 1\}.$$

Θ should be thought of as a "recognition function" – $\Theta(b) = 1$ means that the local neighborhood b is "recognized" as a correct local stop state. For any such function define

$$\Theta(X) = \prod_{a \in V(X)} \Theta(B_r(a, X)).$$

When applied to a whole configuration X, $\Theta(X) = 1$ only when *all* agents recognize a stop state.

Definition 1 ***[Local Check Schemes]*** *Let T be a pattern. A binary-valued function Θ is a* **local check scheme (LCS) for T of radius** *r if*

- *For all $X \in \mathbb{C}_S$, $\Theta(X) = 1 \;\Rightarrow\; X \in T$.*
- *For all n such that $T \cap \mathbb{C}_{n,S} \neq \emptyset$, there is $X \in \mathbb{C}_{n,S}$ such that $\Theta(X) = 1$, where $\mathbb{C}_{n,S}$ are all the configuration over S of size n.*

The smallest r for which there exists a check scheme of radius r for T is the **local check radius of** *T, denoted $LCR(T)$. If there is no check scheme for T of any finite radius with m states, then $LCR(T)$ is defined to be ∞.*

Intuitively, the first condition in the above definition prevents bad deadlocks, while the second condition requires there to be at least one fixed point. The key result is:

Proposition 1 *If F is a robust solution to T, then $r(F) \geq LCR(T)$.*

Proof. Suppose F is a solution to T. Then if $F(c, X) = X$ for all call sequences c only if $X \in T$. Moreover, for each n, choosing any $X \in \mathbb{C}_{n,S}$, let $Y = \lim_n F_s^n(X)$. Then $Y \in \mathbb{C}_{n,S}$ and $F(c, Y) = Y$ for all c, so $Y \in T$. Now define

$$\Theta_F : \mathcal{B}_{r(F)} \to \{0, 1\}$$

by

$$\Theta(B_{r(F)}(i, X)) = 1 \;\Leftrightarrow\; \forall\, i,\; F(B_{r(F)}(i, X)) = X(i).$$

Notice that $X = F(c, X)$ for all c if and only if for all $i \in V(X)$,

$$F(B_{r(F)}(i, X)) = X(i),$$

which holds if and only if

$$\prod_{i \in V(X)} \Theta(B_{r(F)}(i, X)) = 1.$$

Hence, $\Theta(X) = 1$ implies $X \in T$, and for each n there is a $Y \in \mathbb{C}_{n,S}$ such that $\Theta(Y) = 1$. Thus Θ is a local check scheme for T, and $LCR(T) \leq r(F)$. □

In words, **local checkabilty is necessary for solvability**, simply describing the fact that the stopping condition of "satisfying the pattern" must be locally recognizable. Another way to say this is: if we let T_Θ be the pattern consisting of all configurations X such that $\Theta(X) = 1$, the pattern T is locally solvable only if there is a local check scheme Θ such that $T_\Theta \subset T$ and $T \cap \mathbb{C}_n \neq \emptyset \Rightarrow T_\Theta \cap \mathbb{C}_n \neq \emptyset$.

Example 1 The repeat pattern T_{1000} has a radius-2 local check scheme Θ given by setting $\Theta(b) = 1$ iff $b = 01\widehat{000}$, $00\widehat{100}$, $10\widehat{001}$, $00\widehat{010}$, $\widehat{1000}$, $1\widehat{000}$, $10\widehat{00}$, or $100\widehat{0}$. However, a radius of 1 with two states is insufficient to provide an LCS for the T_{1000} pattern, for suppose Θ were such an LCS. The radius-1 neighborhoods include $1\widehat{00}$, $0\widehat{00}$, $0\widehat{01}$, and $0\widehat{10}$. Any LCS for T_{1000} would thus have to accept $0\widehat{00}$, but then it would also have to accept 0 strings of any length, contradicting the first condition on an LCS. Hence, $LCR(T_{1000}) = 2$, so any solution must have radius ≥ 2.

A slight generalization of the above construction shows that all repeat patterns are locally checkable:

Proposition 2 *For all repeat patterns T_q, $LCR(T_q) \leq |q|/2$.*

However, proportionate patterns, in contrast, are not locally checkable at any radius, and are thus not robustly solvable by local rules:

Proposition 3 *$LCR(T_{1/2}) = \infty$, i.e. the half-pattern is not locally checkable.*

PROOF. Suppose Θ is a local check scheme of radius r for $T_{1/2}$. Let $X_n = 1^n0^n$ for $n > 2r + 1$. Since X_n is the only configuration in $T_{1/2}$ of size $2n$, it must be Θ-accepted and thus $\Theta(b) = 1$ for all r-neighborhoods b in X_n. On the other hand, the r-neighborhoods in the configurations 1^n0^{2r+1} are identical to those in X_n. Hence all such configurations must also be Θ-acceptable for all $n > 2r + 1$. But such configurations are not in $T_{1/2}$, contradicting that Θ is a local check scheme for $T_{1/2}$. □

