Propagation Networks: A Flexible and Expressive Substrate for Computation

Alexey Andreyevich Radul

August 4th, 2009, MIT CSAIL

Supervised by Gerald Jay Sussman
I Contribute an

- Expressive prototype of
- General-purpose propagation by
- Generic merging, which is a new
- Module boundary
We are Restricted by Sequential Time

Step 1: Freeflowing Time by Propagation

Step 2: ???

Step 3: Profit
We are Restricted by Sequential Time

Step 1: Freeflowing Time by Propagation

Step 2: Generality by Generic Merging

Step 3: Profit: Coexistence and Composition
We are Restricted by Sequential Time

Step 1: Freeflowing Time by Propagation

Step 2: Generality by Generic Merging

Step 3: Profit: Coexistence and Composition
<table>
<thead>
<tr>
<th>we have</th>
<th>we want</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sequential</strong></td>
<td><strong>Freeflowing</strong></td>
</tr>
<tr>
<td><strong>Time</strong></td>
<td><strong>Time</strong></td>
</tr>
<tr>
<td><strong>Complete</strong></td>
<td><strong>Works in</strong></td>
</tr>
<tr>
<td><strong>Answers</strong></td>
<td><strong>Progress</strong></td>
</tr>
</tbody>
</table>
\[(\text{let } ((x (+ y z))) (\ast (+ x 1) (- x 1))))\]

**Sequential Time, Complete Answers**
(let ((x (+ y z)))
  (* (+ x 1) (- x 1)))

Sequential Time, Complete Answers
(let ((x (+ y z)))
  (* (+ x 1) (- x 1)))

Sequential Time, Complete Answers
(let ((x (+ y z)))
  (* (+ x 1) (- x 1)))

Sequential Time, Complete Answers
(let (((x (+ y z)))
   (* (+ x 1) (- x 1)))

Sequential Time, Complete Answers
(let (((x (+ y z)))
      (* (+ x 1) (- x 1))))

Sequential Time, Complete Answers
(let ((x (+ y z)))
  (* (+ x 1) (- x 1)))

Sequential Time, Complete Answers
(let ((x (+ y z)))
  (* (+ x 1) (- x 1))))
<table>
<thead>
<tr>
<th>we have</th>
<th>we want</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential Time</td>
<td>Freeflowing Time</td>
</tr>
<tr>
<td>Complete Answers</td>
<td>Works in Progress</td>
</tr>
</tbody>
</table>
Freeflowing Time, Works in Progress
Freeflowing Time, Works in Progress
Freeflowing Time, Works in Progress
Freeflowing Time, Works in Progress
Freeflowing Time, Works in Progress
Freeflowing Time, Works in Progress
Freeflowing Time, Works in Progress
Freeflowing Time, Works in Progress
Freeflowing Time, Works in Progress
Freeflowing Time, Works in Progress
Freeflowing Time, Works in Progress
Freeflowing Time, Works in Progress
Freeflowing Time, Works in Progress
Freeflowing Time, Works in Progress
We are Restricted by Sequential Time

Step 1: Freeflowing Time by Propagation

Step 2: Generality by Generic Merging

Step 3: Profit: Coexistence and Composition
A propagator is a machine that reads some cells and can write to some cells always on, asynchronous, stateless
A propagator is a machine that reads some cells and can write to some cells always on, asynchronous, stateless
A propagator is a machine that reads some cells and can write to some cells always on, asynchronous, stateless.
A propagator is a machine that reads some cells and can write to some cells always on, asynchronous, stateless.
Network them, and values propagate
Network them, and values propagate
Network them, and values propagate
Network them, and values propagate
Network them, and values propagate
This much has been done many times before

Guy L. Steele Jr.
*The Definition and Implementation of a Computer Programming Language Based on Constraints.*

... 

Gregory H. Cooper.
*Integrating Dataflow Evaluation into a Practical Higher-Order Call-by-Value Language.*
Current State of the Art

- many systems
- all different
- all complicated
- not composable
My Prototype

- One system with
- many modules that
- reproduce previous work,
- coexist and
- compose
We are Restricted by Sequential Time

Step 1: Freeflowing Time by Propagation

Step 2: ???

