Chapter 6 – Exploratory learning and the EFL

6.0 Overview of the Chapter

In this chapter, we discuss three case studies that represent different ways of applying EM to educational technology. These case studies show: how EM can be used to support pre-articulate exploration in a private learning scenario; how pre-articulate and formal learning activities can be connected within a common exploratory learning environment; and how a learning environment can reinforce a learner's construal of a physical situation. Broadly speaking, the application of computers for learning can be classified into building models and using models. In EM, the distinction between model construction and model use is blurred. Many EM models exhibit qualities of both model construction and model use. This enables modellers to explore partial models and build on top of them. The case studies in this chapter have been selected to highlight how EM can be used in learning activities across the whole range of the EFL.

6.1 Integrating model use and model building

In this section, we consider the use of EM to support a range of learning activities within the EFL. We also show how EM modelling can blur the distinction between building and using models. As discussed in chapter 3, the use of computers for learning is broadly of two kinds. These are:

i) learning through model-building: this involves the construction of models to enhance our personal understanding of a domain. In chapters 2, 3 and 4 we discussed the EM approach to the construction of models from computational and educational perspectives. We concluded that model construction in EM has two advantages over writing conventional

programs. Firstly, EM supports the pre-articulate learning activities situated at the experiential end of the EFL; and secondly, it allows model construction to integrate pre-articulate and formal activities in a single modelling approach.

ii) learning through model-use: this involves the use of a pre-constructed learning environment in the form of (e.g.) instructionist software or a constructionist microworld. In chapter 5, we discussed how EM learning environments can support many different types of learning. The emphasis was on describing techniques for model development that offer flexibility to the developer and facilitate an enhanced learning experience.

The two perspectives outlined above relate to traditional perspectives on the use of computers for learning – users either write their own programs or use programs written by other people. However, in EM, the learner has complete discretion over the interactions they undertake with a model, and is free to add new definitions to a model or refine existing definitions. Therefore, in EM, learning environments can combine model building with model exploration. The two traditional perspectives outlined above define the extreme ends of a spectrum ranging from model building to model use, as represented by the restaurant model and the relational algebra tutor respectively. In this chapter, we discuss three further EM case studies representative of learning environments within which model building and model-use for learning are conflated in different ways. These models are: the Monotone Boolean Functions in 4 variables (MBF4); Heapsort; and the Robotic Simulation Environment (RSE) (see Figure 6.1). We shall first informally describe the roles played in each case study by model building and model use, then discuss some of those aspects that are most significant for learning with reference to the EFL.

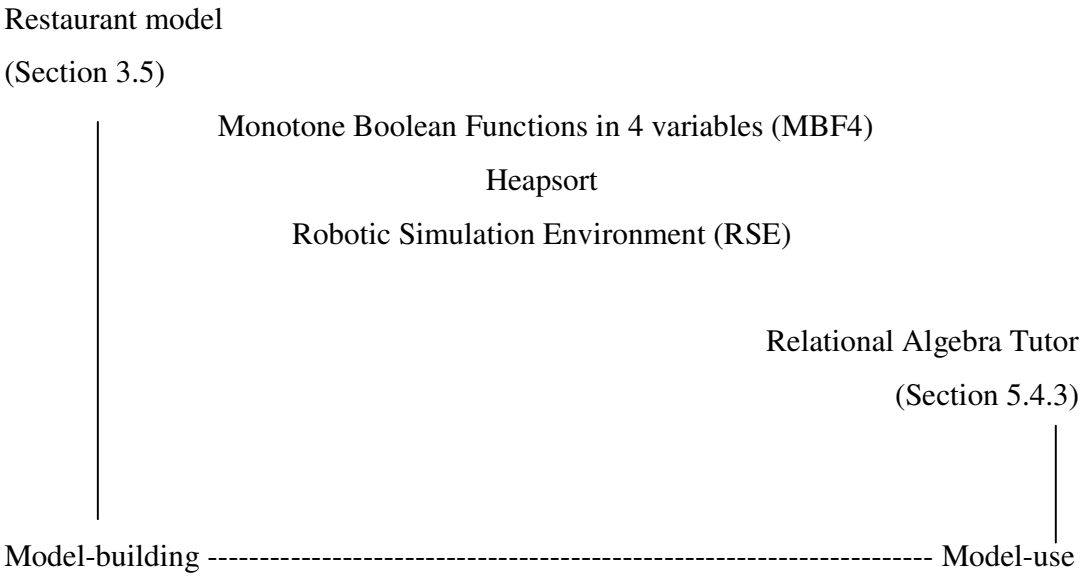


Figure 6.1 – Conflating model building and model use

The Monotone Boolean Functions in 4 variables (MBF4) model [EMRep, mbf4Beynon2003], is primarily the work of Meurig Beynon with some contribution from myself. This model was originally developed by Beynon as a private artefact [EMRep, fdl4Beynon2002] through which to enhance his comprehension of the relationships between several different abstract realisations of FDL4 ("learning through model-building"). There have been two aspects to my engagement with the model as a learner: I have interacted with the model to gain experience of unfamiliar mathematical concepts ("learning through model-use") and have extended the model in ways that make it more easily intelligible to the learner ("learning through model-building").

The Heapsort model [EMRep, heapsortBeynon1998] was constructed by Meurig Beynon, Amanda Wright and Jaratsri Rungrattanaubol and has been discussed in [Bey98, BRS+98, BRS00, Run02]. The model has a particular pedagogical aim, namely to support the exposition of the heapsort algorithm, but it has been constructed in stages in such a way that learners can carry out experiments and extensions. Learning therefore combines elements of model building and model use.

Rather than focusing exclusively on the heapsort algorithm, the model promotes exploration of the concepts around heapsort through interaction outside the bounds of heapsort. Unconstrained exploration can aid learning about heapsort by challenging and reinforcing the construal of the learner.

The Robotic Simulation Environment (RSE) [EMRep, rseRoe2003] is an example of a learning environment that aims to promote conceptual understanding of a real-world situation. I have constructed the model on the basis of a preliminary design developed in collaboration with researchers from the Kids' Club in Joensuu, Finland, as documented in [EJR+02]. In the Kids' Club, the children use the LEGO Mindstorms robot programming environment to program robot behaviours. The RSE is intended to supplement the current system. It provides an exploratory environment for the investigation of robot behaviour through a computer model. The RSE is to be regarded as a case study in model use for learning because the environment contains all the necessary elements for exploratory investigation of the robots. In keeping with a constructionist approach, the environment supports many different styles of interaction and allows learners to engage with the problems of robot programming at many levels.