While a complete characterization of local check schemes is beyond the scope of this paper, it is instructive to note that local checkability has useful closure properties. Given two locally checkable patterns T_1 and T_2 with check schemes Θ_1,Θ_2, $\Theta_1 \cdot \Theta_2$ is a check scheme for $T_1 \wedge T_2$, the logical "AND," while $\Theta_1 \cdot \Theta_2 + \Theta_1 + \Theta_1$ taken modulo 2 checks $T_1 \vee T_2$, the logical 'OR'. Hence

$$LCR(T_1 \wedge, \vee T_2) \leq \max\{LCR(T_1), LCR(T_2)\}.$$

Similarly, local checkability is closed under concatenations: define

$$T_1 \cdot T_2 = \{X \circ Y \mid X \in T_1, Y \in T_2\}.$$

Then if Θ_1 and Θ_2 are check schemes for T_1 and T_2, defining $\widetilde{\Theta}(b) = 1$ for all $r(\Theta_1) + r(\Theta_2)$-neighborhoods in $T_{\Theta_1} \cdot T_{\Theta_2}$ gives local check scheme for $T_1 \cdot T_2$. Hence,

$$LCR(T_1 \cdot T_2) \leq LCR(T_1) + LCR(T_2).$$

A similar construction holds for unordered concatenation, i.e. for the pattern $T_1 \times T_2$ defined as containing configurations

$$x_1 \circ y_1 \circ \ldots \circ x_n \circ y_n$$

with $x_i \in T_1 \cup \{\emptyset\}$ and $y_i \in T_2 \cup \{\emptyset\}$. By using the operators $\{\wedge, \vee, \cdot, \times\}$ in arbitrarily complicated combinations, a wide variety of locally checkable complex patterns can be created. For instance, given the two simple repeat patterns T_{100} and T_{1000}, we can easily form the combination patterns

$$T_{100} \vee T_{1000} = \{(100)^n, (1000)^n,\ |n \in \mathbb{N}\}$$

and

$$T_{100} \cdot T_{1000} = \{(100)^n(1000)^m,\ |n, m \in \mathbb{N}\}.$$

The closure properties and the similarly of the proof of prop. 3 to that of the pumping lemma suggests local check schemes are related to regular languages. The connection is somewhat subtle and beyond the scope of this paper.

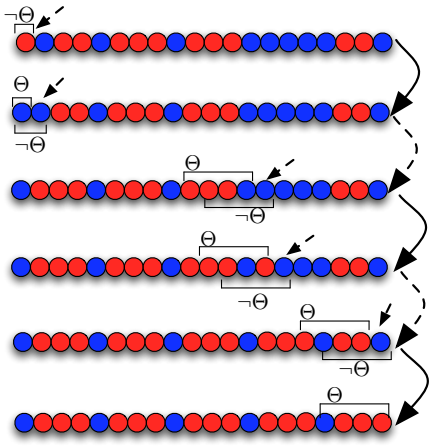


Figure 2: Single-choice algorithm on typical initial condition. Θ = correct locally; $\neg\Theta$ = not locally correct.

4. SUFFICIENCY BY CONSTRUCTION

Proposition 1 establishes that local checkability is a necessary condition for solvability. We now demonstrate, by construction, that it is also sufficient.

Specifically, given a local check scheme Θ, recall that T_Θ is the pattern consisting of all configurations X such that $\Theta(X) = 1$. We will show how for each local check scheme Θ we can construct a local rule F_Θ that is a robust solution for T_Θ. The radius of the local rule F_Θ need not be the same as the radius $r(\Theta)$ as long as it is finite, but by virtue of prop. 1, of course $r(F_\Theta) \geq r(\Theta)$. In the construction we make, the local rule F_Θ will have radius at most $2r(\Theta) + 2$.

4.1 Single-Choice Patterns: A Gradient Algorithm

We will first show how to solve a subclass of local check schemes that have an especially simple structure.

Specifically, define a subconfiguration x to be Θ-*consistent* if there is a configuration $X \in T_\Theta$ such that x is a subconfiguration of X. Then, we say that a local check scheme Θ is *single-choice* if for any Θ-consistent subconfiguration x, there is at most one $i \in S$, such that $x \circ i$ is Θ-consistent and at most one j such that $j \circ x$ is Θ-consistent (where $\circ$ denotes string concatenation). Let that unique i be denoted $\nabla_\Theta(x)^+$ when it exists. Intuitively, this is the "gradient" of Θ from the left direction. If Θ is of radius r, then $\nabla_\Theta(x)^+$ is only a function of the right-most $2r$ states.

