Rethinking programming

Are envisaging a conception of programming that embraces such issues as

- learning
- design
- reinterpretation
- accident
- discovery
- ...

Discuss heapsort with reference to this theme ...

A new perspective on programming ...

Will begin by demonstrating a simple animation of heapsort on 7 nodes ...

What is remarkable or interesting about the heapsort animation model?

- for the computer scientist, the visualisation is of trivial size whilst heapsort is "only" interesting for large datasets
- for the educational psychologist, the limitations of such animations are recognised

Relevant questions are:

- What is involved in conceiving an algorithm such as heapsort?
- What does a human agent need to appreciate in order to perform heapsort?
- How is it that heapsort is a process that is amenable to efficient execution by a computer?
- What assumptions about agency and context surround the process of heapsorting?
- In what way can one model that animates heapsort give more insight to the learner than another?

To appreciate the unusual nature of the model, will look more closely at its construction ...

About the model

The model takes the form of

- a *definitive script* that includes 'all' the observables associated with human interpretation and machine implementation of heapsort together with
- an open *input window* through which definitions can be added to the script, and any definition in the script can be revised
- *automated actions* that can be primed to respond conditionally to changes to specific observables

The script, the input window and the actions are respectively associated with the context for agent interaction, a/the human agent and programmed/automated agents

Interaction with the model

Pieces of the script can be freely extracted, definitions and automated actions introduced, revised and removed at will, and are appropriate subject to whether they admit interpretation by the modeller (or other human agent)

There is a useful analogy to be made between being introduced to a model and being taken on a guided walk. The model presents itself in a state, but it can be entered in the neighbourhood of many different states. In any of these states, there may be many alternative local variants, but there are also many interactions that do not form part of the walk. The model-building activity, inspection and exploration resembles 'walking around the neighbourhood of heapsort'.

Interaction 1

Loading part of the script makes it possible to examine the relationship between the array and the tree representation. Interaction with the model reveals:

- the analogue nature of the model of the data structure - cf. *the script that defines how changes to the key observables are linked in change* with a declarative or procedural *specification of the updating mechanism*
- that the focus is on 'what dependency is latent?' not on 'the algorithm/mechanism that maintains dependency'
- the relationships that are latent in interaction, such as expose the correspondence between the array locations and the tree nodes, and that between segments of the array and subsets of the tree.

These are of course examples of "meaningful interaction". Harder to interpret is a redefinition such as defines the value of the last element of the array to be the square of the first.

What connects the model with the vicinity of heapsort?

Primarily, those observables that we need in order to describe heapsort. These include:

- the values in the array
- the locations of the nodes in the tree
- the notion of a node being in the active segment of the array
- the notion of the heap condition being satisfied at a node
- the order relationship between the value at a node and the value at its parent node
- what we are deeming to be the segment of current interest in the array

All these observables have explicit counterparts in the model that can be inspected at all times. The dependencies perceived to interrelate these observables are expressed by definitions.

It is also significant that e.g. order relationships between nodes that are unremarked in heapsort are *not* represented in the script

---

A useful impression of the range of observables introduced in the heapsort model is gained by looking at the observables and dependencies associated with a typical node - use the Dependency Modelling Tool for this.

The model development

To develop a guided walk is simply to undertake the walk yourself as if in the role of the guide.The walk is established in the mind of the guide, and realised at the guide's discretion. In the initial engagement with a walk, less subtle observation is involved; richer more sophisticated observations evolve later, as more experience of the walk is acquired.

The development of the heapsort model is analogous to developing a guided walk.

It is sometimes appropriate to return to previously visited states and gain familiarity with a particular pattern of transitions, or venture alternative approaches. Can illustrate this by resetting the array to specified values, for instance.

It is also possible to revert to previous versions of the script by over-writing current definitions with previous definitions.

To illustrate this, can introduce a file that restores more primitive definitions for the heap conditions (hc1, hc2 etc) and the index of the greater child at a node (ixgtch1 - 'the IndeX of the GreaTer CHild at node 1'). Specifically, these conditions are considered with reference to the entire array rather than a proper segment.

Significant experimental contexts

An important feature of model development (cf. walk development) is experimenting in specific contexts with a view to better understanding. One of the basic experiments underlying the heapsort method involves finding a systematic way of building a heap by exchanging parent-child elements. This is best explored initially in the context of the entire array.

Associating atomicity to agent action is a crucial feature of *observation-oriented* modelling. (Re)definitions are associated with atomic actions. In heapsort, exchanging a pair of elements can be deemed to be atomic (cf. rotating an edge of the tree through 180 degrees). Dependency mediates its impact indivisibly to all contingent observables, such as the ixgtch*, hc* etc.

What agency characterises 'understanding heapsort'?

There is a distinction between being able to type the appropriate definition into the input window, and (e.g.) being able to identify the most appropriate node at which to apply an exchange operation. This can be explored by introducing automated actions ("agents") that respond to mouse clicks in the vicinity of a node.

The relationship between the precise form that agency takes and domain understanding is further illustrated by:

- removing the colour coding of the nodes that satisfy the heap condition
- changing the nearness criterion for selecting a specific node

Such experiments illustrate the complexity of the interaction between the heapsort model and the human agent interacting with it, and the relevance of both human skill and the qualities of the technology. The model can be seen in this way as resembling an *instrument* rather than a tool.

Certain observables, such as 'being halfway around the walk' or 'being on the way to the summit', are only meaningful on the understanding that we are taking the walk, not merely rambling at random.

Any constraint on interaction with the heapsort model is at the discretion of the human agent interacting with it. Manual and automatic interaction can be contrived to follow a standard reliable pattern.

Can demonstrate this by introducing more automated actions to follow the standard progression of actions involved in heapsort. In order to discriminate between the 'making the heap' and 'outputing the source' phases, the observational criterion for changing from one phase to the other must be recognised.

In this context, highly abstract observables such as 'the current phase' impact on the notion of when the array should be deemed to be sorted.

It is of interest that, even when the standard procedures of heapsort are being respected, the heapsort model exhibits characteristics unlike a traditional heapsort program.

How can such a model be interpreted ... ?

- Visualisation to illustrate the heapsorting process
- Model to expose the concepts and perceptions that are involved in understanding heapsort
- As a bridge between an experiential and a formal account of the heapsort algorithm ...

Possible ways to exploit EM for programming

- EM as a means to requirements specification in conventional programming (cf. heapsort)
- EM as a way to produce models from which conventional programs can be derived (cf. JaM and Richard Cartwright's experience at BBC)
- EM as a means to generate programs with special characteristics (e.g. where experiential elements are prominent, adaptability is essential, where emphasis is educational purpose etc)

Concluding thoughts

As an approach to programming:

- observation-oriented modelling can be viewed as revisiting the original aspiration of object-orientation – to realise "programming as modelling"
- it offers an alternative framework for integrating procedural and declarative ways of thinking but only in direct relation to modelling not programming

The engineer's assurances about artefacts are rooted in

- reasoning based on established theory
- empirical evidence and *ad hoc* observation particular to the context

From an observation-oriented perspective, computer programming has to be seen in the same light.

- the emphasis on *program as mathematical object* is a legacy of history
- the range of modern computing obliges us to give more emphasis to understanding that is particular to the program context

---