Problems of First-Class Functions over Space-Time

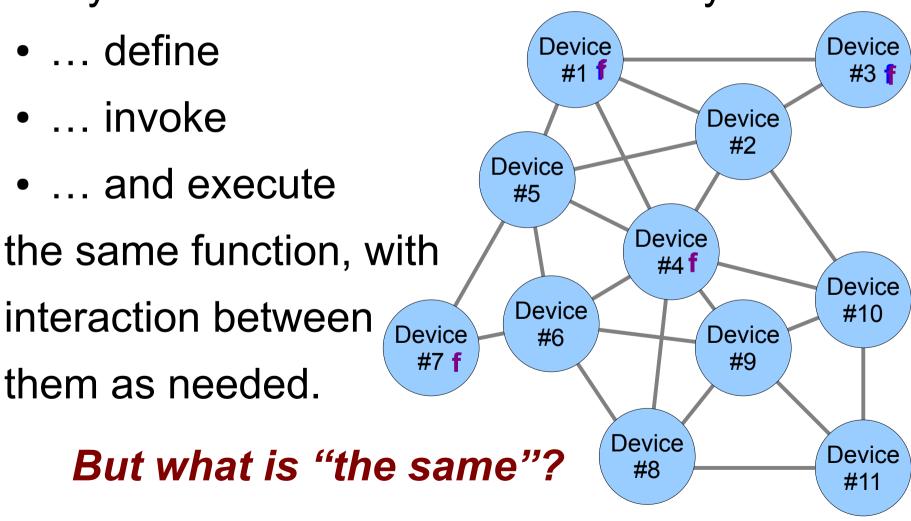
Jacob Beal

NEPLS, June 2012



What's a distributed first-class function?

Many different devices simultaneously...



Motivation: Aggregate Programming

Users / Applications

Global-to-Local Compilation

System of Myriad Interconnected Computing Devices P2P systems

Decentralized cloud

Sensor networks

Robotics

Massive multicore

Bioengineering

...

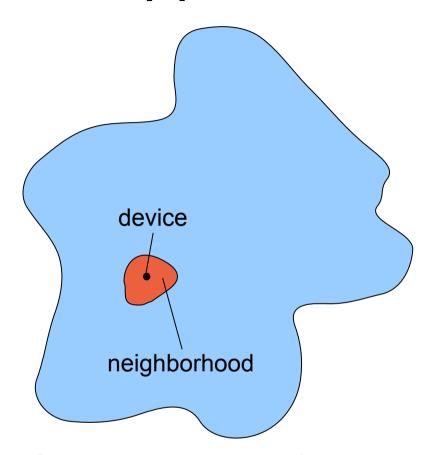
First Class Functions Checklist

- Function Calls
- Recursion
- Anonymous Functions
- Nested Functions
- Closures
- Higher-Order Functions
- Function-valued Fields
- Runtime Function Definition

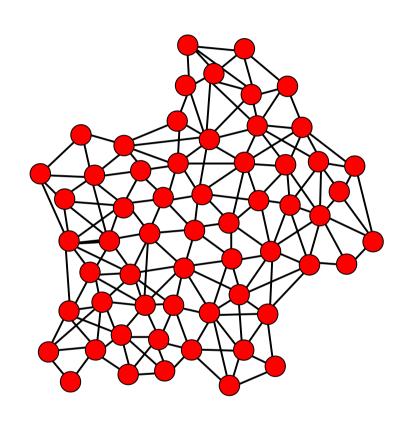
Prior models limit generality

- Many-to-one (i.e. client-server)
- One-to-Many (i.e. parallel batch processing)
- Limited many-to-many:
 - Specialized p2p
 - Population protocols
 - Replicated state machine
 - etc...