We will now use the gradient to produce robust solutions to single-choice check shemes. Let $R = 2r$ and let B be any R-neighborhood. Let B_- denote the r-neighborhood consisting of the portion of B to the left of the center agent, and let $B(0)$ denote the state of the center agent. Now, let F_Θ be the local rule of radius R defined by

$$F_\Theta(B) = \begin{cases} \nabla^+_\Theta(B_-), & \text{if } \Theta[B_-] \\ B(0), & \text{otherwise} \end{cases}. \qquad (1)$$

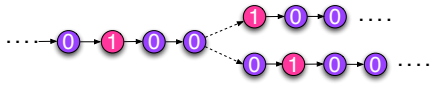
In words, the first clause causes an agent that is just to the right of a patch of Θ-correctness to switch its state to extend the patch of Θ-correctness rightward; while the second clause instructs an agent that is not on the "right-most border of correctness" to do nothing; and left-end agents are automatically on the border of correctness if they have incorrect states. Intuitively, what this rule does is generate a right-moving "wave of Θ-correctness" by applying the local Θ-gradient operator (fig. 2).

Proposition 4 *For any single-choice local check scheme Θ, F_Θ as defined in eq. 1 is a solution to T_Θ.*

PROOF. If Θ is single-choice, then for any n such that $T_{\Theta,n} \triangleq T_\Theta \cap \mathbb{C}_n \neq \emptyset$, there is a unique configuration $X_n \in T_n$. We have now to show that if X_0 is any configuration of size n and c is any live call sequence, then $(F_\Theta)_c^n(X_0)$ converges to X_n. We prove inductively that for each $j \leq n$ and some t, $X_t \triangleq (F_\Theta)_c^t(X_0) = X_n[1:j] \circ Z$ for some configuration Z. To this end, suppose inductively that $X_0 = X_n[1:j] \circ Y$ for some $j < n$, and that $Y(1) \neq \nabla_\Theta^+(X_n[j-2r-1:j])$. In words, $j+1$ is the first position in X_0 where the local check scheme Θ fails to hold. Then because of the second clause of the definition of F_Θ in eq. 1, all $k \leq j$, $F_\Theta(B_R(k, X_0)) = X(k)$. Let t the be first timestep in c such that agent $j+1$ is called (such a t must exist since c is assumed to be live). Then $X_{t-1} = X_n[1:j] \circ Y(1) \circ Z$ where Z is some configuration. Let $B = B_R(j+1, X_{t-1})$. Since $\Theta[B_-]$ holds by inductive assumption, the first clause of eq. 1 yields $F_\Theta(B) = \nabla_\Theta^+(B_-)$. Thus $X_t = X_n[1:j+1] \circ Z'$, completing the induction. □

4.2 The General Case: A Self-Organized Turing Machine

In general, local check schemes can allow multiple choices to follow a consistent subconfiguration. The radius-2 check scheme for the pattern $T_{100} \cdot T_{1000}$ is multi-choice because both 0 and 1 are acceptable states following the 2-neighborhood $00\widehat{1}00$:



Suppose we tried to solve the pattern $T_{100} \cdot T_{1000}$ along the lines of what we did for single-choice patterns. We'd have to choose a value for $\nabla_\Theta^+(00\widehat{1}00)$, say 0. In this case, the solution will write a string of repeats of 1000 until it reaches the right end. If the number of agents in the configuration is such that the end does not precisely line up with a complete unit of 1000, the only way the configuration can be solved is if the number of repeats of 100 back toward the left end of the configuration is changed. This means that the right-end agent has to communicate to left-end agents, sending a signal with the message: 'substitute another copy of 100 in place of 1000'.

The signal will travel toward the left until reaching the left-most instance of 1000, whereupon it should cause the agent whose local radius-6 neighborhood is $100100\widehat{0}1000100$ to substitute the choice of '0' state made originally with '1' instead. Having done this, the signal must dissipate and cause a new gradient wave to travel to the right, realigning all the repeats of 1000 relative to the new choice. When this wave reaches the right end, if a complete unit of 1000 fits, the configuration is solved. If not, another signal will have to be sent right-ward with the same message. Repeating these events, the configuration will eventually be solved (see fig. 3).

A similar idea works in general. We construct a local rule F_Θ, with radius $2r(\Theta)+2$, that self-organizes a brute-force "lexicographic" search through the set of possible locally correct structures. This rule comes to equilibrium only when the local ball around every agent is consistent with the check scheme Θ – if no Θ-consistent state of size n exists, F_Θ will never achieve equilibrium. Ten detailed local rules defining this construction appear in table 3. For ease, the rules assume we have $m+2$ extra states to work with.

The 10 rules work by implicitly implementing a virtual self-organizing Turing machine. As described in detail in the text accompanying figure 3, **Rules 1-4** essentially re-implement the single-choice algorithm in §4.1. A head – represented by an extra state denoted $\triangleright$ – is "born" (as per **Rule 1**) when an agent determines that it is on the "border of correctness". Then, the head propagates across the system (as per **Rules 2-3**), writing a trail of local Θ-correctness in its wake. Finally, the head halts, as per **Rule 4**, disappearing off the right boundary. Sometimes, the right-moving $\triangleright$-head is unable to halt and disappear, because it will encounter a situation in a multi-choice check scheme in which no possible local completion exists. In this case, **Rules 5-10** implement the signalling described in the discussion above, using $m+1$ extra states denoted $\triangleleft$, and $\triangle_1$, $\triangle_2$, …, $\triangle_m$.

