

Amorphous Computing's Programming Languages

Jacob Beal
April, 2005

Amorphous Motivation

- Biological programs are robust
 - e.g. morphogenesis, repair
- Computer programs are fragile
 - Is our hardware too perfect?

We consider it a language problem.

Can we engineer with biologically inspired limitations?

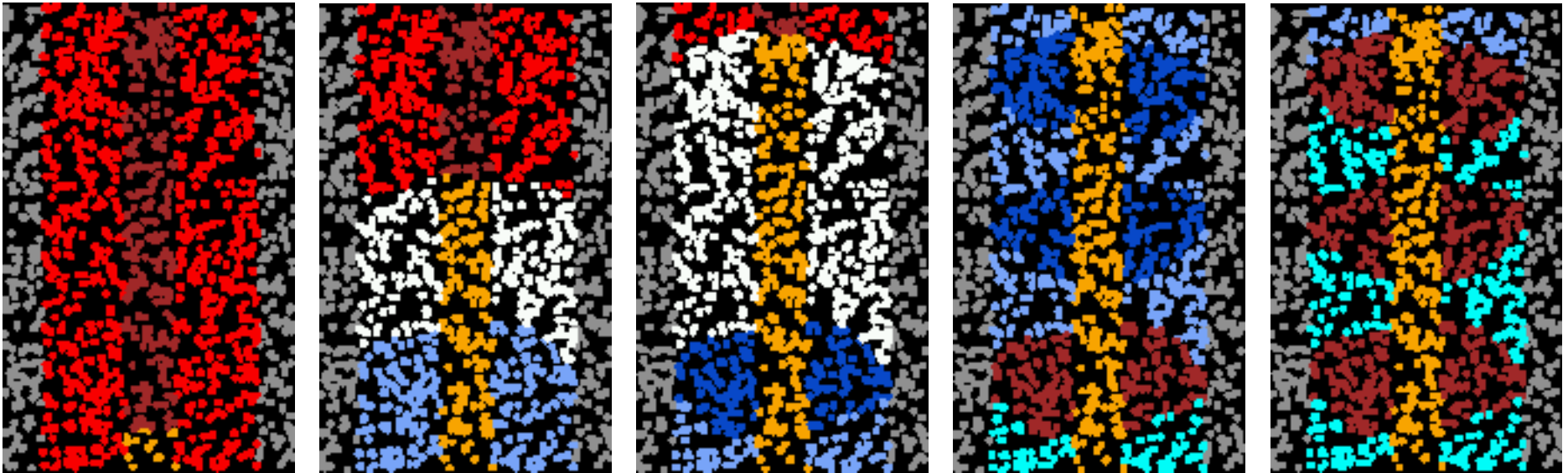
- Myriad unreliable, simple devices
- Distributed through space, talk only w. nbrs
- Identically programmed, simple initial conditions
- No high-level services (e.g. time, coordinates, naming, routing)
- No “user”, no centralization
- *(often homogenous and immobile)*

What is a Language?

- Standardized library of parts [**Primitives**]
- Rules for building bigger parts by combining smaller parts [**Composition**]
- Mechanism for naming parts and treating them like primitives. [**Abstraction**]