Approach: Continuous Model



- Continuous space & time
- Infinite number of devices
- See neighbors' past state

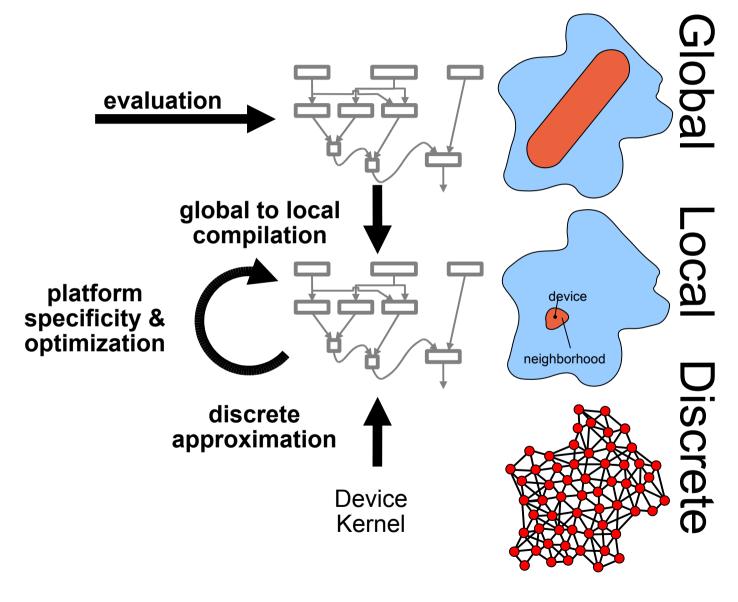


Approximable with:

- Discrete network of devices
- Signals transmitting state

Extreme case can specialize to all applications

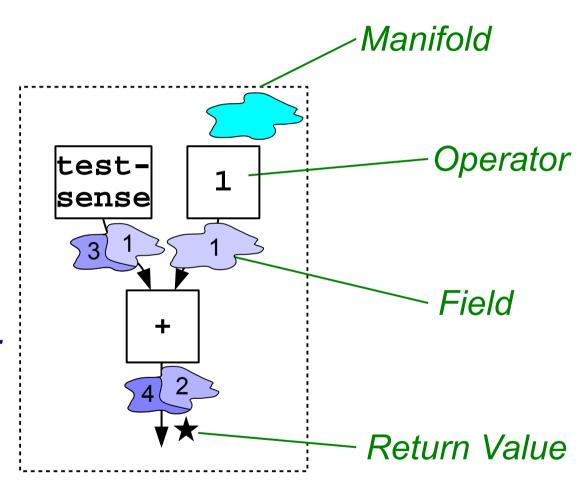
Proto



Continuous Space-Time Programs

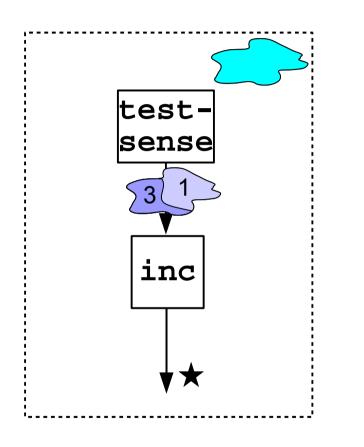
Well-defined iff: each operator's inputs and outputs have same domain.

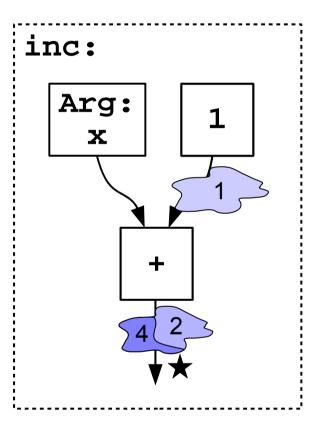
(except domainchange operators)



```
(+ 1 (test-sense))
```