The Turing machine is virtual, since it does not exist apart from states of the agents in the system. At any given time, it heads are emergent structures virtually hosted by several neighboring agents. Moreover, it is a *distributed* Turing machine because at any given time, there may be multiple heads present in the system; the rules encode their interactions so that eventually a consistent choice is made. The distributed Turing machine is implementing a self-organized *distributed signalling system* to coordinate the construction of long-range correlations.

Combining this construction with proposition 1, we have:

Theorem 1 *Local checkability is a necessary and sufficient condition for local solvability. Moreover, a pattern T with local check radius $LCR(T)$ has a robust solution with radius $2 \cdot LCR(T) + 2$ (using $m+2$ extra states).*

By definition of robustness, the solutions are guaranteed by theorem 1 to be completely self-repairing and time-delay agnostic. At the expense of an additional $2r(\Theta)+3$ added to the radius, the requirement for the extra states can be removed. The rule F_Θ can be quite slow, since it must search through (potentially) exponentially many configurations. An optimized linear time algorithm can be constructed [15].

5. THE RADIUS-STATE TRADE-OFF

Often times, agents have limits on either their memory or communication radius. One of the useful important properties of local check schemes is their ability to tradeoff between these two resource parameters. The pattern

$$T_{1000} = \{1000, 10001000, \ldots\},$$

for example, uses two states and has a local check radius of 2. If four states were available, one could encode essentially the same information as T_{1000} with the pattern

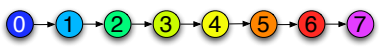
$$T_{1234} = \{1234, 12341234, \ldots\}.$$

However, T_{1234} only requires a radius of 1 to check locally. To show how this radius/state tradeoff works in general, we demonstrate two algorithms, one which trades off state for radius, and the other which trades off radius for state.

5.1 Trading State for Radius

We will first show how to solve the problem of decreasing agent state by increasing local radius. We illustrate this in one of the most important representative cases – the "coordinate" pattern.

A "k-coordinate pattern" is any state pattern in which each agent can determine locally k unique coordinate values. For instance, an 8-coordinate pattern is:



A local check scheme for the k-coordinate pattern is provided by the well-known *discrete gradient*:

$$\Theta(B) = \begin{cases} 1 & , \text{ if } B(-1) = B(0) - 1 \\ 0 & , \text{ otherwise} \end{cases}.$$

Figure 3: Left: 10 rules defining F_Θ. Right: Action of F_Θ for the pattern $T_{100} \cdot T_{1000}$.

Define F_Θ, with radius $2r(\Theta) + 2$, by:

Rule 1: If
- $B(-2r-1:-1)$ satisfies Θ, and
- $B(-2r:0)$ does NOT satisfy Θ,

then $F_\Theta(B) = \triangleright$.

Rule 2: If
- $B(0) = B(1) = \triangleright$, and
- $B(2r-1:-1)$ satisfies Θ,

then $F_\Theta(b) = \nabla_\Theta(b)^+$ when the latter exists.

Rule 3: If
- $B(-1) = \triangleright$ and
- $B(-2r-2:-2)$ satisfies Θ,

then $F_\Theta(B) = \triangleright$.

Rule 4: For the right-end agent, if
- $B(0) = \triangleright$, and,
- $B(-2r-1:-1)$ satisfies Θ,

then $F_\Theta = \eta(B)$ when the latter exists.

Rule 5: If
- The agent is as in **Rule 2**, BUT $\nabla_\Theta(B_-)^+$ does not exist, or
- the agent is as in **Rule 4**, BUT $\eta[B]$ does not exist,

then $F_\Theta(B) = \triangleleft$.

Rule 6: If
- $B(1) = \triangleleft$, and
- $B(-2r-1:-1)$ and $B(-2r:0)$ both satisfy Θ, and
- $B(0) = M(B)$,

then $F_\Theta(B) = \triangleleft$.

Rule 7: If
- $B(0) = \triangleleft$, and
- $B(-2r-2:-2)$ and $B(-2r-1:-1)$ both satisfy Θ, and
- $B(-1) \neq M(B(-2r-1:-1))$,

then $F_\Theta(B) = \triangle_{B(-1)}$.