6.2 Monotone Boolean Functions in 4 variables

The EM model of Monotone Boolean Functions in 4 variables (MBF4) illustrates Wilenski's observation that:

> '[t]he more connections we make between an object and other objects the more concrete (familiar) it becomes for us. The richer the set of representations of the object, the more ways we have of interacting with it, the more concrete it is for us' [Wil93].

The MBF4 model [EMRep, mbf4Beynon2003] has been composed by combining resources created by a number of different people. These include Bibi Hussain's EM model of the free distributive lattice on 3 elements [EMRep, fdl3Hussain2001], Allan Wong's EM model of the group of symmetries of a cube [EMRep,

symcubeWong2001] and John Buckle's diagrammatic representation of the Hasse diagram for the Free Distributive Lattice on 4 generators (FDL4) [Buc90]. The relevant parts of the EM models have been brought into a single environment and, using dependency, links have been created between them to establish the appropriate connections. The three main components of the MBF4 model are depicted in Figure 6.2.

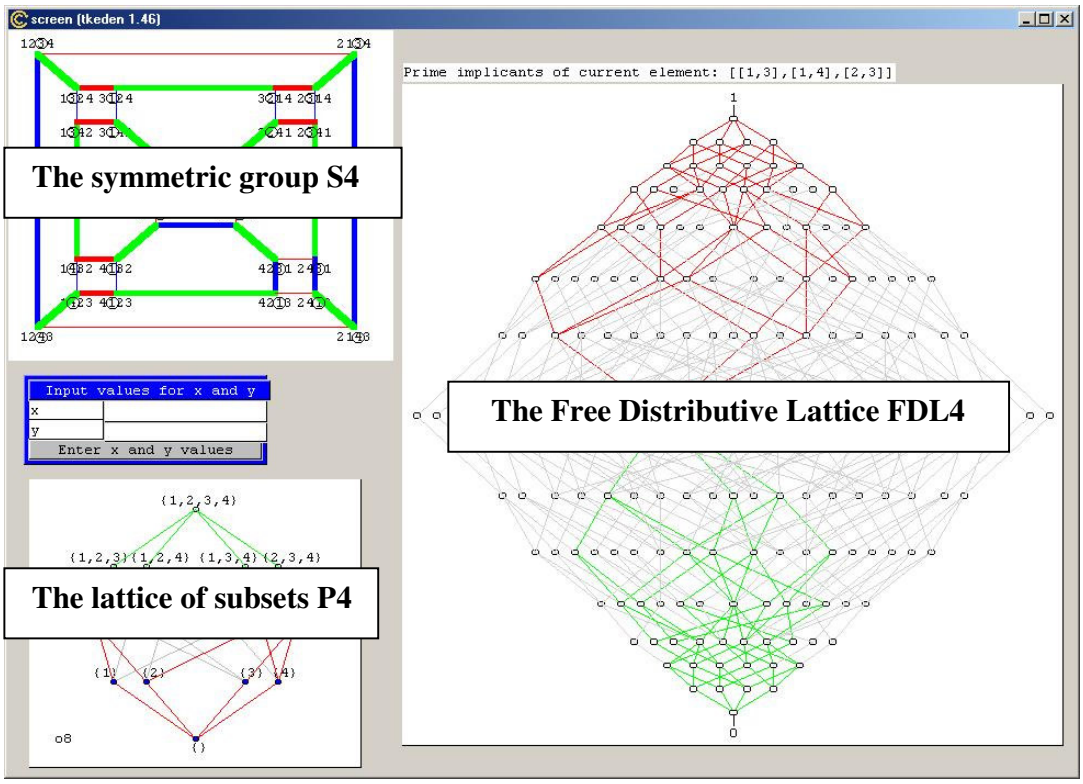


Figure 6.2 – The diverse components of the MBF4 model

The original prototype for the MBF4 model (see [EMRep, fdl4Beynon2002]) was developed by Beynon as a personal visual aid to mathematical research, and its interface was not designed for use by novices. The visual representations of the lattices P4 and FDL4 in this model are Hasse diagrams [Bir95] with 16 and 166 nodes respectively, and S4 is represented by a Cayley diagram in which the edges correspond to the basic transpositions (12), (23) and (34).

The above description of the model components is appropriate for the mathematical expert – it presumes some advanced knowledge of mathematical concepts and terminology. Our purpose in this section is to describe how interaction with the original MBF4 model – involving both model use and model building – can assist the novice in gaining the understanding of the mathematical objects involved. In this section, writing as a novice lattice theorist, I give an account of the types of activities that contributed to my learning, and the partial comprehension of the abstract mathematics represented in the model that emerged during the interaction. In the process of learning, I have also enhanced the model as a learning artefact.

For a learner, the principal role of the MBF4 artefact is to provide concrete support for coming to understand the concepts behind the artefact. Interaction and experimentation with the artefact can provide a stable source of experience on which to base an understanding of the lattice theoretic concepts in the model. The primary objective of the learner is to utilise the experience gained from interacting with the MBF4 artefact in order to understand the formal mathematics it embodies. The central importance of concrete intuitions as a foundation for formal mathematics is something that I have observed in my experience of teaching mathematics to undergraduates (cf. the emphasis placed on concrete understanding by Papert and others [TP91, Pap93, Wil93]). For instance, it would be absurd to expect a learner to reason symbolically about decreasing subsets if they did not have some experiential support for grasping the concept. The role of the interactive artefact is to provide a concrete means by which to attain the appropriate construal. In the following discussion, we outline the abstract mathematics underlying the MBF4 model, and explain the role of the model in helping a learner gain concrete experience of abstract structures through active experimentation. Each of the mathematical components in Figure 6.2, (P4, FDL4 and S4) exists as an EM model. We shall discuss each of these in isolation before illustrating how their connections can be explored.

6.2.1 P4: The lattice of subsets of {1,2,3,4} ordered by inclusion

The EM model of P4 can be used to illustrate some basic concepts of lattice theory, including decreasing subsets, partially ordered sets and lattices. Figure 6.3(a), which has been extracted from a screenshot of the executing P4 model, shows all the subsets of the set {1,2,3,4}. These are depicted using the conventional representation of a partially ordered set as a Hasse diagram. The role of the Hasse diagram is to provide concrete support for the abstract notion of a partial order on the subsets in P4. For instance, one subset contains another if there is an upward path in the Hasse diagram between the two. A mathematician might describe this visual relationship more formally as: the subset X contains the subset Y if and only if there is an upward path in the Hasse diagram from the node x that represents X to the node y that represents Y. This attempt to give a more precise and abstract description of a visual experience does not convey the immediate and concrete way in which the Hasse diagram is apprehended by the learner.

