Incremental Interactive Computation

Matthew Hammer
hammer@cs.umd.edu
Incremental Interactive Computation

Matthew Hammer
hammer@cs.umd.edu
Batch Computation vs Interactive Computation

**Batch Model**
- No time, no dialogue

**Interactive Model**
- Dialogue in time

Diagram:
- Input flow:
  - Batch Model: Input → C → Output
  - Interactive Model: U → C → U

Thursday, February 27, 2014
Interaction is a Dialogue

Time

Creating input
Demands Output
Mutates input
Demands Output

Interacting User

Reactive Computation

Time
Elements of Interactive Dialogue

• User and system interact across time
• User mutates / changes input structure
• User demands / observes output structure
• The system maintains a correspondence between input and output structures
• I/O correspondence is computational
Incremental Interactive Computation
Claim: Interesting interactive systems consist of incremental computations.
Example: Spreadsheets

Input Structure

Incremental Computation

Output Structure

Cell Formulae

Formula evaluation

Cell values
IC systems work by recording traces of computations and then reusing portions of those traces as inputs change. Unfortunately, prior IC approaches had two major limitations in trace reuse. First, because traditional IC imposes a total ordering on traces (typically, because reliance on the Dietz-Sleator order-maintenance data structure, several straightforward kinds of reuse are impossible. For example, consider using IC to implement a spreadsheet, so that visible formulae are minimally
<table>
<thead>
<tr>
<th>Input</th>
<th>Text file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>Parse tokens</td>
</tr>
<tr>
<td>Output</td>
<td>AST</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>AST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>Lexical scoping</td>
</tr>
<tr>
<td>Output</td>
<td>Def/use edges</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>AST + Def/use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>Type inference</td>
</tr>
<tr>
<td>Output</td>
<td>Type errors</td>
</tr>
</tbody>
</table>
Computing Incrementally

1. Input changes are gradual

Full re-computation is often **redundant**

2. Output observation is limited

Full re-computation is often **overly eager**
# Computing Incrementally

1. Input changes are gradual

2. Output observation is limited

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spreadsheet</td>
<td>Cells change slowly</td>
</tr>
<tr>
<td>Word processing</td>
<td>Document and dictionary both change slowly</td>
</tr>
<tr>
<td>Programming</td>
<td>Program changes slowly</td>
</tr>
</tbody>
</table>
**Computing Incrementally**

1. **Input changes are gradual**

2. **Output observation is limited**

<table>
<thead>
<tr>
<th>Example: Spreadsheet</th>
<th>One worksheet is active, others hidden</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example: Word processing</td>
<td>Viewport shows one or two pages</td>
</tr>
<tr>
<td>Example: Programming</td>
<td>Viewport shows one file, module or function</td>
</tr>
</tbody>
</table>
Adapton
Abstractions for Incremental Interaction

Current draft available here:
To appear at PLDI 2014!
Adapton Programming Abstractions

- **Mutable references:**
  - Hold changing input structure

- **Lazy thunks:**
  - Demand-driven computations
  - Output structure
Mutable references

Incremental Computation (thunks)

Demand-driven Outputs
Mutable references

Incremental Computation (thunks)

Demand-driven Outputs
Mutable references

Incremental Computation (thunks)

Demand-driven Outputs

Switch Demand:
Mutable references

Incremental Computation (thunks)

Demand-driven Outputs

Sharing

max 2

+ 5

min 3

max 7

+ 7

* 15

min 3

max 2

2

+ 5

3

+ 5

min 3

7

* 15

min 3
**Mutable references**

**Incremental Computation (thunks)**

**Demand-driven Outputs**

Input Change:

```
1 2 3 4
max 2 + 5 min 3
max 7 + 7 * 15 min 3
max 7 + 5
min 3
```

Thursday, February 27, 2014
Some previous results are affected
Mutable references

Incremental Computation (thunks)

Demand-driven Outputs

Demand new output:
Mutable references

Incremental Computation (thunks)

Demand-driven Outputs

Demand new output:
Lazy Structures

• **Spreadsheet example:**
  • Each thunk returns a single number

• **Lazy Lists:**
  • Each thunk returns `Nil` or `Cons`
  • `Cons` holds head value and a thunk tail
  • Laziness can be applied to trees, graphs, and essentially any other data structure

• **Inputs:** Special thunks are mutable
type 'a thunk = unit -> 'a

let force : 'a thunk -> 'a = fun t -> t ()

type 'a lzlist = [ | `Nil | `Cons of 'a * ('a lzlist thunk) ]

let rec from_list l =
  match l with
  | [] -> `Nil
  | h::t -> `Cons(h, fun() -> from_list t)
type 'a lzlist = [ `Nil | `Cons of 'a * ('a lzlist thunk) ]

let rec merge l1 l2 = function
 | l1, `Nil -> l1
 | `Nil, l2 -> l2
 | `Cons(h1,t1), `Cons(h2,t2) ->

   if h1 <= h2 then
   `Cons(h1, fun() -> merge (force t1) l2)

   else
   `Cons(h2, fun() -> merge l1 (force t2))
Mergesort Example

Input: \([3, 5, 8, 2, 1, 7]\)

Singletons: \([3, 5, 8, 2, 1, 7]\)

Merge #1: \([3, 5], [2, 8], [1, 7]\)

Merge #2: \([3, 5], [1, 2, 7, 8]\)

Merge #3: \([3, 5], [1, 2, 7, 8]\)

Merge #4: \([1, 2, 3, 5, 7, 8]\)

Flatten: \([1, 2, 3, 5, 7, 8]\)
Course Project
Project: Interactive Program Analysis

• **Assume:** Interactive “Structure Editor”
• **Programmer** manipulates AST directly
• **Learn:** Adapton IC framework
• **Build:** Incremental Program Analysis
• **Example:** Use/def information
• **Example:** Type Inference
• **Example:** Control-flow analysis
Background: Structure Editors

- Philosophical claims:
  - Programs consist of rich structure
  - Rich interaction exposes this structure

- Example prototypes:
  - Haskell -- [http://www.youtube.com/watch?v=v2ypDcUM06U](http://www.youtube.com/watch?v=v2ypDcUM06U)
  - Citris -- [http://www.youtube.com/watch?v=47UcOspbZ2k](http://www.youtube.com/watch?v=47UcOspbZ2k)
  - TouchDevelop -- [http://www.youtube.com/watch?v=a6GRg2glKpc](http://www.youtube.com/watch?v=a6GRg2glKpc)