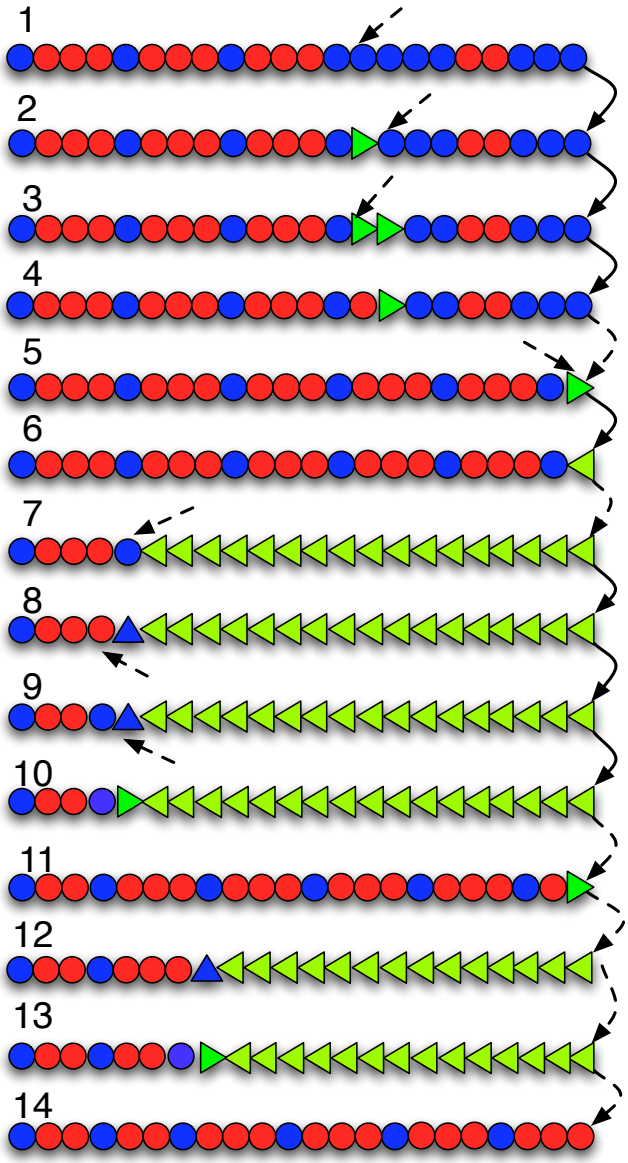
Rule 8: If
- $B(1) = \triangle_{B(0)}$ and
- $B(-2r:0)$ satisfies Θ, and
- $B(0) \neq M(B(-2r:0))$,

then $F_\Theta(B) = \Theta(B(-2r:0))^+$.

Rule 9: If
- $B(0) = \triangle_j$ for some $j \neq B(-1)$, or
- $B(0) = \triangle_{B(-1)}$ and $B(-1) = M(B(-2r-1:-1))$,

then $F(B) = \triangleright$.

Rule 10: For the agent is the left-end, if $B(0) = \triangleleft$ then $F(B) = \triangleright$.



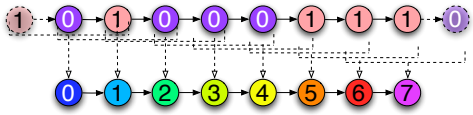
Action of F_Θ for the pattern $T_{100} \cdot T_{1000}$ on a typical initial condition with 22 agents: Straight arrows indicate agent whose action is being shown. Curved side arrows indicate time steps, solid = one step, dotted = many. Step $1 \to 2$: **Rule 1** generates $\triangleright$ head. Steps $2 \to 4$: **Rules 2-3** apply to propagate $\triangleright$ head rightward, leaving trail of Θ-correct states. Steps $4 \to 5$: Head continues to propagate until reaching right end. Step $5 \to 6$: there is no completion, since 1000 unit doesn't fit evenly; so **Rule 5** reverses $\triangleright$ into $\triangleleft$. Step $6 \to 7$: **Rule 6** propagates $\triangleleft$ to first point where new decision is possible. Steps $7 \to 10$: **Rules 7-9** replace state 0 in agent 3 with state 1, and reverse $\triangleleft$ back to $\triangleright$. $\triangle_0$ state arises in step 8 to prevent inappropriate updating in agent 3. Step $10 \to 11$: **Rules 2-3** apply again, propagating $\triangleright$ back to right end. Steps $11 \to 14$: Process repeats until configuration solved.

Technical definitions for Rules: In the above specifications, the following definitions apply. Let b be any r-neighborhood. Define "local coordinates" on b so that 0 is the position of the center agent, $+i$ is the agent i hops to the right of center, and $-i$ the agent i hops to the left. Given a $2r+1$-neighborhood B, let $B(i:i+2r)$ denote the r-neighborhood with central agent at position $i+r$. $\Theta^+(b)$ denotes the minimal $j > b(|b|)$ such that $\Theta(b(1:|b|-1) \circ j) = 1$ when such a j exists (where $|b|$ is the number of nodes in b). $M(B)$ denotes the maximal l such that $\Theta[b(1:|b|-1) \circ l]$. Given a Θ-consistent subconfiguration x, let $\nabla_\Theta(x)^+$ denotes the minimal $i \in S$ such that $x \circ i$ is Θ-consistent, when such an i exists. A subconfiguration y is *right-end Θ-consistent* if there is a configuration X with $\Theta(X) = 1$ such that $X = z \circ y$ for some z. Given a Θ-consistent subconfiguration x, let $\eta_\Theta(x)$ denote the minimal $i \in S$ such that $x \circ i$ is right-end Θ-consistent, when such an i exists.