In similar fashion, the set of subsets of {1,2,3,4} ordered by inclusion is an example of a partially ordered set since inclusion of sets is reflexive, antisymmetric and transitive. For instance, in the Hasse diagram, if there is an upward path from node x to node y and from node y to node z there is an upward path from node x to node z. We can also establish that the partially ordered set {1,2,3,4} with inclusion is a lattice because every pair of elements has a union and an intersection, where both are defined as being the minimal instances of such unions and intersections. For example, the pair of elements {1,2} and {1,4} have a union of {1,2,4} and an intersection of {1}, both of which exist and can easily be apprehended by following links on the Hasse diagram. By virtue of being an interactive artefact, the P4 model can provide additional visual support for identifying unions and intersections by allowing the learner to select a pair of subsets and displaying their union and intersection.

The mathematical concepts discussed above could be introduced to the learner through their formal representations, but this does not give a learner access to the

stable experience that underpins this formal representation. The problems faced by the learner become more significant as the concepts become more complex. For instance, the notion of a decreasing subset in a partial order relies upon prior understanding of the notion of 'subset' and 'partial order'. Its mathematical formalisation is:

A subset Y of a partially ordered set P is decreasing if $\forall x,y \in P$: $y \in Y$ and $x \leq y \Rightarrow x \in Y$.

Figure 6.3(b) depicts a decreasing subset Z of the partially ordered set P4 as a set of blue nodes linked by red edges. The fact that Z is a decreasing subset can be appreciated by observing that all nodes beneath a blue node are blue. Reasoning about formal concepts can also gain experiential support from artefacts. For instance, a learner can observe that the decreasing subset Z is not a lattice as not all pairs of elements have a union in the subset (e.g. {2,4} and {3,4} are in the subset, but {2,3,4} is not).

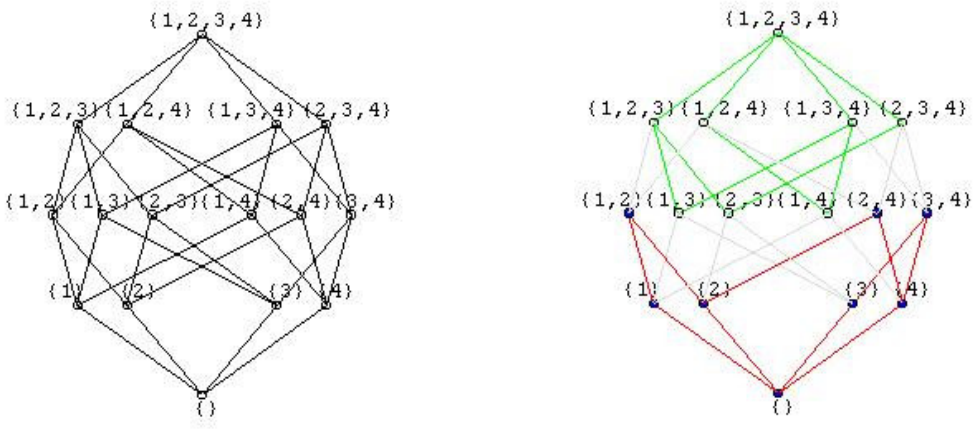


Figure 6.3 – (a) The lattice of subsets P4; (b) an example of a decreasing subset of P4

6.2.2 FDL4 as the lattice of decreasing subsets of P4 ordered by inclusion

Each decreasing subset of {1,2,3,4} can be regarded as itself an element in a set. This set of decreasing subsets is itself a partial order when ordered by inclusion. This partial order is depicted as a Hasse diagram on the right of Figure 6.4 – it defines one representation of the free distributive lattice on 4 generators (FDL4). The Hasse diagrams of P4 and FDL4 are linked by dependency in the EM model MBF4 in such a way that the selection of a node in FDL4 leads to the display of the corresponding decreasing subset of P4. Through experimenting with the selection of nodes in FDL4, I could observe that there is a line between two nodes if one of the associated decreasing subsets can be obtained from the other by adding a single subset of P4. From this, we can infer that the horizontal row of a node in FDL4 is determined by the number of subsets of P4 in the corresponding decreasing subset.

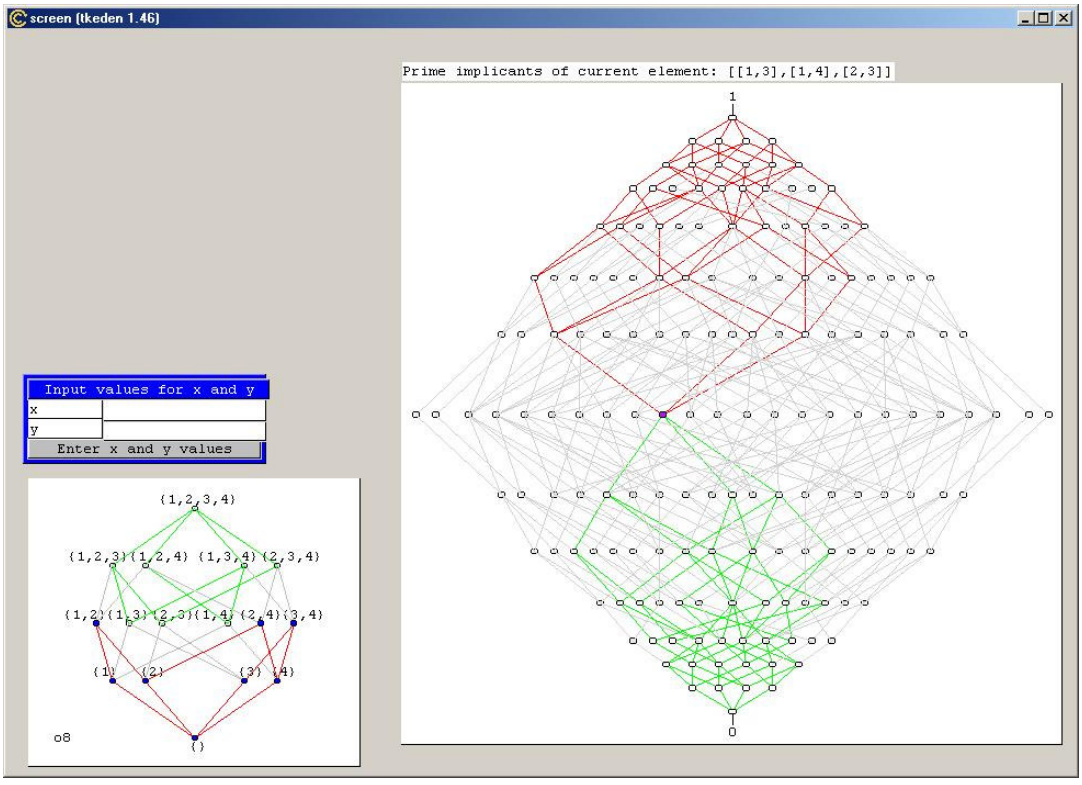


Figure 6.4 – The Hasse diagram with 166 nodes corresponding to the set of decreasing subsets of {1,2,3,4}

Other experiments that can be performed with the Hasse diagram of FDL*4* include picking pairs of nodes and establishing their point of union and intersection. For this purpose an interface to select two points and show their union and intersection is essential, given the size and complexity of the diagram. It is clearly infeasible to verify by hand that all pairs have a union and an intersection. However, through testing many examples we can become convinced that this is indeed true. As Beynon observes, such testing still leaves room for uncertainty:

> 'expectations developed through experiment are always subject to falsification, and are asserted subject to faith in prediction from past evidence. Expectations can be confirmed and confounded – but never justified – by experiment' [BAC+94].

