Global-to-Local Programming: Design and Analysis for Amorphous Computers

Radhika Nagpal and Daniel Yamins
School of Engineering and Applied Sciences (Computer Science)
Harvard University

SASO 2007 Tutorial
First IEEE International Conference on Self-Adaptive and Self-Organizing Systems
July 8, 2007, Boston, USA

Emerging Technologies

Multi-agent Systems in Nature

Motivation

- Emerging Engineered Systems
  - Massive numbers of agents
  - Simple, cheap, interchangeable and expendable
  - Embedded in space
  - Local interactions, sensing, and actuation
  - Fault-tolerance without external intervention

- Plethora of similar systems in nature
  - Can we learn "engineering principles" for self-organization by looking at these systems?

Global-to-Local Challenge

- How do we program robust global behavior from the cooperation of vast numbers of limited and unreliable agents?

- How do we program complex desired global goals from the local interactions amongst ensembles of comparatively simple agents?
Tutorial Outline

• Part 1: Global-to-Local Programming
  – Amorphous Computing Model
  – Two Spatial Amorphous Computing Languages
    • Demonstrate global-to-local compilation
    • Inspired by principles from developmental biology
    • Encode self-repair and scale-invariance
  – Applications

• Part 2: Global-to-Local Theory
  – Formal framework for addressing fundamental questions:
    • When do global-to-local compilers exist?
    • What is the minimal complexity of local rules?

Amorphous Computing Model

An Amorphous Computer

A Simple Pattern: Polka Dots

Example: A Simple Pattern

Example: A Simple Pattern

[Abelson, Knight, Sussman, 1996]

[St Clair, Principles of Development]
Polka Dot Program

Global Properties
- Dot spacing
- Self-repair

Agent Behaviors
- State (differentiation)
- Randomness
- Gradients and thresholding

“Lateral Inhibition”

How would you make a bull’s eye pattern?

How much more complex are these?

Or These?
Global-to-local Compilation for Pattern/Shape Formation

Two Examples from Amorphous Computing
- Origami Shape Language
- Directed Growth Language

[for others, see Abelson et al CACM 2000]

Suppose we want to create an arbitrary pattern

Origami Shape Language (OSL)  
Nagpal, 2001

Global Shape Description
using a language based on geometry axioms from paper-folding (origami)

Agent Program
(all agents have identical program)
using biologically-inspired primitives

A Set of Construction Rules  
(Huzita’s Axioms of Origami)

A "Generative" Global Program

OSL Program for Drawing a CMOS Inverter

Within-region IN (color h1 "poly")
Within-region OUT (color h1 "poly")
Within-region CNTR (color v2 "poly")
Within-region CONN (color v2 "poly")
Within-region CONTACT (color v1 "contacts")
Within-region CONTACT (color v2 "contacts")
Within-region CONTACT (color v3 "contacts")

At this level of description, there are no "agents"
Compilation from Global Description to Local Program

- **Points** and **lines** are groups of agents
- Each **global construction rule** maps to a local agent behavior that uses a composition of primitives (e.g. state, gradients, etc)

**Agent program** is a sequence of local rules + local state

Initial Conditions: “Determinants”

Morphogen Gradients

Implementing the Rules

Individual Element

```
(define (axiom2-rule i1 i2 g1 g2 gend)
 (if i1 (create-gradient g1))
 (if i2 (begin (wait-for-gradient g1) (create-gradient g2)))
 (if i1 (begin (wait local-delay) (create-gradient gend)))
 (wait-for-gradients gend)
 (if (<= (abs (- g1 g2)) threshold) #t #f)
)
```

An Inverter Program
Pattern-Formation on an Amorphous Computer

OSL Cup Program

Example: Cup Formation

Many Shapes and Patterns

Analyzable Properties

- Resource consumption
  - Local state: Boolean per distinct crease, point, region
  - Gradients: Many, but short-lived (~Storage for 6 Gradients)

- Agent code conservation

- Robustness
  - Geometric patterns in the face of irregular grids, limited and variable numbers of agents, asynchronous timing, random agent death, etc.