Function Calls



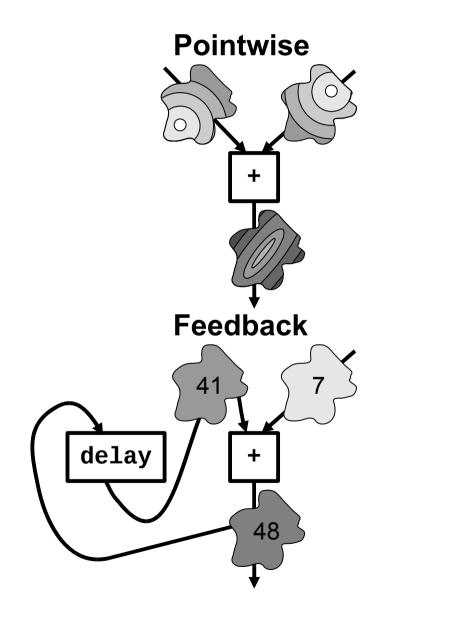


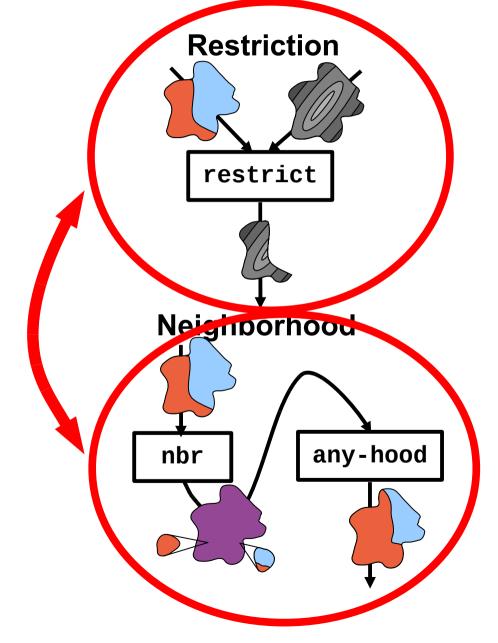
```
(def inc (x) (+ x 1))
(inc (test-sense))
```

First Class Functions Checklist

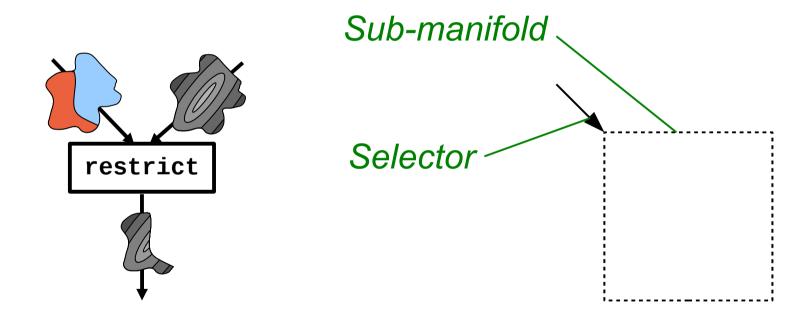
- Function Calls
 - Recursion
 - Anonymous Functions
 - Nested Functions
 - Closures
 - Higher-Order Functions
 - Function-valued Fields
 - Runtime Function Definition