As it happened, the visualisation in the original model contained an error: the internal model of union and intersection of decreasing subsets was correct but one link between two nodes was wrongly placed. It was only through personally extending the FDL*4* artefact that I encountered this error in the display of the lattice – it appeared that a pair of nodes had no intersection. The model user/developer's response to encountering anomalies of this nature in a model is heavily dependent on the context in which they occur. For instance, in model building at the frontiers of research such occurrences might dispose the modeller to discard a valid hypothesis. In our lattice example, had I encountered this error during my early interactions with the model I would have been less likely to believe that the structure was truly a lattice. As it was, I was already convinced that the structure was a lattice – and I construed the anomaly as an error in the graphical representation rather than an abnormality in the underlying mathematical structure.

6.2.3 FDL4 as monotone boolean functions in 4 variables ordered by implication

FDL*4* (see Figure 6.4) admits another interpretation – as a lattice of monotone boolean functions in 4 variables. A monotone boolean function is a function $f:\{0,1\}^4 \rightarrow \{0,1\}$ such that $f(x_1,x_2,x_3,x_4)$ is defined by a logical expression in x_1,x_2,x_3,x_4 using the operators *or* and *and*. An example of such a function is: $f(x_1,x_2,x_3,x_4) = (x_1 \text{ and } x_3) \text{ or } (x_1 \text{ and } x_4) \text{ or } (x_2 \text{ and } x_3)$.
Such a function is commonly represented by a circuit containing *and* and *or* gates. For this reason, we often refer to the tuple (x_1,x_2,x_3,x_4) as an input value and to $f(x_1,x_2,x_3,x_4)$ as the output. Given a monotone boolean function, we can consider the input values for which the output is 0. For instance, for the function f above, if $x_1=1,\ x_2=1,\ x_3=0,\ x_4=0$ then $f(x_1,x_2,x_3,x_4) = 0$. We can identify this input value with the set {1,2}, the set of indices of $x's$ that are assigned the value 1. With this convention, for the example function f above, the complete set of input sets for which $f(x_1,x_2,x_3,x_4) = 0$ is $\{\ \{\},\ \{1\},\ \{2\},\ \{3\},\ \{4\},\ \{1,2\},\ \{2,4\},\ \{3,4\}\ \}$. This set is exactly the decreasing subset that is depicted in Figure 6.3(b). By applying this general construction, the diagram of FDL4 in Figure 6.4 can be interpreted as the set of logically distinct monotone boolean function in 4 variables. Under this interpretation, the ordering of monotone boolean functions is by implication.

For me as a learner, the interactive artefacts described above played an important role in understanding lattices and related concepts. Drawing on my own personal experience, the interaction with the artefact served to both create and reinforce my understanding of the connections between the abstract mathematical objects P4 and FDL*4*, and monotone boolean functions in 4 variables. Interaction based on a formal symbolic approach would not have been as effective as I did not have the solid conceptual understanding needed to interpret the formal representations.

I was able to introduce the extra visualisations apparent in Figure 6.4 despite my incomplete understanding of the mathematics they represent. The following account of how I carried out this development is included to show that there is an intimate

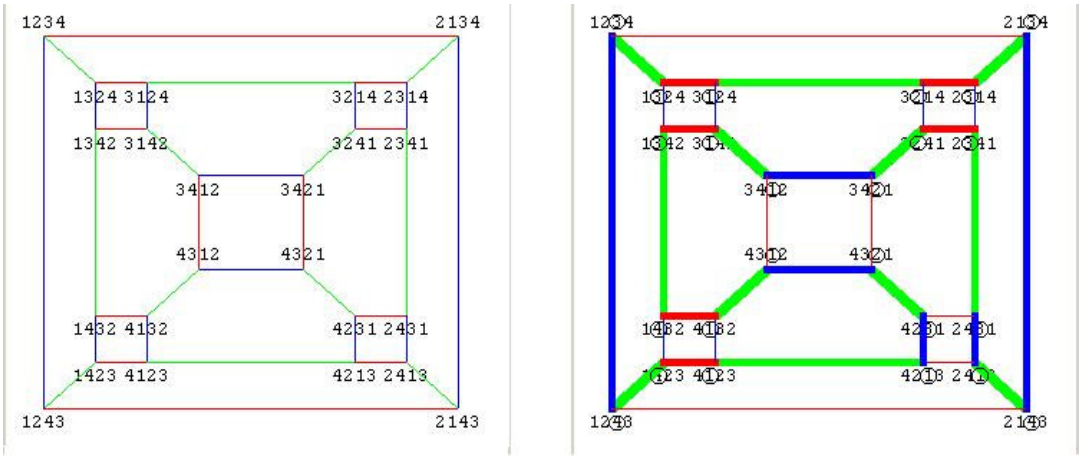
correspondence between analysing the dependencies in the model and understanding the relationship between different abstract representations of FDL4.

The red and green lines in Figure 6.4 are defined such that, if a line is above (respectively below) the currently selected point and it can be reached from the point by a strict upward (respectively downward) path then it is red (respectively green). With this convention, determining whether a particular line should be coloured red or green is equivalent to checking that the union of the selected point and the points at the ends of the line is either the selected point or one end of the line. Each point in Figure 6.4 is stored in an internal database and has a representation in EDDI (see section 5.4.3). This representation takes the form of a tuple that contains sixteen fields, each corresponding to a different subset of {1,2,3,4}, together with an identifier for each point. To discover the criterion for colouring lines above, I generated EDDI queries to construct tables of three example points, and observed a correlation between the existence of an upward/downward path in the Hasse diagram and a relationship between the internal representations of the three points. It was on the basis of this experimental evidence that I was able to understand the general relationship and implement this through introducing appropriate definitions (this was the activity that disclosed the visualisation error described earlier in this section).

