Programming Manifolds

Jacob Beal & Jonathan Bachrach  
MIT CSAIL  
September, 2006
Space-filling Computers

Distributed Control Systems

Biological Computing

Robot Swarms

Sensor networks

Peer-to-Peer Wireless Networks

Programmable Matter
Amorphous Medium Approach
Amorphous Medium Approach

Program the space...
Amorphous Medium Approach

Program the space... approximate with a network.
Amorphous Medium Approach

The discretization hardly matters!
Global v. Local v. Discrete
Global v. Local v. Discrete

Program

Compiler

Kernel

Global

Local

Discrete
Amorphous Medium

- Manifold (locally Euclidean space)
  - Assume Riemannian, smooth, compact
  - Simple locally, complex globally
Amorphous Medium

- Points access past values in their neighborhood
  - Information propagates at a fixed rate \( c \)
- Evaluation is repeated at fixed intervals
Expressions

- An expression maps a manifold to a field
  \[ \text{rgn}: M \rightarrow (M \rightarrow \mathbb{R}) \]
Expressions

• An expression maps a manifold to a field

\[ \text{rgn}: M \rightarrow (M \rightarrow \mathbb{R}) \]
Operators

- Operators map fields to fields (= rgn 2)
  
  \[ =: (\mathbf{M} \to \mathbf{R}) \times (\mathbf{M} \to \mathbf{R}) \to (\mathbf{M} \to \mathbf{B}) \]
Composition & Abstraction

• Functional composition:
  - operator \circ expressions = expression
  - operator \circ operators = operator
  - scope \circ expression = operator

}\{ Lambda!

Purely functional pointwise computation
Computation over Neighborhoods

(or ((n xor) (nbrval x) (local x)))
Computation over Neighborhoods

\[(\text{or } ((n \text{ xor}) (\text{nbrval } x) (\text{local } x))\]

- \text{local, nbrval} select fields of neighborhood fields
Computation over Neighborhoods

\[(\text{or (} (n \text{ xor}) \ (\text{nbrval} \ x) \ (\text{local} \ x)\)\))\]

- \(n\) applies an operator to neighborhood fields
Computation over Neighborhoods

\[(\text{or} \ ((n \text{ xor}) \ (\text{nbrval } x) \ (\text{local } x)))\]

- Measures (e.g. \text{or}, \text{integral}) reduce fields to values
- Sugar: \((\text{reduce-nbrs or (xor x (local x))})\)
Conditional Computation

\[(\text{mux } x \ (\text{or} \ (\text{nbrval} \ (\text{restrict} \ x \ \text{vent})) \ #F))\]
Conditional Computation

\[(\text{mux } x \ (\text{or } \ (\text{nbrval} \ (\text{restrict} \ x \ x \ \text{vent})) \ #F))\]

- **restrict** limits the domain of a field
(mux x (or (nbrval (restrict x vent)) #F))

- operations proceed normally in the restricted field
Conditional Computation

\[
\text{mux} \ x \ (\text{or} \ (\text{nbrval} \ (\text{restrict} \ x \ \text{vent})) \ \#F)
\]

- \text{mux} constructs a field piecewise from inputs
- Sugar: \( (\text{if} \ x \ (\text{or} \ (\text{nbrval} \ \text{vent}))) \)
Computation with State

(delay default init)

- Previous values, current domain
Computation with State

(delay default init)

- Previous values, current domain
Computation with State

(delay default init)

- Previous values, current domain
Computation with State

\[(\text{delay} \text{ default init})\]

- Previous values, current domain
Computation with State

(letfend ((n 0 (+ n 1))) n)
Putting it all together

• State chains neighborhoods to arbitrary regions
  – Example: relaxation to calculate distance

```
(lambda (src)
  (letf
    ((d (∞ (mux src 0
          ((d (∞ (reduce-nbrs min (+ d nbr-range)))))))
      d))
```

![Diagram of a map with distances marked as infinity and a starting point labeled 0.](image)
Putting it all together

• State chains neighborhoods to arbitrary regions
  – Example: relaxation to calculate distance

\[
\text{(lambda (src)}
\quad (\text{letfed})
\quad \left((d \sim (\text{mux src 0}})
\quad \left((d \sim (\text{reduce-nbrs min (+ d nbr-range)})))))
\quad d))
\]
Putting it all together

- State chains neighborhoods to arbitrary regions
  - Example: relaxation to calculate distance

```scheme
(lambda (src)
  (letfed
    ((d  (mux src 0
      ((d ∞ (mux src 0
        (reduce-nbrs min (+ d nbr-range))))))
      d)))
```
Putting it all together

• State chains neighborhoods to arbitrary regions
  – Example: relaxation to calculate distance

\[
\text{(lambda (src)}
(\text{letfe}}d
((d \infty (\text{mux src 0}
\ \ \ \ \ \ \ \ \ \ \ (\text{reduce-nbrs min (+ d nbr-range)})))))))
\text{d}))
\]
Target Tracking

(def local-average (v) (/ (reduce-nbrs v integral) (reduce-nbrs integral 1)))
(def gradient (src)
  (letfed ((n infinity
            (+ 1 (mux src 0 (reduce-nbrs min (+ n nbr-range)))))
            (- n 1)))
(def grad-value (src v)
  (let ((d (gradient src))
         (letf ((x 0 (mux src v (2nd (reduce-nbrs min (tup d x)))))))
           x)))
(def distance (p1 p2) (grad-value p1 (gradient p2)))
(def channel (src dst width)
  (let* ((d (distance src dst))
          (trail (<= (+ (gradient src) (gradient dst)) d))
          (dilate width trail))
    (def track (target dst coord)
      (let ((point
              (if (channel target dst 10)
                (grad-value target
                 (mux target
                     (tup (local-average (1st coord))
                       (local-average (2nd coord)))
                    (tup 0 0)))))
                (tup 0 0)))
            (mux dst (vsub point coord) (tup 0 0))))))
Threat Avoidance

(defun exp-gradient (src d)
  (letf (\(n src (max (* d (reduce-nbrs max n)) src)))
    n))
(defun sq (x) (* x x))
(defun dist (p1 p2)
  (sqrt (+ (sq (- (1st p1) (1st p2)))
           (sq (- (2nd p1) (2nd p2))))))
(defun l-int (p1 v1 p2 v2)
  (pow (/ (- -2 (+ v1 v2)) 2) (+ 1 (dist p1 p2))))
(defun max-survival (dst v p)
  (letf (\(ps 0 (mux dst
    1
    (reduce-nbrs max (* (l-int p v (local p) (local v)) ps))))
    ps))
(defun greedy-ascent (v coord)
  (- (2nd (reduce-nbrs max (tup v coord))) coord))
(defun avoid-threats (dst coords)
  (greedy-ascent
   (max-survival
dst
   (exp-gradient (sense :threat) 0.8) coords) coords))
Future Directions

• Continuous time evaluation
• Analysis of distortion from space discretization
• Evaluation on a changing manifold
• Actuation of the manifold
• Applications!