Step 3: Profit
Normal code-data interface
Usual propagation code-data interface
More modular interface
We are Restricted by Sequential Time

Step 1: Freeflowing Time by Propagation

Step 2: Generality by Generic Merging

Step 3: Profit: Coexistence and Composition
A generic merge function

;;; The contract for a well-behaved mergeable
;;; knowledge representation is ...
;;; ... implement appropriate methods on the
;;; generic procedure MERGE ...

(define merge (make-generic-operator 2))
Merge each according to its kind

- Merge complete answers by equality testing
- Merge intervals by intersection
- Merge supported value lists by union
- Merge time-varying values by keeping the freshest
- Merge possibility sets by intersection
- Merge structures with holes by unification
- ...
Raw values merge by equality

(define (merge-raw old new)
  (if (equal? old new)
      old
      (error "Contradiction" old new)))

(set-default-operation! merge merge-raw)
Normal programming still works
Normal programming still works
Normal programming still works
Normal programming still works
Normal programming still works
Normal programming still works
Normal programming still works
Normal programming still works
We can compute through constraints
We can compute through constraints
We can compute through constraints
We can compute through constraints
We can compute through constraints
We can compute through constraints
We can compute through constraints
We can compute through constraints

(define (sum a b total)
  (adder a b total)
  (subtractor total a b)
  (subtractor total b a))
Intervals merge by intersection

(define (merge-intervals old new)
  (make-interval
   (max (interval-low old) (interval-low new))
   (min (interval-high old) (interval-high new))))

(defhandler merge merge-intervals interval? interval?)
We can do interval arithmetic
We can do interval arithmetic
We can do interval arithmetic
We can do interval arithmetic
We can do interval arithmetic
We can do interval arithmetic
We can do interval arithmetic
We can do interval arithmetic
We can do interval arithmetic
We can do interval arithmetic.
We can do interval arithmetic
We can do interval arithmetic
We can do interval arithmetic

(define (temperature f c k)
  (let ((f-32 (make-cell))
        (c*9 (make-cell)))
    (sum 32 f-32 f)
    (product f-32 5 c*9)
    (product c 9 c*9)
    (sum c 273.15 k)))
Truth Maintenance Systems merge by union

(define (merge-tms old new)
  (fold-left tms-assimilate-one
    old
    (tms-value-list new)))

(defhandler merge merge-tms tms? tms?)
Baker, Cooper, Fletcher, Miller, and Smith live on different floors of a building that has only five floors. Baker does not live on the top floor. Cooper does not live on the bottom floor. Fletcher does not live on either the top or the bottom floor. Miller lives on a higher floor than does Cooper. Smith does not live on a floor adjacent to Fletcher’s. Fletcher does not live on a floor adjacent to Cooper’s.
We get Hypothetical Worldviews

(define (multiple-dwelling)
  ;; ...
  (require (> miller cooper))
  (forbid (= 1 (abs (- fletcher cooper)))))
  ;; ...
  )
We get Hypothetical Worldviews
We get Hypothetical Worldviews
We get Hypothetical Worldviews
We get Hypothetical Worldviews
We get Hypothetical Worldviews
We get Hypothetical Worldviews
We get Hypothetical Worldviews
We get Hypothetical Worldviews
We get Hypothetical Worldviews and Dependency-Directed Search
We get Hypothetical Worldviews and Dependency-Directed Search
We get Hypothetical Worldviews and Dependency-Directed Search
We get Hypothetical Worldviews and Dependency-Directed Search
We get Hypothetical Worldviews and Dependency-Directed Search
We get Hypothetical Worldviews and Dependency-Directed Search
We get updating of stale signals
We get updating of stale signals
We get updating of stale signals
We get updating of stale signals
We get updating of stale signals
We get updating of stale signals
We get updating of stale signals
We get updating of stale signals
We get updating of stale signals
Concurrency is natural
Concurrency is natural
Concurrency is natural
Concurrency is natural
Concurrency is natural
Concurrency is natural
Concurrency is natural
Concurrency is natural
We are Restricted by Sequential Time

Step 1: Freeflowing Time by Propagation

Step 2: Generality by Generic Merging

Step 3: Profit: Coexistence and Composition
Coexistence
Composition
Composition
I Contribute an

- Expressive prototype of
- General-purpose propagation by
- Generic merging, which is a new
- Module boundary