The artefacts described in this section illustrate representations of particular concrete instances of free distributive lattices. The FDL4 artefact – which has only 166 nodes – can be used to support learners in gaining experience of the concepts underlying the MBF4 model in an exploratory fashion, but this approach cannot be extended to larger free distributive lattices. The visualisation of FDL5 would be impossible as it has several thousand nodes. The FDL4 artefact can nevertheless be used to gain the experience and concrete understanding necessary to be confident in manipulating the more general abstract concepts (cf. learning about geometry in 2 or 3 dimensions and moving to higher-order dimensional geometry).

6.2.4 S4: The symmetric group on 4 symbols

The third artefact that is part of the MBF4 model is a Cayley diagram for the symmetric group S4 of permutations of the 4 elements {1,2,3,4}. The structure of this Cayley diagram can be informally explained as follows. The numbers {1,2,3,4} can be arranged in 24 different permutations. Each permutation can have a pair of elements switched in three different ways – by transposing the first and second, second and third, and third and fourth elements respectively. In the Cayley diagram, two permutations are connected by an edge if each can be obtained from the other by such a transposition. This edge is coloured red, green or blue respectively, according to whether the transposition involves the first, second or third pair of elements. This is shown in Figure 6.5(a).



(a) (b)

Figure 6.5 – (a) A Cayley diagram for S4; (b) an example of a CPL map

Beynon's purpose in constructing the MBF4 model was to investigate the relationship between monotone boolean functions and functions defined on the Cayley diagram. This correspondence is formally described in the paragraph that follows. It represents an aspect of the MBF4 model that is outside the scope of interest of the novice learner.

Each monotone boolean function g in 4 variables determines a function G from the symmetric group S4 into the set {1,2,3,4}. To determine the function G, we interpret each permutation of {1,2,3,4} as an ordering for switching on the inputs to the monotone boolean function g. If we then switch on the inputs to the monotone boolean function g in the order associated with permutation p then the value of G on the permutation p is the index of the input that switches the output from false to true. For instance, for the function f introduced above, the switching sequence {1,4,2,3} will make the function true for the first time when input 4 switches from false to true (since f(1,0,0,0)=0 and f(1,0,0,1)=1). On this basis, the function F: S4 → {1,2,3,4} associated with f assigns the value 4 to the permutation 1423. The functions F and G generated in this way are known as *combinatorially piecewise linear* maps (CPL maps) [Bey74]. There is a constraint on a CPL map H: if two permutations are connected by an edge in the Cayley diagram and the value of H at one of the permutations is not one of the pair of values being transposed across the edge then H must have the same value at the other permutation. This constraint is sufficient to characterise CPL maps [Bey74]. For a given CPL map H, an edge of the Cayley diagram is *non-singular* if the value of H at the endpoints of the edge is not one of the pair of values being transposed.

The characteristic features of a CPL map can be visually represented as shown in Figure 6.5(b). For a given switching sequence, the index of the input that switches the output from false to true is circled. The singular edges are thicker than the non-singular edges. As Figure 6.5(b) illustrates, for some of the nodes in FDL4 (Figure 6.4) the singular edges form cycles in the Cayley diagram. For instance, in the example displayed in Figure 6.5(b), there is a Hamiltonian cycle of length 24. The interactive nature of the artefact allows the researcher to record and explore the relationship between FDL4 and CPL maps far more effectively than any paper-based approach (cf. the way in which this relationship is documented in [Bey74, Bey87b]).

The MBF4 artefact comprising the FDL4, S4 and P4 models described above is shown in Figure 6.6. As explained above, each correspondence between a pair of

components reflects a different perspective on lattice theoretic issues. Other relationships can easily be added to the model as they are encountered, by linking them through dependency to the relevant parts of the existing artefact.

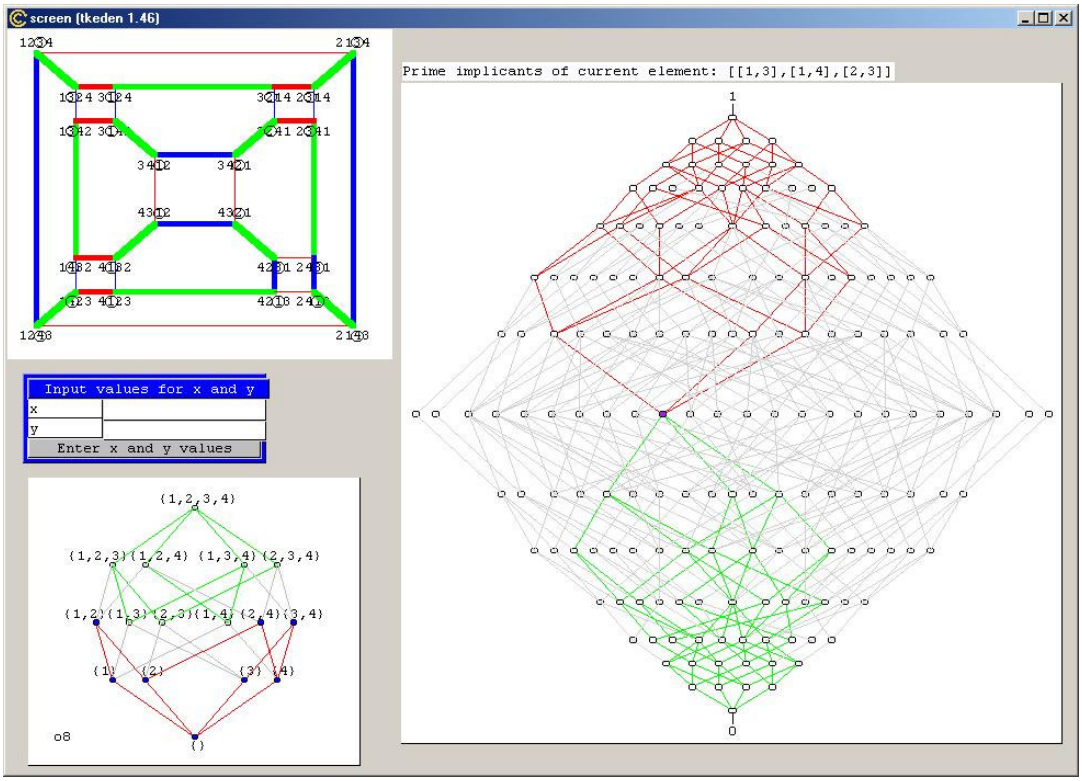


Figure 6.6 – The complete MBF4 model

The discussion in this section has focused on how the MBF4 model permits experimental interaction in order to learn about the concepts that inform its construction. This experiential approach is different in character from interacting with symbolic representations. Essentially this model has been used for two contrasting purposes: firstly as a learning aid, so that I personally could come to understand some of the basic concepts of lattice theory; and secondly as a research aid, so that exploration of connections between mathematical objects could be undertaken.