- Abstract Global Description
  - Local agent behavior automatically compiled
  - Significant descriptive power
    - Any 2D Euclidean rule pattern (Huzita), tree-based shapes (Lang)
Scale-invariance and Variations in Morphology


Scale-invariance

Same agent program

Related Shapes and Asymmetric Scaling

Another Arbitrary Shape

Repeating Patterns

; Segment into 8 COMpartMENTS
; Execute
; inverse pattern program
; within a region

\{within-region r1
\{create- Inverter
\left-border1
\right-border1\}\}
Directed Growth Language
Kondacs, 2003

Agent can reproduce and/or die

Global Shape Description
Network of overlapping "circles"

Agent Program
Single agent that "grows" the shape

Constructing a 2D shape

Global Description
- Construct using overlapping circles with different radii
- Each parent circle locates children in its internal coordinate system
- Network represents "growth order"

Global-to-Local Compilation

Global to local mapping
- Grow circle of given radii
- Circle color represents agents' state in circle
- Agents compete to become reference points and new circle centers; Reference points emit "active gradients"
- Shape = Set of rules (proportional to # distinct circles)

Self-Assembly using directed growth

Kondacs, Chang, 2003
Self-Repair and Regeneration

Regenerating structures

“Rule of Normal Neighbors”

Absence of neighbor causes circle to recreate its neighbor, which in turn recreates its neighbor—thus regenerating the broken structure.

Self-repairing patterns

If a line is broken, one part dies off and the other regrows.

Global-to-Local Programming

- Amorphous Computing Research
  - Growing Point Language (Coore, 1999)
  - Origami Shape Language (Nagpal, 2001)
  - Directed Growth (Kondacs, 2002)

- Paintable Computing Language (Butera, 2002)
- Proto Language (Bachrach and Beal, 2006)

Summary

- Other Global-to-local examples
  - Graph Grammars for self-assembly (Klavins, et al)
  - DNA self-assembly (Winfree et al)
  - Macroprogramming for Sensor Networks (e.g. Madden et al, Newton and Welsh)
Global-to-Local Challenge

- How do we program robust global behavior from the cooperation of vast numbers of limited and unreliable agents?
- How do we program complex desired global goals from the local interactions amongst ensembles of comparatively simple agents?

Amorphous Computing Approach

High-level Programming Languages in terms of Goals / Tasks

Amorphous Computing Approach

High-level Programming Languages in terms of Goals / Tasks

Global-to-Local Compiler

Program for Myriads of Agents

Contributions

Amorphous Computing Research:

1. Provided a small but powerful catalog of robust, analyzable, bio-inspired local behaviors (“primitives”)
2. Demonstrated high-level languages and global-to-local compilation, by composing primitives

Biologically-Inspired Primitives

1. Morphogen Gradients
2. Neighborhood query
3. Local Inhibition
4. Local Monitoring
5. Agent Differentiation
6. Agent-to-agent Contact
7. ...

Useful local behaviors in many contexts, robust, simple and analyzable

Global-to-Local Compilation

- High level languages solve complex global tasks by using principled ways to compose simpler primitives
  - Easily express large (possibly infinite) classes of complex global goals
  - Robustness through bio-inspired primitives
  - Analysis becomes tractable and reusable
  - Encode complex notions, such as scale-invariance and regeneration, into the language itself

Conclusion

Amorphous Computing showed that one can design domain-specific high-level languages that compile abstract global descriptions into robust local rules for identically-programmed agents.

- Open Questions:
  - When do such global-to-local compilers exist?
  - How complex must an agent be? (in state, local radius, time) in order to solve a given problem
Some Applications

Applications of Gradients

- Pervasive Computing
  - Gradient-based languages for localization and coordination (e.g. Mamei and Zambonelli)
- Sensor Networks and Robot Swarms
  - Gradient-based localization, tracking and movement coordination (e.g. Proto, McLurkin)
- Synthetic Biology
  - Genetically engineer pattern formation and control in bacteria (e.g. Weiss et al)

Proto Language
(Bachrach and Beal, 2006)

Programming Spatial Computers
- Global abstraction is continuous space and time
- Compiler takes care of discretization into local agent behaviors
- General-purpose functional language, with primitives like active gradients and streams

Applications of Self-Assembly

- Many problems in robotics are related to pattern and shape formation
- Examples
  - Self-reconfiguration and self-assembly in modular robots (e.g. Stoy et al)
  - Collective construction by simple robots (e.g. Werfel et al)
  - Formation control in robot swarms (e.g. Cheng et al, Belta et al)

Self-Assembly in Robotics using “Directed Growth”

Self-Reconfigurable Robots
- (Stoy and Nagpal, 2004)
- Structure “grows” by recruiting wandering modules using a recruitment gradient

Collective Construction
- (Werfel, 2006)
- Structure “grows” by robots bringing materials

Global-to-local approach
- Both allow user-specified global shapes & rely on decentralized, simple, and robust agent behaviors

Collective Construction

Werfel et al, 2006

Hardware demonstrations of simple robot behaviors for 2D collective construction

Use RFID-based “stigmergy” for indirect interaction between robots through the structure

[Justin Werfel, Matt Valente, Crystal Schu]