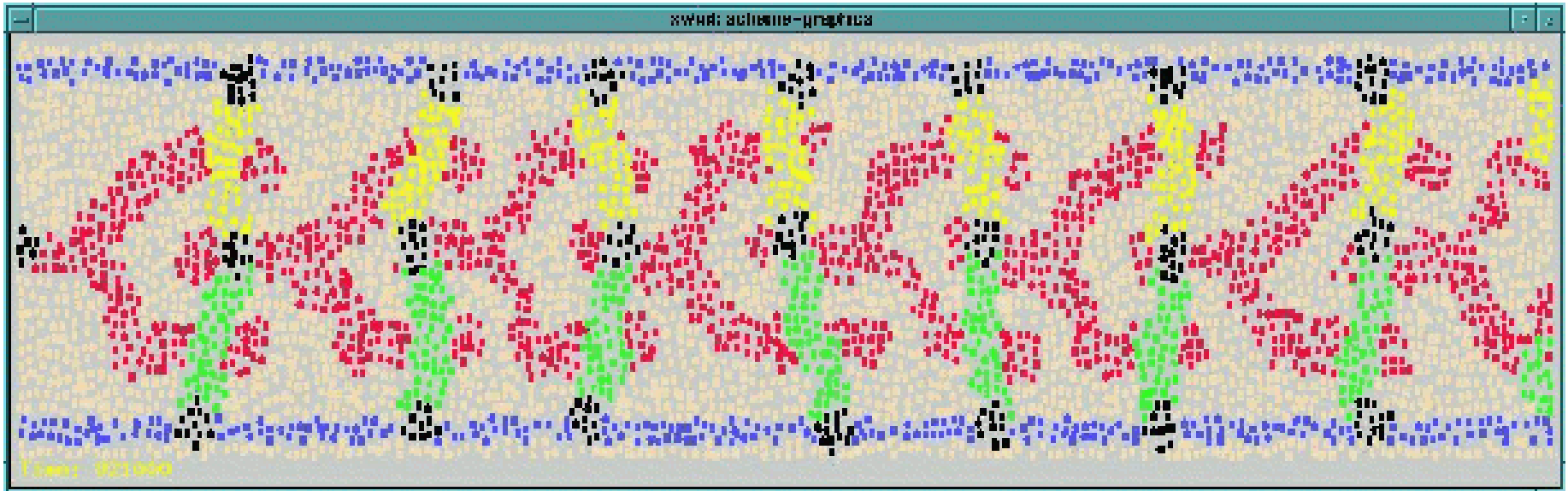
What is explicit and what is implicit?

Microbial Colony Language



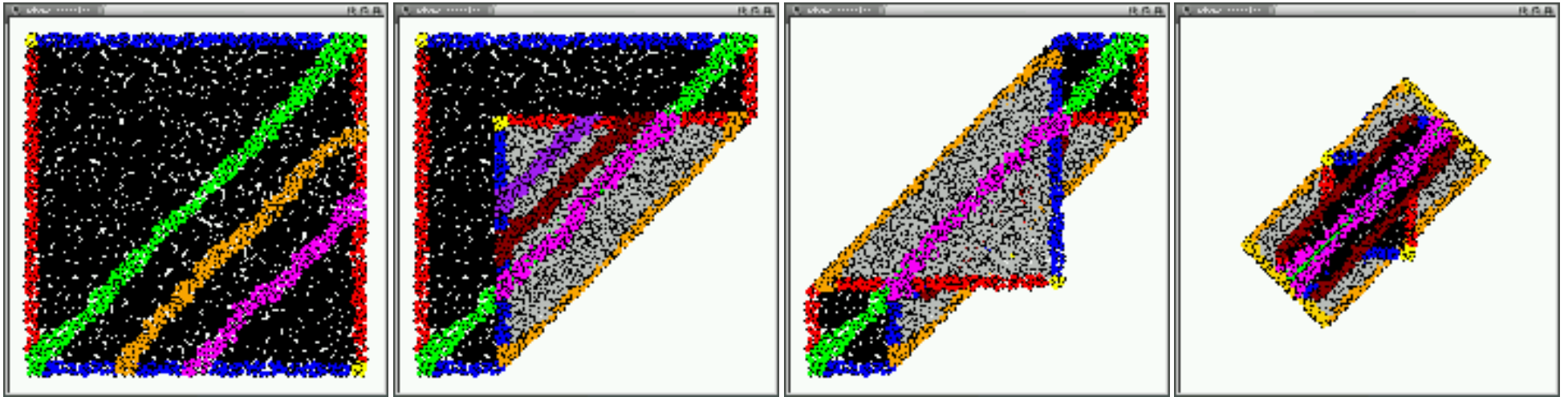
- MCL (Weiss, Homsy & Nagpal, 1998)
 - Explicit: marker diffusion & decay, events
 - Closely targetted at engineered bacteria

