Modelling State in Mind and Machine

Meurig Beynon

Department of Computer Science
University of Warwick
Coventry CV4 7AL

Acknowledgments

Steve Russ
Simon Yung, Richard Cartwright, Patrick Sun
Amanda Wright

http://www.dcs.warwick.ac.uk/pub/research/modelling

Background

History of the Empirical Modelling (EM) project

state in programming - procedural vs declarative

... motivating idea behind ARCA (1982)
"a definitive (definition-based) notation for graphics"

programming paradigm?
-> HCI for modelling and design (1985)
-> concurrent systems modelling (1986)

Representation of state by metaphor via artefact

Definitive script :
variables represent **observables**
definitions represent **dependencies**

observation-oriented analysis of **agency**, **dependency**, **state**

Evolution of the **observable**

explicitly observed: sensory data
corner of the room / table
scientific observable
voltage / current
conceptual observable
cricket match is won

Conceptual observable is paradoxical?

cf. classic empiricism vs Radical Empiricism

William James' (1910) *Essays in Radical Empiricism*

'Pure Experience'

procedural and experiential vs. declarative and logical knowledge representation

representation via artefact rather than symbol

Programming in the EM context

Programming (esp. for e.g. reactive systems)
is a form of concurrent systems modelling

programmer / user / computer / peripherals as agents

EM as *prior* to conventional programming
empirical knowledge that informs discovery of programs
exploratory modelling prior to commitment
to explicit control patterns
emphasis on programmer/user perception of state:
super agency / uncircumscribed behaviour

EM develops "family" of models from which many 'programs' can be extracted

default execution is semi-automatic, in part driven by intelligent human super-agent

abstract data structures as conceptual observables
representing cognitive processes prior to program construction

Heapsort from an EM Perspective

Three stages in modelling the heapsort process:

(1) constructing a visual model of a heap with which the user can experiment in order to understand the heap concept;

(2) constructing state-based models to represent the stages in the heapsort process, allowing the user to trace the steps involved in heap-building and sort extraction through a sequence of manual operations;

(3) introducing automatic mechanisms to carry out the appropriate sequence of steps.

... follow layers of understanding associated with learning heapsort principles ....

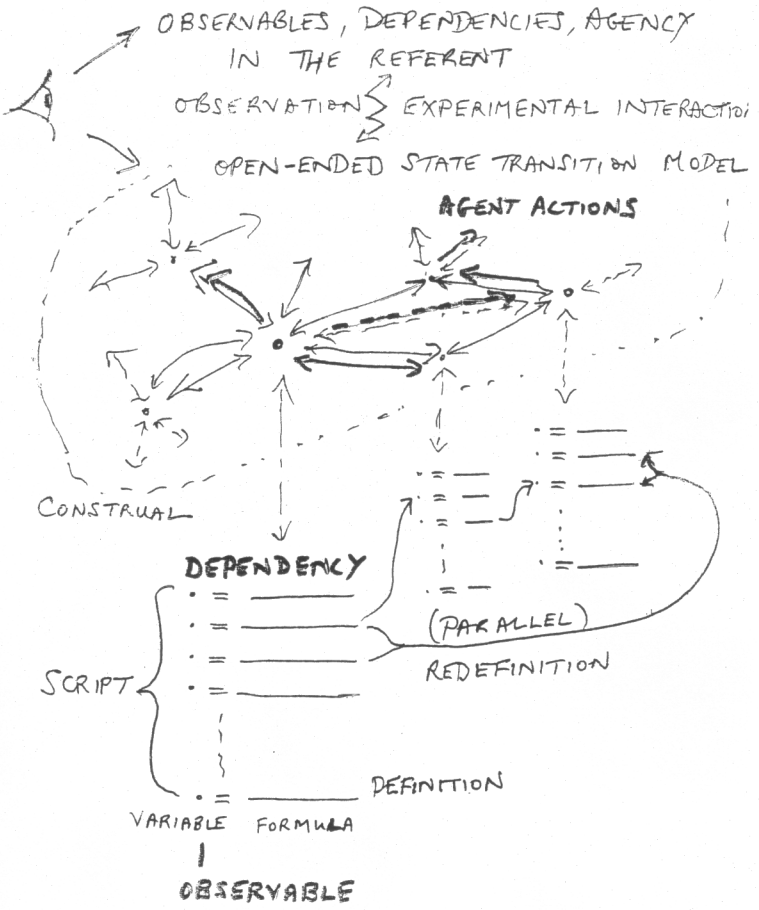
Constructing a Visual Model of a Heap on 7 Elements