This check scheme has radius of 1/2, meaning that it only makes use of information from the agent to the left. Threre is no radius-1/2 local check scheme for k-coordinatization with fewer than k states.

However, if we allow a somewhat larger radius, we can dramatically lower the required amount of state. Another well-known idea in computer science is the *DeBruijn sequence*: for m states and window-length n, a deBruijn sequence $B(n,m)$ is an m-ary sequence of length-m^n in which each length-n m-ary sequence arises exactly once as a contiguous subsequence in $B(n,m)$, wrapping around at the end. For example, an instance of $B(3,2)$ is 10111000. It is a classic and simple result that such sequences exist for all m and n [4].

The key realization is to think of the view within each length-n window in $B(n,m)$ as encoding a unique position along a line up to m^n positions in length. For example, a coordinate gradient of length 8 can be encoded by $B(3,2)$:



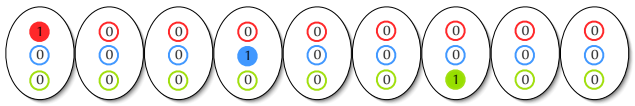
That is, we make the encoding $101 \to 0$, $010 \to 1$, $100 \to 2$, &c. Hence, a system that forms the "DeBruijn repeat pattern" $T_{B(n,m)}$ is self-organizing a length-n encoding of the m^n-coordinatized pattern.

To get a system to self-organize the pattern $T_{B(n,m)}$, we must find a local check scheme for this pattern. The key fact here is that $B(n,m)$ has a radius $\lceil (n+1)/2 \rceil$ check scheme defined by accepting precisely the $n+1$ windows in the pattern. For example, the radius-3/2 local check scheme accepting precisely the 3/2-neighborhoods $10\widehat{1}0$, $01\widehat{0}0$, $10\widehat{0}0$, $00\widehat{0}1$, $00\widehat{1}1$, $01\widehat{1}1$, $11\widehat{1}0$, and $11\widehat{0}1$, is a local check scheme for $B(3,2)$. Hence, the radius-1/2 discrete gradient check scheme in 8 states can be replaced by a radius-3/2 check scheme in 2 states. More generally, a radius-1/2 check scheme for the 2^n-coordinate pattern can be replaced with a radius-$\lceil (n+1)/2 \rceil$ check scheme in 2 states.

Coordinate patterns represent only one type of local check scheme. However, the DeBruijn encoding idea can easily be modified for any local check scheme to make a general state/radius tradeoff procedure [15].

5.2 Trading Radius for State

Conversely, suppose we were given the pattern $T_{100000000}$ over the binary states $\{0,1\}$. This pattern has local check radius of 4. But now, instead of just one binary variable, suppose each agent had access to a three-variable state space $\{0,1\}^3$. Consider the local check scheme Θ accepting "staggered shifts" of the 100000000 pattern, each shift separated by three positions, in each of the three variables. Pictorially, this is:

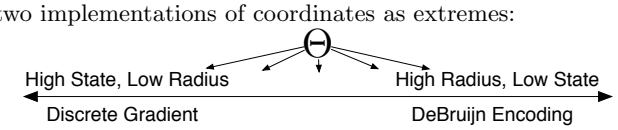


with Θ accepting precisely the 1-neighborhoods in the above figure. Each radius-1 neighborhood is unique, and the neighborhoods cannot appear in any other order than the original pattern, shift-repeated in each variable. The result is a radius-1 check scheme for $T_{100000000}$, now over 8 states.

A general version of this "cut and shift" construction can easily be made. Intuitively, instead of modeling agent internal states as a single m-ary variable taking values in S, suppose each agent now has access to the k-variable state space S^k, with m^k states. Let $B_{j,r}(i,X)$ denote the r-neighborhood in X at agent i, for variable j. For instance, in the configuration above, $B_{2,2}(5,X) = (0,1,\widehat{0},0,0)$.

Now, suppose we're given a radius r local check scheme Θ with m states, and our goal is to get a radius l local check scheme for the pattern T_Θ generated by Θ, for any $l < r$. Let Θ^l be the radius-l local check scheme on $M = 2\lceil (r-l)/(2l+1) \rceil + 1$ variables defined setting $\Theta^l(B = (b_1, \ldots, b_l)) = 1$ if and only if there is an $X \in T_\Theta$ and $i \leq |X|$ such that $B_j = B_l(i+j(2l+1), X)$ for $j = -(M-1)/2, \ldots, (M-1)/2$. Evidently $\Theta^l((b_1, \ldots, b_l)) = 1$ IFF $\Theta(b_1 \circ b_2 \circ \ldots b_l) = 1$. Hence for any Θ^l-satisfying configuration, the original pattern can be "read off" from the first variable b_1, or in a shifted version in any other variable. Θ^l is thus a radius-l check equivalent to Θ.