Growing Point Language



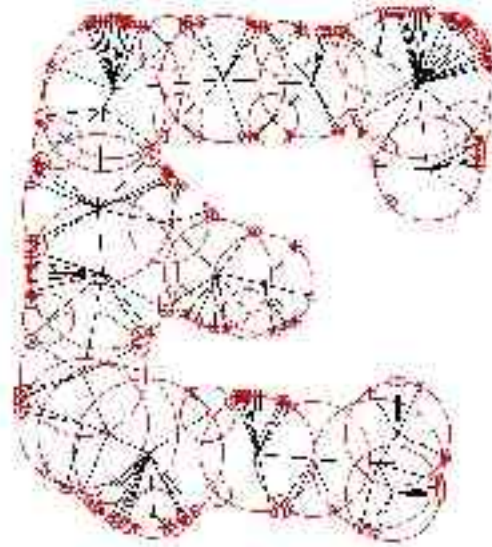
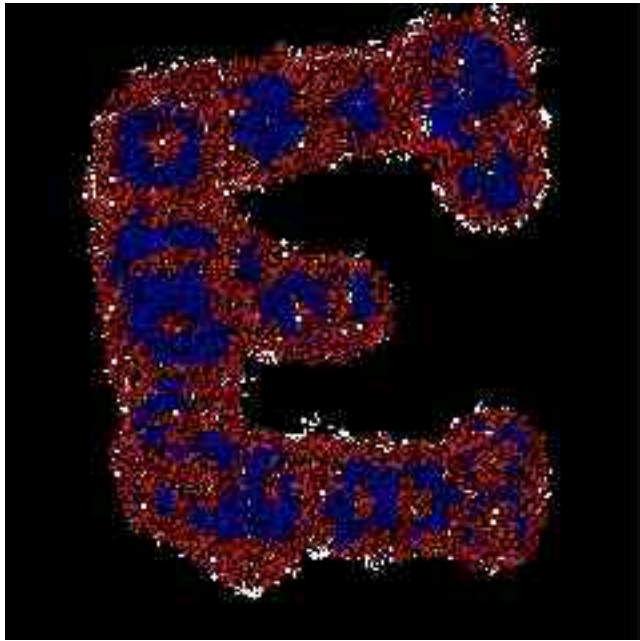
- GPL (Coore, 1999)
 - Explicit: botanical growing points, chemical tropism
 - Can construct arbitrary planar graphs

Origami Shape Language



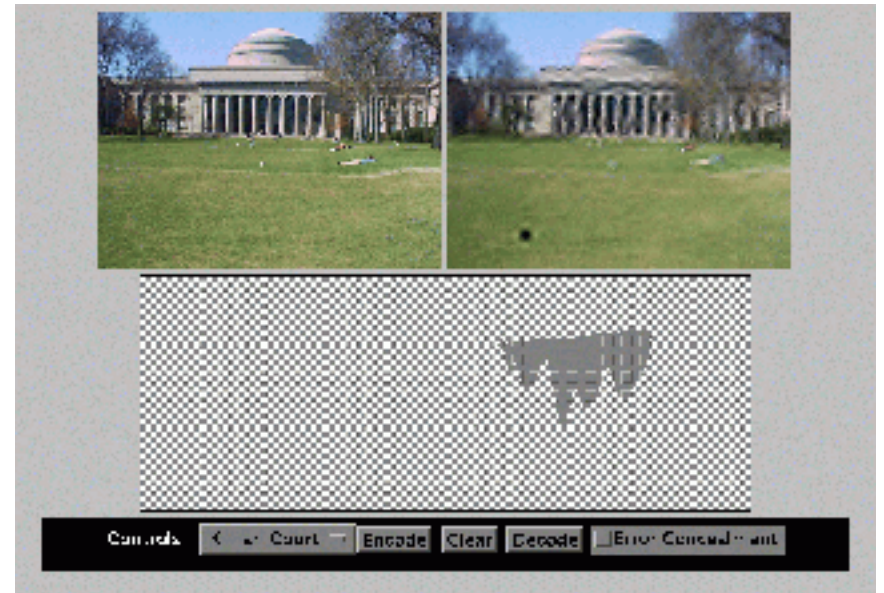
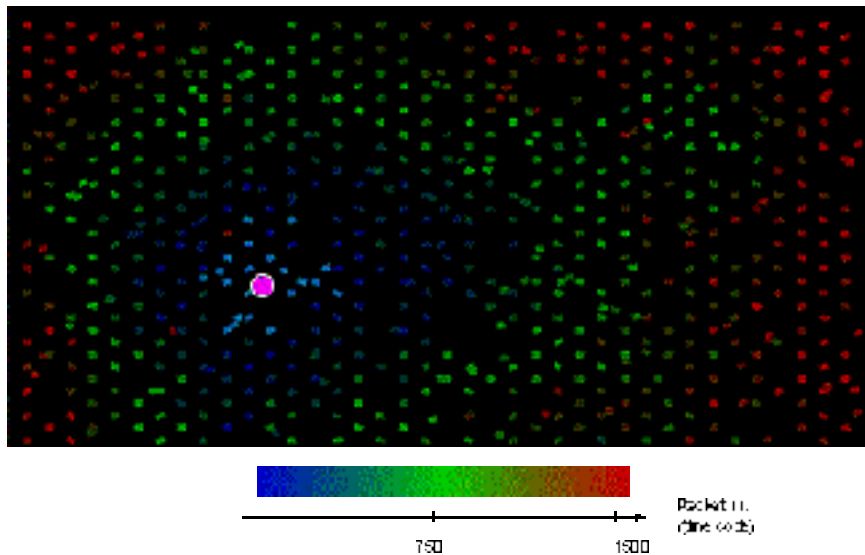
- OSL (Nagpal, 2001)
 - Explicit: geometry and folding sequence
 - Huzita's 6 axioms (e.g. fold Line-1 onto Line-2)
 - Predicts *drosophila* morphological variation