Establishing the Heap Condition: Control and Agency

Heap extension: Associating Tree Segments with Array Intervals

Constructing State-based Models of the Heapsorting Process

SITUATED MODELLING



- VALUE OF VARIABLE IS 'DIRECTLY APPREHENDED'
- CHANGE IN VALUES INDIVISIBLY LINKED BY DEFN

HEAPSORT 2

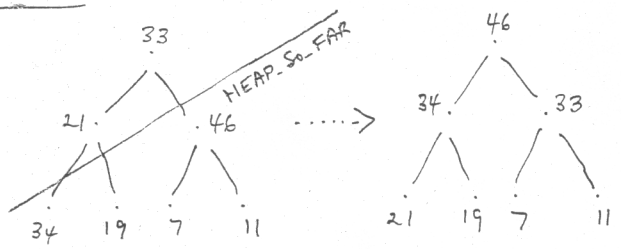
(*)

? HC-AT-NODE$_N$

→ EXCHANGE-VALS (N, IX-GC(N))

HUMAN AGENT PERFORMS "SUPER-AGENT"

SUBSEQUENTLY INTRODUCE AUTOMATIC AGENT

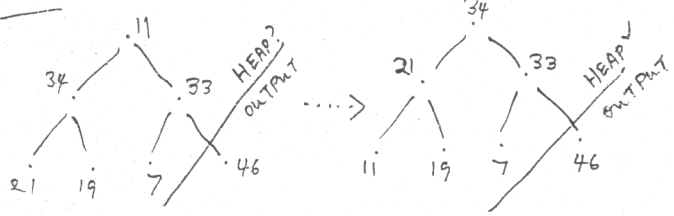


HEAP BUILD

FIRST-OF-HEAP INITIALLY 4

DO FIRST-OF-HEAP --; (*)