Proto's Families of Primitives

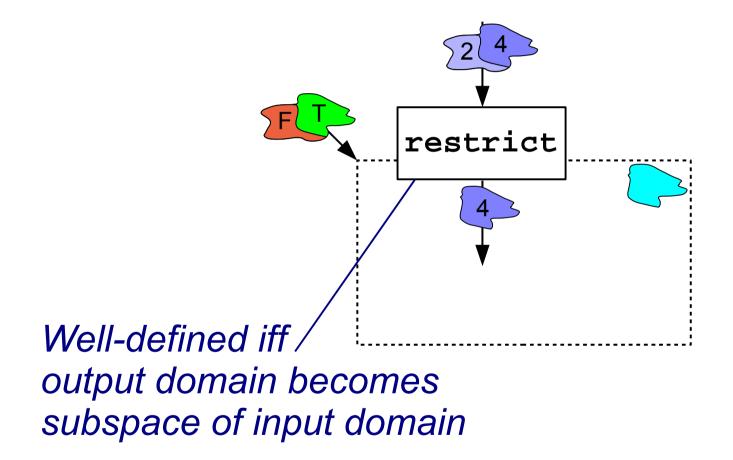




Focus: Restriction

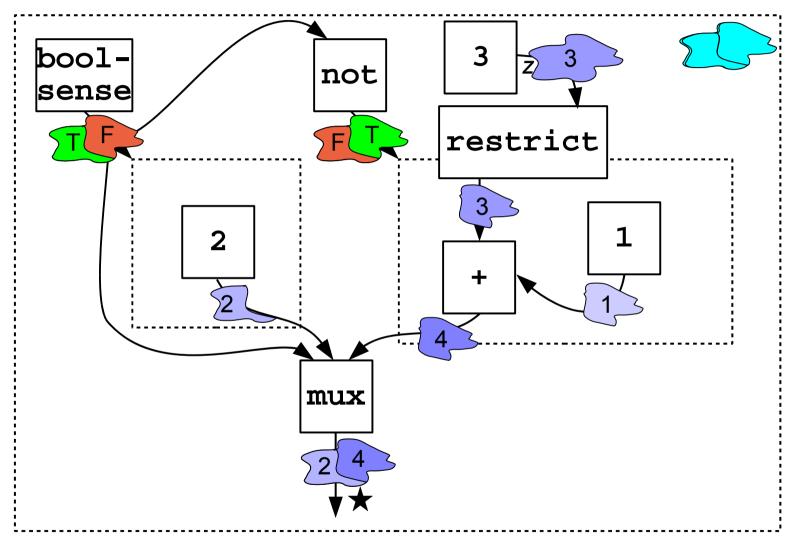


Changing Field Domains



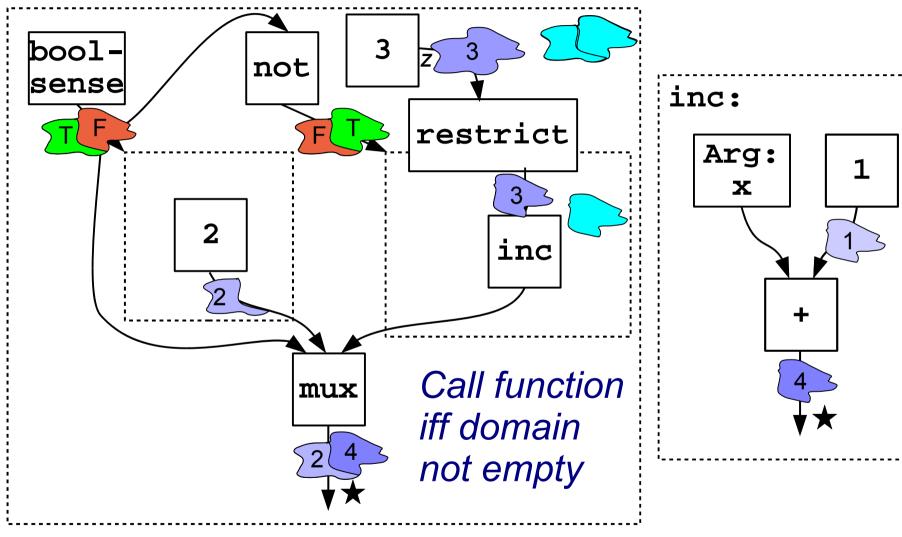
Operator implicit in all domain-crossing uses of field.

Example: Branching



```
(let ((z 3))
 (if (bool-sense) 2 (+ z 1)))
```

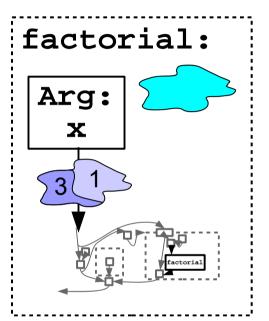
Branches and Function Calls

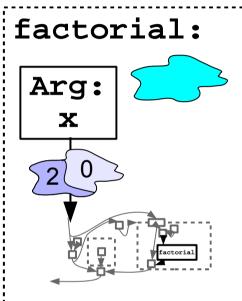


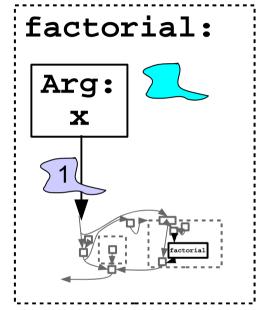
(let ((z 3)) (if (bool-sense) 2 (inc z)))