Growing 2D Shapes



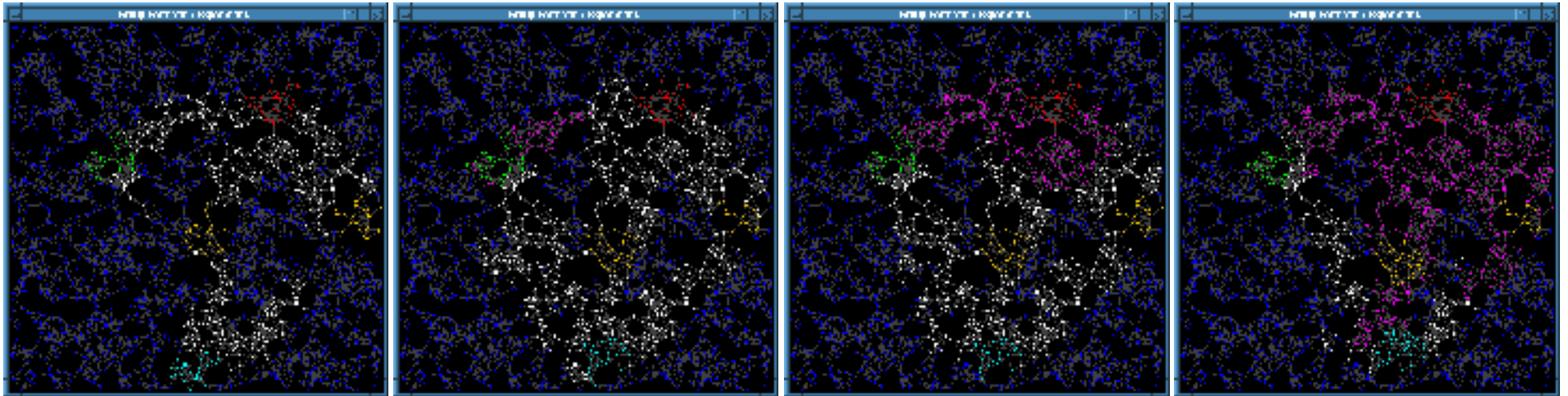
- Morphogenesis Language (Kondacs, 2003)
 - Explicit: shape
 - Grows from a single point, filling space with cells
 - Temporary structure garbage collect via apoptosis