UNTIL FIRST-OF-HEAP = 1

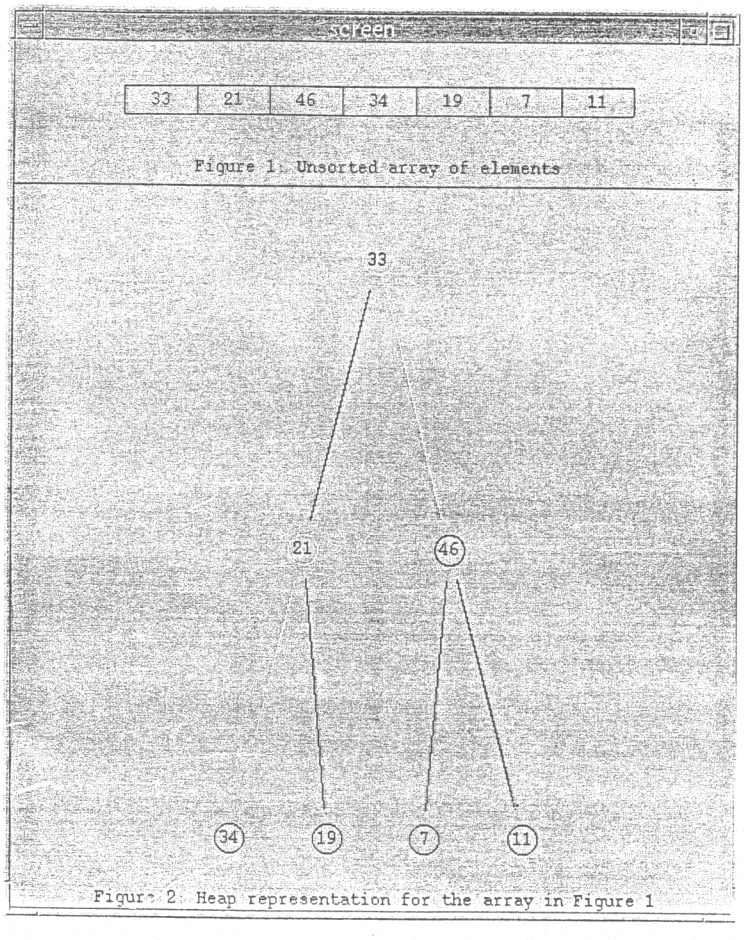


OUTPUT SORT

LAST-OF-HEAP INITIALLY 7

DO EXCHANGE-VALS (FOR.LOW), L-O-H--

HEAPSORT 3



HEAPSORT 4

HEAPSORT 5

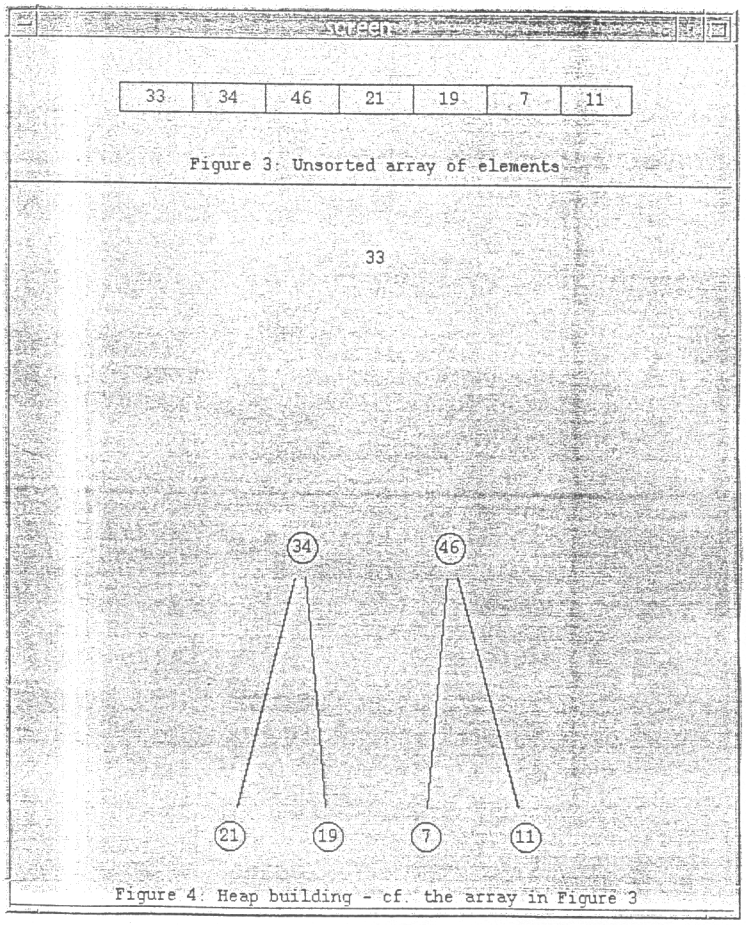


Figure 3: Unsorted array of elements

Figure 4: Heap building - cf. the array in Figure 3

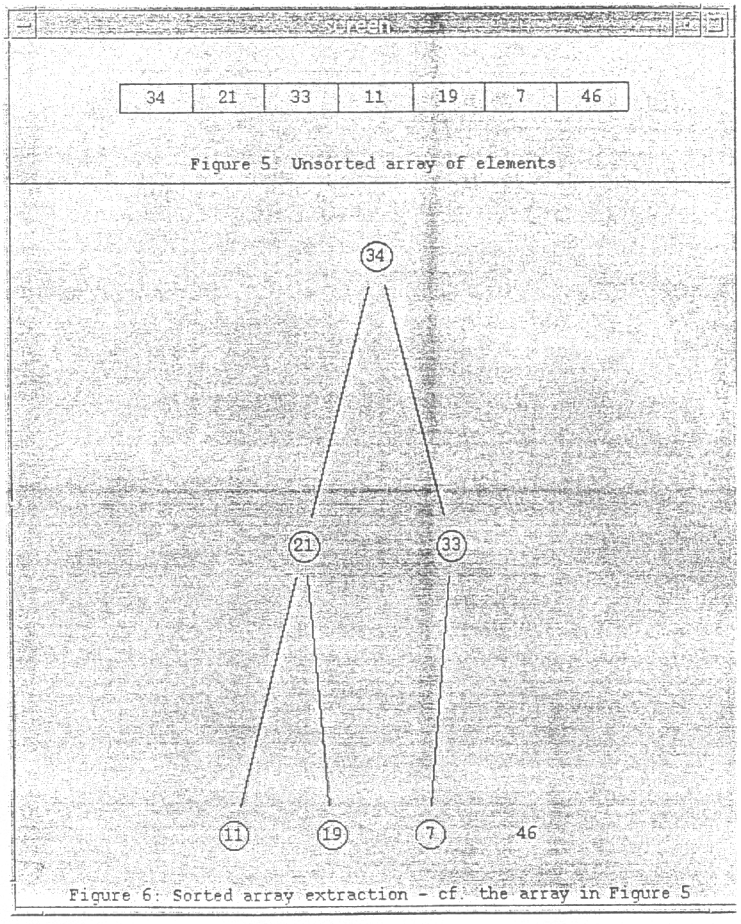


Figure 5: Unsorted array of elements

Figure 6: Sorted array extraction - cf. the array in Figure 5

5

```
%eden                    "an evaluator for definitive notations"

val = [33,21,46,34,19,7,11];

harray is [node1, node2, node3, node4, node5, node6, node7];
htree is proj2(harray);

val1 is val[1];                    // define individual values
val2 is val[2];
...

node1 is [1, val1];                // attach values to nodes
node2 is [2, val2];
...

ord12 is cf(1,2,htree);            // define order relation
ord13 is cf(1,3,htree);
...

// user-defined functions used in definitions above

func cf {
    para i , j, arr; auto result;
    result = (arr[i] > arr[j]) ? 1 : ( (arr[i] < arr[j]) ? -1 : 0 );
    return result;
}

func proj2 {
    para lx; auto result;
    result = [ ];
    while ( lx != [ ] ) {
        result = result // [ lx[1][2] ]; shift lx;
    };
    return result;
}
```

6

```
%scout          // a definitive notation for screen layout

window heapwin = {
        type: DONALD
        box: [{0 , 100}, {500 , 600}]
        pict: "heapview"
        bgcolor: "grey"
        border: 0
        sensitive: ON
};

display screen = <heapwin / ... >;

%donald         // a definitive notation for line drawing

label lab7, lab6, lab5, lab4, lab3, lab2, lab1
line l37, l36, l25, l24, l13, l12
point p7, p6, p5, p4, p3, p2, p1

