On the Evaluation of Space-Time Functions

Jacob Beal, Kyle Usbeck

Spatial Computing Workshop @ SASO 2011
Distributed function calls are hard!

- Action at a distance
- Recursion
- Function equivalence
Distributed function calls are hard!

- Action at a distance
- Recursion
- Function equivalence

\[ f(Device_{#6}, #10, #11) \]
Distributed function calls are hard!

- Action at a distance
- Recursion
- Function equivalence
Distributed function calls are hard!

- Action at a distance
- Recursion
- Function equivalence
  - Worse w. 1st class fns

[c.f. Beal, 2009]

Most “distributed” calls avoid the last two…
Approach: Continuous Model

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

Approximate with:
- Discrete network of devices
- Signals transmit state

Advantages: simple, scalable, robust, adaptive...
(def gradient (src) ...)
(def distance (src dst) ...)
(def dilate (src n)
  (<= (gradient src) n))
(def channel (src dst width)
  (let* ((d (distance src dst))
         (trail (<= (+ (gradient src)
              (gradient dst))
                 d)))
   (dilate trail width)))

Proto

evaluation

global to local compilation

platform
specificty &
optimization

discrete
approximation

Device
Kernel

Global
Local
Discrete

[Beal & Bachrach, '06]
Proto's Families of Primitives

Pointwise

Feedback

delay

+ 48

Restriction

Neighborhood

restrict

nbr

any-hood
Continuous Space-Time Programs

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

(+ 1 (test-sense))
Functions: Substitution Model

\[
\text{inc:} \quad \text{Arg: } x \quad 1
\]

\[
\text{inc:} \quad \text{Arg: } x \quad 1
\]

\[
(\text{def inc (x) (+ x 1))}
\]

\[
(\text{inc (test-sense))}
\]

Problems: code bloat, can't compile recursion or 1st class fns
Functions: Call In-Place Model

\[(\text{def inc} \ (x) \ (+ \ x \ 1))\]

\[(\text{inc} \ (\text{test}\text{-sense}))\]

Better: allows “normal” compiled function calls
Recursion: Substitution vs. Call-in-Place

(factorial:

Arg: x

(= x 0)

1

(* x (factorial (- x 1))))

To make call-in-place work, we'll need to be able to branch...
Proto's Families of Primitives

Pointwise

Restriction

Feedback

Neighborhood

delay

+ 41

+ 7

+ 48

restrict

nbr

any-hood
Different Spaces Interacting

Sub-manifold

Selector
Different Spaces Interacting

(let ((x (bool-sense)))
  (+ (restrict x 2)
      (restrict (not x) 3)))

Not well-defined!
Syntactic safety: if & mux

Well-defined iff selected inputs cover output.

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

Not well-defined!
Changing Field Domains

Well-defined iff output domain becomes subspace of input domain
(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!
(let ((y 2))
  (def inc-by (x) (+ x y))
  (inc-by (test-sense)))

Problem: what if the function call was in an if?
(let ((y 2))
  (def inc-by (x) (+ x y))
  (inc-by (test-sense)))

Solution: external references pass through restrict
Distributed Recursion

```lisp
(defun factorial (x)
  (if (= x 0)
      1
      (* x (factorial (- x 1))))
)

Solution: external references pass through restrict
Proto's Families of Primitives

Pointwise

Feedback

delay

Restriction

Neighborhood

restrict

nbr

any-hood
Branches → Feedback Operators

Combination ensures well-defined

\[
\text{(rep } x \ 0 \ (+ \ x \ (dt)))
\]
Proto's Families of Primitives

Pointwise

Feedback

delay

+ 41 7

+ 48

Restriction

Neighborhood

nbr

any-hood
(max-hood (+ (nbr (test-sense)) 3))
Restricting Neighborhoods

Neighborhood must change domain too!

Same problem as for restriction without if construct

Solution: compile-time error checking
Implementation in Proto

- Previously, all function calls were inlined!
- Upgrades:
  - Function call opcodes added to VM
  - External references → implicit arguments
  - Error checking for bad nbr / if interactions
Distributed function call problems:

- Action at a distance ✔️
- Recursion ✔️
- Function equivalence ✔️
  - $1^{st}$ class fns (partly implemented)
  - (except part of $1^{st}$ class)
Contributions & Future Work

- Evaluation model for space-time function calls
- Analysis of Proto operator interactions
- Implementation of call-in-place model in Proto

- Future work: 1st-class fns, implement recursion

http://proto.bbn.com/