Paintable Computing



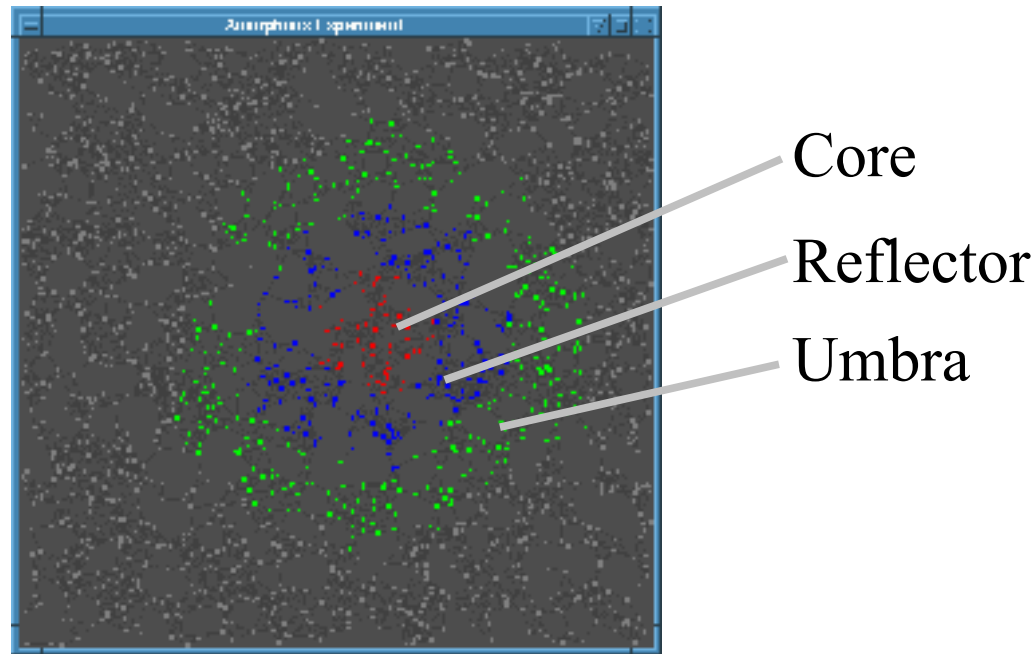
- PFrag Toolkit (Butera, 2001)
 - Explicit: local neighborhood behavior
 - Mobile program fragments replicate and diffuse

Dataflow Hack



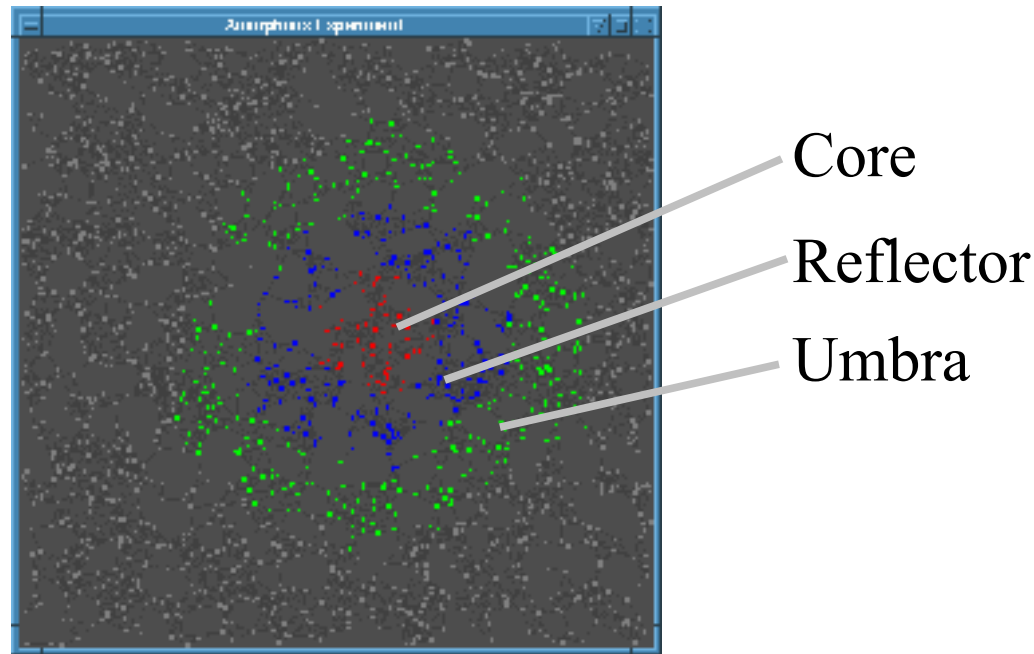
- (Beal & Newton, 2002, unpublished)
 - Explicit: simplified functional LISP
 - Data flows through space to nodes that operate on it