l12 = [p1, p2]              // define the edges of the tree
l13 = [p1, p3]
...

p1 = {500 , 900}            // locate the nodes of the tree
p2 = {400 , 500}
...

%eden           // define labels of points and attributes of lines

A_l12 is "color=" // (ord12>=0) ?"black": "white");
A_l13 is "color=" // (ord13>=0) ?"black": "white");
...

_lab1 is label(str(val1), _p1);
_lab2 is label(str(val2), _p2);
```

```
%eden          // modelling the 7 element heap

first = 1;
last = 7;

hc1 is (ord12 >= 0) && (ord13 >= 0);
hc2 is (ord24 >= 0) && (ord25 >= 0);
...

ixgtch1 is (val2 > val3) ? 2 : 3;
ixgtch2 is (val4 > val5) ? 4 : 5;
....

A_l12 is "color=" // (ord12 >= 0) ? "black" : "white"; ...
```

```
%eden          // modelling a heap on range <first, last>

func inheap {
        para i, f, l;
        return (f <= i) && (i <= l);
}

inhp1 is inheap(1, first, last);
...

hc1 is (!inhp1) || (inhp1 && ( !inhp2 || (inhp2 && ord12 >= 0) ) ... );
...

ixgtch1 is (!inhp3) ? 2: ( (val2 > val3) ? 2 : 3 );
....

A_l12 is  "color=" // (inhp1 && inhp2) ?
        ( (ord12 >= 0) ? "black" : "white" ) : "grey" ); ...
```


Summary

- not one model
  - adaptable to many purposes
  - incorporating many experimental environments
  - wide range and diversity of control strategies
  - highlighting behavioural modes of observation

- in timately connected with learning and discovery of heapsort

- generated by distributed client-server version of interpreter
  - study in collaborative development
  - teaching environment for EM and heapsort principles

- pedagogical, but potentially adaptable for
  - extraction of conventional implementation by translation
  - DAM implementatioin