6.3 The Heapsort model

The heapsort model is intended to demonstrate the potential for the pedagogical use of EM to support understanding of formal algorithms. Heapsort is an advanced sorting method that relies on maintaining partial orders within a data structure known as a heap. For a fuller description of the heapsort algorithm, see [AHU82]. Meurig Beynon constructed the basic EM heapsort model discussed here, initially with the assistance of Amanda Wright [BRS+98]. Significant extensions to the model were made by Jaratsri Rungrattanaubol and the relationship between her model and a conventional program to teach heapsort is discussed in [Run02, Chapter 6].

The EM heapsort model is not intended to be a formal representation of the heapsort process, but rather an environment within which activities that are related to heapsort can be investigated in an experiential fashion. The emphasis is on exposing the empirical knowledge that contributes to the design of the heapsort algorithm. Conventional ways of providing computer assistance for teaching heapsort involve animating the algorithm and displaying the stages of the process (cf. the JELIOT system [BMS+02], Animated Algorithms [GDL]). However, this approach assumes that the learner has already understood the notion of a heap and the basic operations that can be performed upon it. The approach adopted in the EM heapsort model has more in common with that of the Brazilian educator Valdemar Setzer, who proposed that learners manually perform the sorting operations with physical objects [SH93].

In contrast to a conventional animation, the EM heapsort model can fulfil many different learning objectives [Bey98]. For instance, it can be used as:

1) an environment for testing a student's understanding of the concept of a heap and of the procedures used in heapsort.
2) a visualisation aid for the exposition of the heapsort algorithm.
3) a prototype for the implementation of heapsort in a conventional programming paradigm.
4) a platform for investigating variants of the heapsort algorithm.

This diverse range of learning objectives can be satisfied within a single EM learning environment by building up the partial heapsort model with combinations of agent actions that are stored in separate auxiliary files. In general these files are added to the model according to the current needs of the learner and the purpose for which they are using the heapsort model.

Beynon describes three stages in using the EM model to learn about heapsort [Bey98]:

1) experimental manipulation of a visual heap to understand the heap concept. This is possible because the heap is embedded in a definitive script that does not constrain the possible agent actions to be added to the model later.
2) the construction of 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. These stages are introduced as agents to perform parts of the sorting activity.
3) the introduction of automatic mechanisms to carry out the appropriate sequence of steps. This is achieved through the automation of sensible heapsort behaviours in particular patterns.

As mentioned above, the implementation of a conventional heapsort teaching program engages primarily with the activities at stage 3. Stages 1 and 2 are concerned with obtaining a solid construal of the concepts and basic operations involved in heapsort. Stages 1 and 2 admit exploratory learning and experiments to achieve the necessary background knowledge to fully appreciate the heapsort process. As Beynon remarks in [Bey98]:

> 'the most effective way to present the model construction is to systematically introduce the underlying concepts as they might have been encountered in the discovery of the heapsort algorithm'.

Initial exploration of the heapsort model centres on understanding the heap structure. Figure 6.7 shows the heap data structure. A binary tree is a *heap* if each node satisfies the *heap condition*, which states that the value of a node is greater than that of both its children. Figure 6.7 shows how the visualisation of the heap structure provides cues to the learner about the state of the heap. Edges and nodes are coloured to reflect the current status of the heap condition at each node and the relationships between nodes and their children. For instance, if all the edges are blue then the tree is a heap.

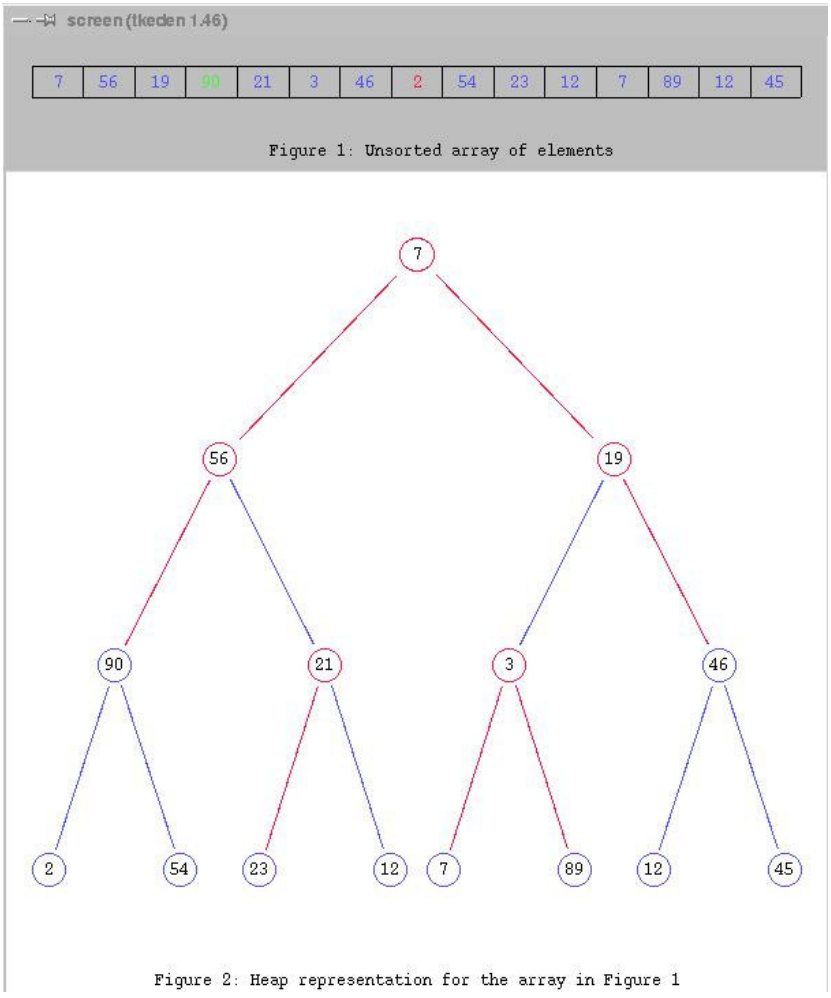


Figure 6.7 – The heapsort model showing a representation of a heap

Embellishments to the basic model add agent actions that correspond to primitive operations on the heap and are a stepping-stone to understanding the heapsort algorithm. The simplest operation on the heap is exchanging a pair of elements. This can be performed manually, or by agents that can be attached to the nodes of the heap model to perform this exchange automatically, given a pair of indices. The learner can attempt to manipulate the tree into a heap by exchanging suitable pairs of elements. Through exploration, the learner will begin to comprehend sensible strategies for establishing a heap. For example, if a node does not satisfy the heap condition then the most effective strategy is to exchange values with the child node that has the larger value. The index of the child with the larger value is maintained by dependency and can be consulted as an observable by the agent at each node (cf. Figure 6.4 in [Run02]). Such agents can then automatically establish and maintain the heap condition at each node. The model can be set up so that these agents act autonomously to perform the complete process of heap construction. This is most appropriate when the learner has first understood how manual redefinitions can establish a heap.