The continuum: Combining the two complementary directions of radius-state tradeoff, we have shown that there is a continuum between high-state/low-radius and high-radius/low-state implementations of equivalent check schemes, with the two implementations of coordinates as extremes:



6. PATTERN DESCRIPTION

Our driving problem is building a "compilation" procedure – an algorithm whose input is a solvable pattern T and a resource limitation, and whose output is explicit robust solution to T respecting that limitation. To serve as the input to such a procedure, we must find a language for compactly describing infinite patterns. An extremely simple approach is *local feature invariance*: the user specifies a set of *sample configuration* that are instances of the pattern, together with a *feature radius* at which she'd like the features of the samples to be preserved. We then compute the simplest local check scheme consistent with those features.

Formally, let $\mathcal{X} = \{X_1, X_2, \ldots, X_K\}$ be a finite set of configurations (the samples) and fix an $r > 0$ (the feature radius). Let

$$\mathcal{B}(\mathcal{X}, r) = \{B_r(i, X) | X \in \mathcal{X},\ i \in \{1, \ldots, |X|\}\},$$

that is, the set of all distinct r-neighborhoods present in the samples. Define a local check scheme $\Theta(\mathcal{X}, r)$ to accept only r-neighborhoods consistent with the samples, i.e.

$$\Theta(\mathcal{X}, r)(B) = \begin{cases} 1, & \text{If } B \in \mathcal{B}(\mathcal{X}, r) \\ 0, & \text{otherwise} \end{cases}.$$

Denote the pattern generated by $\Theta(\mathcal{X}, r)$ as $T(\mathcal{X}, r)$. By definition, $T(\mathcal{X}, r)$ is the set of all configurations Y such that $\Theta(\mathcal{X}, r)$ accepts every r-neighborhood in Y. Since all the r-neighborhoods accepted by $\Theta(\mathcal{X}, r)$ are exactly those appearing as r-neighborhoods in the sample set $\mathcal{X}$, $T(\mathcal{X}, r)$ can simply be thought of as the largest pattern consistent with the features of the original samples, at scale determined by r. In words, the samples are a set of "pictures" that indicate features to be generalized into the complete (and often infinite) pattern.

For example, let

$$\mathcal{X}_1 = \{(1000000100000001000000)\}.$$

This sample is evidently "trying" to capture the 1000000 repeat pattern. Taking large enough radius generates the "right" answer, i.e. $T(\mathcal{X}_1, 4) = T_{1000000}$. By choosing a smaller radius, other structures emerge. For instance,

$$T(\mathcal{X}_1, 2) = 10000 \cdot (T_{10000} \times T_{00000}) \cdot 00000.$$

Only a few samples are required to generate complex patterns. For the three-sample set

$$\mathcal{X}_2 = \{100100, 10001000, 1001000\},$$

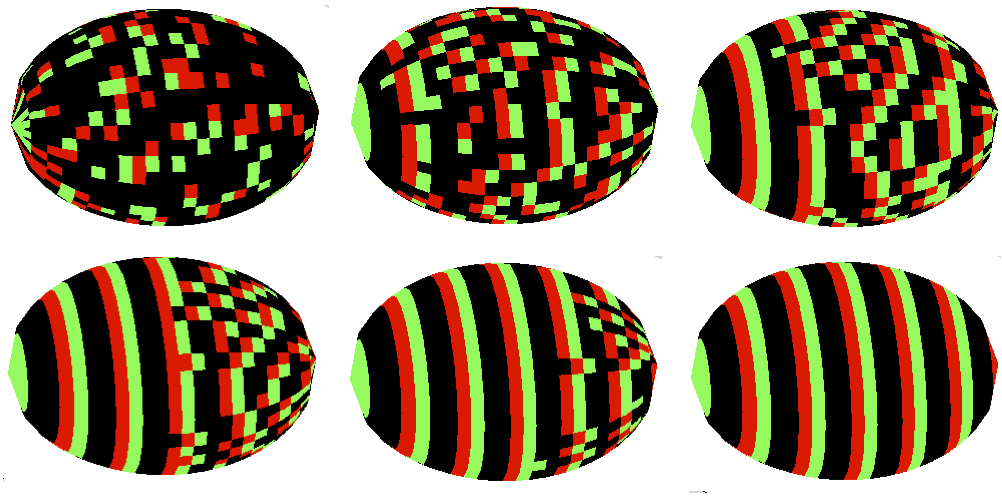


Figure 4: Snapshots from trajectory of rule outputted by global-to-local compiler described in section 6, on a ellipsoidal repeat pattern with rotational symmetry, with random initial condition. Compare to Fig. 1.

we have $T(\mathcal{X}_2, 3) = T_{100} \cdot T_{1000}$. Thus the logic and concatenation operators $\{\wedge, \vee, \circ, \times\}$ described in §3 are easily expressed by the invariant features of their samples.