Persistent Node



- (Beal, 2003)
 - Mobile virtual node, useful primitive
 - Regrows lost parts (may split!)
 - Moves following local gradient

PN vs. Virtual Mobile Node



- Strong liveness, looser consistency guarantees
- PN does not assume time, location, or localcast
- PN operates on stationary particles

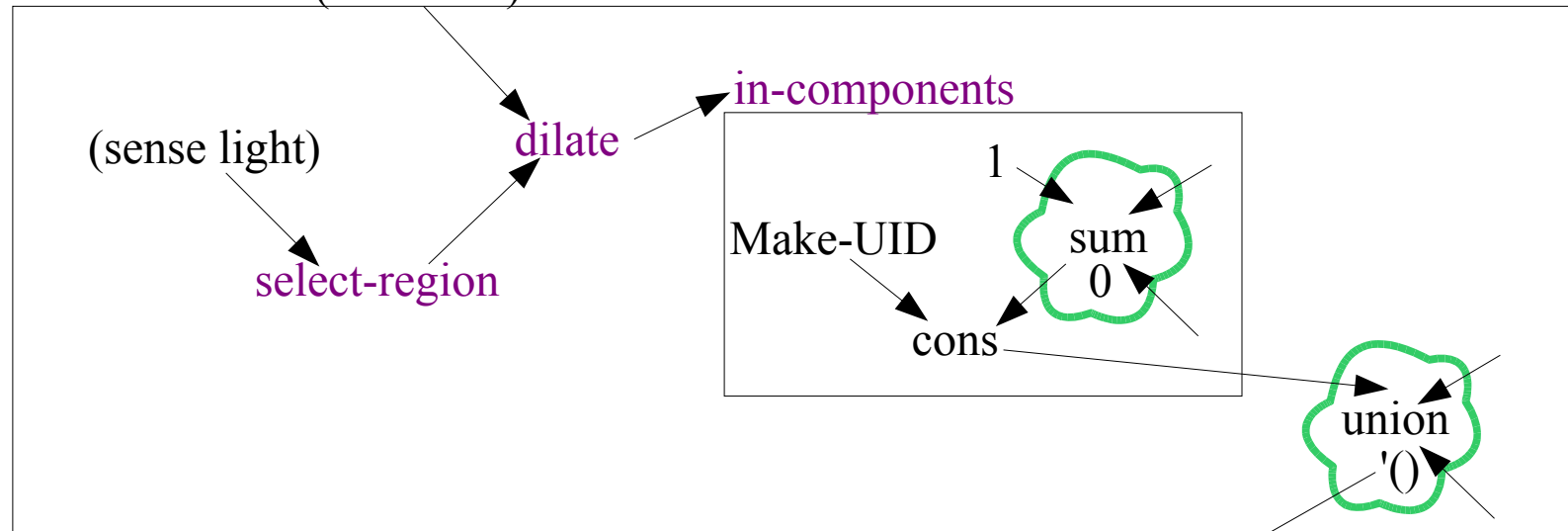
AML

```
(defun (measure-blobs fuzziness)
  (let ((r (select-region (sense :light))))
    (region-join
     (in-components (dilate r fuzziness)
      (cons
       (make-uid)
       (region-join 1 :join #'sum :base 0)))
     :join #'pushnew :merge #'union :base '()))))
```

- AML (Beal, 2004; Beal & Sussman 2005)
 - Explicit: behavior in the context of regions
 - **Regions** are partially instantiated first-class objects
 - **Region-join** aggregates state
 - Specify behavior in terms of homeostasis conditions

AML

Measure-Blobs (fuzziness)