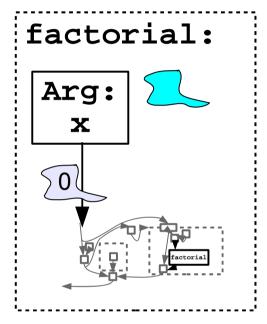
Manifold restriction → well-defined distributed function calls

Distributed Recursion









First Class Functions Checklist

- Function Calls
- Recursion
 - Anonymous Functions
 - Nested Functions
 - Closures
 - Higher-Order Functions
 - Function-valued Fields
 - Runtime Function Definition

Anonymous & Nested Functions

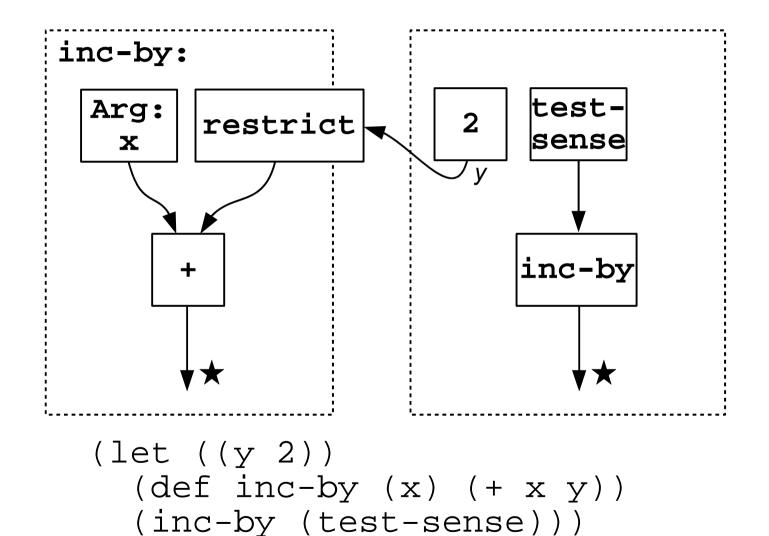
```
(let ((z 3))
                (if (bool-sense)
                    (inc z)))
                          (let ((z 3))
(let ((z 3))
                            (if (bool-sense)
  (def inc (x) (+ x 1))
  (if (bool-sense)
                                ((fun (x) (+ x 1))
                                 z)))
      (inc z)))
```

purely syntactic solution...

First Class Functions Checklist

- Function Calls
- Recursion
- Anonymous Functions
- Nested Functions
 - Closures
 - Higher-Order Functions
 - Function-valued Fields
 - Runtime Function Definition

Closures via Restriction



External references pass through restrict Well-definedness moves to evaluation...