Using this input framework, we can define a Global-to-Local compilation procedure. Given a pattern input described as a sample/feature pair $(\mathcal{X}, r)$, and a maximum possible communications radius R, the following three-step process compiles the input into a local rule solution:

$$\begin{aligned}(\mathcal{X}, r) &\longrightarrow \Theta(\mathcal{X}, r), \quad \text{(Local Feature Invariance)} \\ &\longrightarrow \Theta(\mathcal{X}, r)^{(R-1)/2}, \quad \text{(Radius} \rightarrow \text{State Tradeoff)} \\ &\longrightarrow F_{\Theta(\mathcal{X},r)^{(R-1)/2}}, \quad \text{(Construction)}\end{aligned}$$

where step 2 uses the construction in §5.2. Let $GC(\mathcal{X}, r, R)$ denote the result of this process. By construction, $GC(\mathcal{X}, r, R)$ is a robust solution to the pattern $T(\mathcal{X}, r)$, using radius at most R. For example, with the input from above, $GC(\mathcal{X}_2, 3, 1)$ outputs a nearest-neighor local rule solving the pattern $T_{100} \cdot T_{1000}$.

We have implemented this Global-to-Local compiler in Matlab. Code is available for download at `www.people.fas.harvard.edu/~yamins`.

7. APPLICATION AND GENERALIZATION

Our paper focusses on one-dimensional systems. Many questions in spatial multi-agent systems turn out to be essentially one-dimensional in nature, even when the spaces nominally have more complex geometry. For example, one of the basic techniques in the field – the discrete gradient – is precisely meant to construct a one-dimensional coordinate system, whatever the dimension of the underlying space [1, 2]. Moreover, many spatial computers naturally produce structures with a high degree of symmetry, rendering patterns trivial along all but one dimension. The radial symmetry of the early *Drosophila* embryo forms a one-dimensional repeat pattern, even though the organism is a three-dimensional ellipsoid, thus allowing our techniques to apply (see fig. 4). And, artificial self-assembly systems that do construct nontrivial higher-dimensional patterns often treat the multiple dimensions independently [7, 9].

In addition, we have studied the application of these techniques to more complex underlying geometries. We find, and will present in future work, that almost all the results discussed in this paper generalize. The 1-D multi-agent model presented in §2 generalizes simply by replacing the 1-D directed line graph with a graph $\mathcal{G}$ describing some other space. The proof of prop. 1 also makes no specific reference to the 1-D model, so local checkability as a necesary criterion is a very general result. In fact, local checkability is useful for almost all quasi-regular underlying spatial graphs [15]. The generalizations of the radius/state tradeoff algorithms and local feature invariance technique are similarly straightforward.

Generalizing the sufficiency construction is more complex. The basic idea is to replace "zero-dimensional" point-like Turing heads with higher-dimensional wavefronts, making the tie to signalling waves more explicit. With this generalized "wave-computing" approach, it is possible to design flexible higher-dimensional distributed signaling systems [15]. In future work, we will assemble these components into generally applicable global-to-local compiler.

8. REFERENCES

[1] H. Abelson et al. Amorphous computing. *Comm. ACM*, 43(5), 2001.

[2] W. Butera. *Programming a Paintable Computer*. PhD thesis, MIT, 2002.

[3] J. Conway. The game of life. *Scientific American*, March 1970.

[4] N. DeBruijn. A combinatorial problem. *Indagationes Math.*, 8, 1946.

[5] E. Klavins. Directed self-assembly using graph grammars. In *Foundations of Nanoscience*, 2004.

[6] M. Kloetzer and C. Belta. Hierarchical abstractions for robotic swarms. In *Proc. IEEE ICRA 06*, 2006.

[7] R. Nagpal. *Programmable Self-Assembly*. PhD thesis, MIT, 2001.

[8] R. Olfati-Saber et al. In *Proc. CDC03*, 2003.

[9] G. Poulton et al. *Agent Theories, Architectures, and Languages*, 2004.

[10] D. Rus et al. Self-reconfiguration robots. *Communications of the ACM*, 45, March 2002.

[11] J. v. Neumann. *The Theory of Self-reproducing Automata*. U. Illinois Press, 1966.

[12] G. Werner Allen et al. Monitoring volcanic eruptions. In *Proc. EWSN 05*, 2005.

[13] S. Wolfram. *Rev. Mod. Phys.*, 55, 1983.

[14] L. Wolpert. Positional information. *J. Theor. Bio.*, 25(1), 1969.

[15] D. Yamins. *A Theory of Local to Global for One-Dimensional Multi-Agent Systems*. PhD thesis.