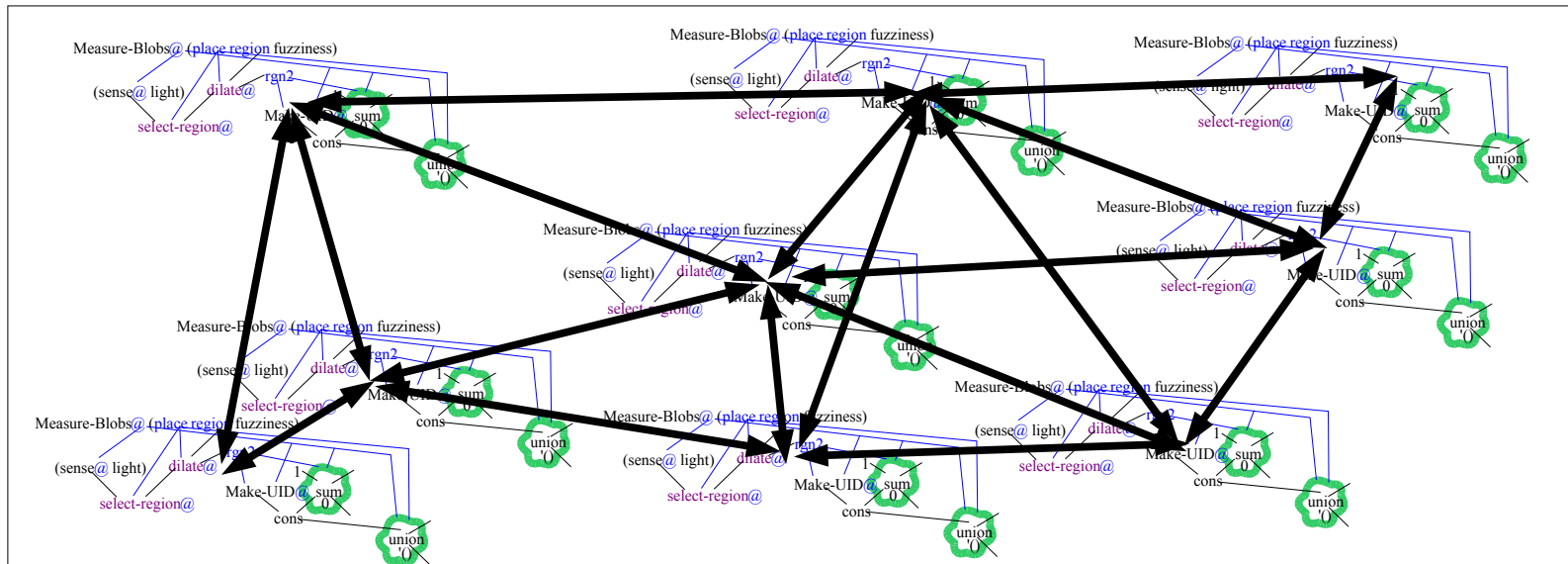
- AML (Beal, 2004; Beal & Sussman 2005)
 - Evaluation instantiates a structures of streams
 - Dried up streams are garbage collected

AML: Global to Local

```
(defun (measure-blobs@ place region fuzziness)
  (let ((r (select-region@ region (sense@ place :light))))
    (region-join@ place region
      (let ((rgn2 (dilate@ region r fuzziness)) ;; in-components@
            (cons
              (make-uid@ place rgn2)
              (region-join@ place rgn2 1 :join #'sum :base 0)))
          :join #'pushnew :merge #'union :base '()))))
```

- AML (Beal, 2004; Beal & Sussman 2005)
 - Make spatial context explicit ([region](#))
 - Transform to behavior at each point ([place](#))

AML: Local to Global



- AML (Beal, 2004; Beal & Sussman 2005)
 - Localized version is instantiated on nodes
 - Discrete approximation of global specification

Future Directions

- Actuation
- Language Development
 - Composition
 - Abstraction
 - Primitives
- Testing on real hardware, applications

Open Problems

- Analysis
 - Convergence
 - Behavior on continuously evolving topology
- Better Primitives