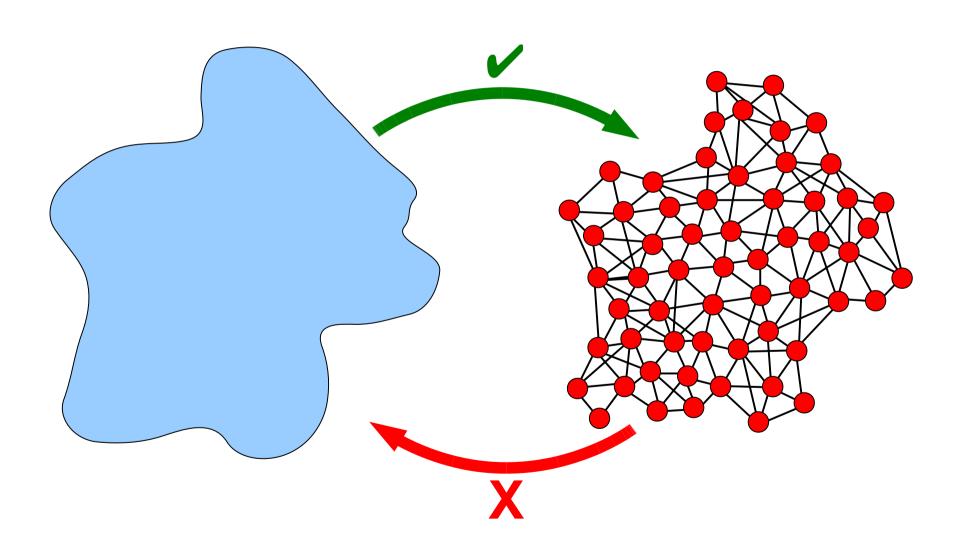
First Class Functions Checklist

- Function Calls
- Recursion
- Anonymous Functions This is as far as we're
- Nested Functions
- Partial Closures
 - Higher-Order Functions
 - Function-valued Fields
 - Runtime Function Definition

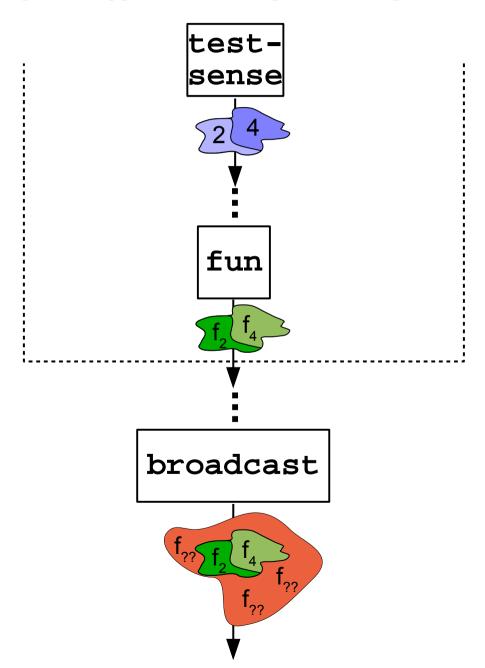
This is as far as we're confident...

Notice that so far everything is handled at compile time

Aggregate vs. Local Definition



Problem #1: Well-Definedness



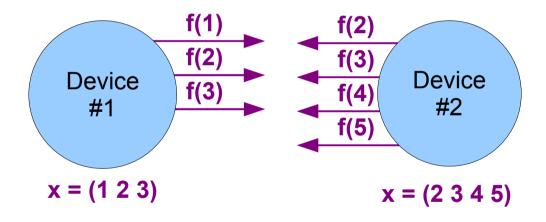
How can field domains within a function be expanded?

Problem #2: Function Equality

```
(if (bool-sense)
  (fun (x) (any-hood (nbr x)))
  (fun (x) (all-hood (nbr x)))
                                  When should
(if (bool-sense)
  (fun (x) (any-hood (nbr x))) local function
  (fun (x) (any-hood (nbr x)))) instances share
                                  their state?
(let ((x (test-sense)))
  (if (bool-sense)
    (fun () (any-hood (nbr x)))
    (fun () (any-hood (nbr x))))
(let ((f (fun (x) (any-hood (nbr x))))
  (if (bool-sense) f f))
```

Problem #2: Function Equality

```
(let ((x (vector-sense)))
  (map f x))
```



When should local function instances share their state?

An unsatisfactory syntactic solution:

```
(procs (elt sources)
      ((var init evolve) ...)
    (same? run? &optional terminate?)
          body)
```

This definition permits some intriguing possibilities, like functions with domains that self-overlap, but it's awkward.

There's got to be a more elegant way...

Conclusions & Invitation

- Aggregate programs need first-class functions.
- Restriction models enable function properties that can be resolved at compile-time.
- Runtime properties require expanding domains and testing function equality.

We need more elegant ways of doing this!

Get involved: http://proto.bbn.com

Acknowledgements

Raytheon BBN Technologies

- Kyle Usbeck
- Brett Benyo

External Collaborators:

- Mirko Viroli
- Stefan Dulman
- Nikolaus Correll