All the basic operations required to understand heapsort are exercised in the heap construction phase (stage 2). From this point, it is a relatively easy to derive the complete heapsort algorithm, which repeatedly removes the root of the tree and then re-establishes the heap condition until the tree becomes empty (stage 3). As is typical of other EM models discussed in this thesis, the openness of the EM approach allows interaction and exploration outside the scope of normal heapsort operation. Rungrattanaubol outlines many possible scenarios for the use of the EM heapsort model that would be impossible with conventional educational software [Run02, pages 177–178].

The heapsort learning environment described above illustrates interaction with an EM model that conflates both model-use and model-building perspectives. This conflation of perspectives allows the integration of concrete and formal learning activities within a single learning environment. The definitions in the heapsort model are organised

into files in a directory. Each file addresses a different feature of the model. Initially the learner is presented with a simple model of a binary tree within which they can explore the concept of a heap. This approach is beneficial in two respects:

i) personally constructing the model gives the learner an appreciation of the key concepts and operations involved in heapsort.

ii) because of its interactive nature, the heapsort model allows the learner to experiment at each step in the construction to ensure that they understand the interaction between components.

Note that in the model building the learner has flexibility in how and when the supplementary files are added to the existing model, but in general they constrain the heapsort model towards the heapsort algorithm. The learner can follow a prescribed sequence of steps in the construction of the model (cf. the README file in [EMRep, heapsortBeynon1998]). The learner nonetheless has complete discretion over their interactions with the model, can interact outside the scope of conventional heapsort, or can manually perform heapsort operations. This flexible style of interaction is essential to experiential learning and provides the type of activities that can foster a full and deep understanding of the heapsort algorithm.

The heapsort model exemplifies the way in which EM can support the wide range of learning activities in the EFL. Stages 1 and 2 referred to above are concerned with understanding the observables and dependencies that are characteristic of the heap data structure and subsequently identifying the indivisible stimulus-response mechanisms that govern the behaviour of the heapsort algorithm. The discussion above has described how, in using the heapsort model, the learner initially focuses on the heap structure, then moves through manual operation of the sorting process to the automation of heapsort. The basic heap model is a definitive script that describes the state of the heap. It is gradually embellished with agent actions to represent necessary operations on the heap. This process reflects the emphasis on state-as-experienced as prior to behaviour-as-abstracted, fundamental to the EM approach, that was discussed in chapter 3. A further extension of the heapsort model that has been implemented by

Rungrattanaubol [Run02] complements the model with a formal specification of heapsort in the Weakest Precondition formalism [Dij76] (see Figure 6.8).

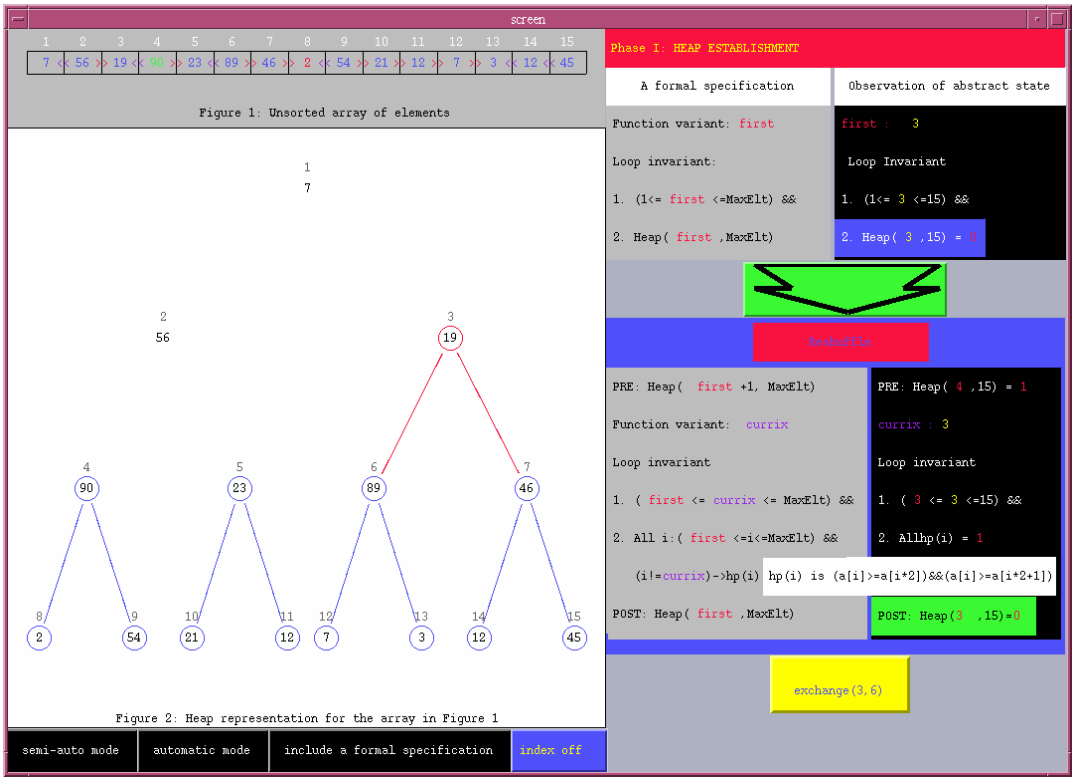


Figure 6.8 – Heapsort and its associated formal specification

In this extension, the logical relationships between key variables are themselves interpreted as observables at a high level of abstraction, as discussed in detail in [BRS00]. This extension of the heapsort model illustrates how EM can support learning activities across the whole of the EFL, from experimental interaction to formal reasoning.

6.4 The Robotic Simulation Environment

In this section, we describe a prototype EM learning environment developed to help learners understand the behaviour of robots in LEGO Mindstorms™, a system that allows computer programming and the construction of physical LEGO robots to be

integrated. The learning environment to be described in this section is targeted at the particular scenario being used in Kids' Club, Joensuu, Finland. It shows how EM can be used to provide support for experiential learning activities.

The concept of out-of-school technology clubs for children has been successful in giving children a deeper understanding of, and confidence with, computer programming and technology in many different cultural contexts [RR96, RRC98, ESV+02]. The primary aim of these clubs is to give children of various ages the chance to work on personally meaningful technology-based projects within a relaxed and informal setting. These clubs employ students and teachers to assist children in achieving their own personal goals without imposing rigid curricula or examinations on them. The first technology club for children was set up in 1993 and run by the Computer Museum in Boston in collaboration with the MIT Media Laboratory [Com03]. The aim was to create an atmosphere within which children and adults could collaboratively construct technological artefacts facilitated by computer technology. Many computer clubhouses now exist across the world under the auspices of the Intel Computer Clubhouse Network [Com03].

The Kids' Club, run by the Educational Technology Research Group based in the Computer Science Department at the University of Joensuu in Eastern Finland, is a recent initiative that shares the ideology of the Intel computer clubhouse. The motivations behind the Kids' Club are twofold: firstly to be an environment where children can undertake technological projects built on their own interests beyond the boundaries of the school curriculum; and secondly as a laboratory setting within which researchers can field test new educational technologies [ESV+02]. Children are not merely participants in the technological phase of the project; they often contribute to the research that is being undertaken. The close involvement of the children in the research process is the main difference claimed by the Kids' Club organisers from the Intel Computer Clubhouses [ESV+02].

There is a wide range of technology-based activities that children undertake in the Kids' Club. They have the opportunity to program computers, to use the Internet, to create interactive Java type animations and to use modern digital media such as video cameras. One task in which children have been heavily involved is in constructing LEGO™ Mindstorms robots (hereafter to be referred to as 'robots') and programming them to achieve tasks in a real-world environment. In the next section, we discuss the fundamentals of robot construction and programming and outline some of the conceptual problems facing learners. These problems motivate the EM learning environment to be discussed in section 6.4.2.

6.4.1 Building and programming robots

LEGO Mindstorms robot kits consist of conventional LEGO blocks and pieces, wheels, connectors, a programmable device, and sensors that can be attached together to build robots. Wheels are connected to motors that enable the robots to move. Robots have independently operating motors to drive the wheels on each side of the robot. This allows the robot to turn with only one motor running. Sensors can be attached to the robots that allow it to interact with its environment. For example, a touch sensor will return a positive value if it is touching an object in its environment. Each construction kit contains touch sensors, light sensors and colour sensors which are used in conjunction with the programming language so that the robot can interact with, and respond to, its environment. At the heart of a robot is the Robotic Command eXplorer (RCX) Programmable Brick. This is a large LEGO brick that provides the battery power for the motors and is also a computer that can store and run programs. The programs are written on a personal computer before being uploaded to the brick by an infrared communication device.

Robots can be constructed in many different ways. In our prototype environment, we have chosen one specific robot design. An example of such a robot can be seen in Figure 6.9: it has a motor attached to the pair of wheels on each side. In principle, our environment could be extended to encompass arbitrary robot design.

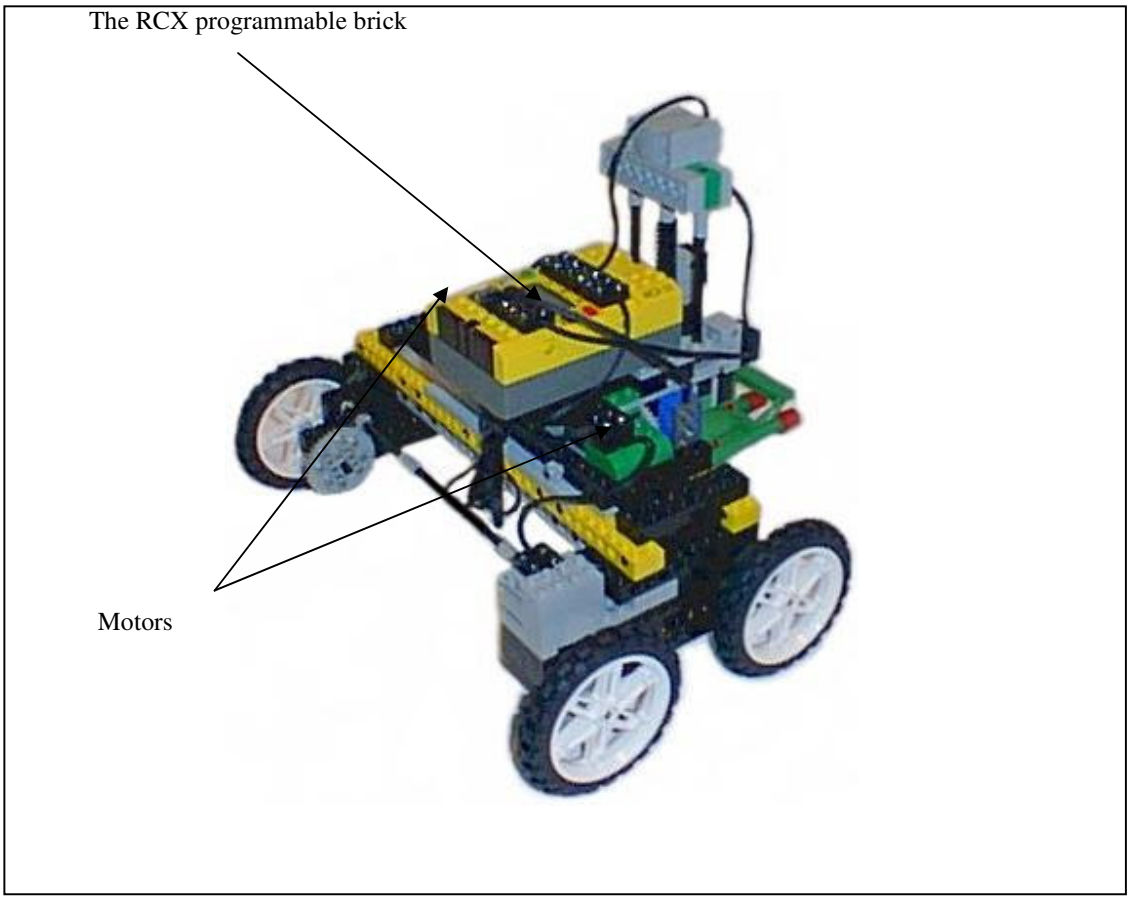


Figure 6.9 – An example robot and its main features

Robot programs are created on a computer and transferred to the programmable brick. The robot programming language has commands to control the motors that drive the wheels. The controls allow the robot to move forwards or backwards in a straight line, to rotate on the spot, to rotate in either direction centred on a wheel, or to move in a circular path forward or backward. Each motor can be turned on or off or have its speed or direction changed. The complex behaviour that can emerge from a set of simple commands requires a good understanding of how the commands map onto the real-world movement of the robot. There are commands that allow the robot to respond to input from the sensors. For instance, when a robot touches an obstacle, a touch sensor will return a true value that can trigger a change in the robot's movement. Programming language constructs such as 'if-then-else' or 'while-do' loops can be used to create more complex robot behaviours. Listing 6.1 illustrates

how a robot can be programmed to move around a room. When it hits an obstacle, the robot reverses, changes direction and then moves forward again.

```
REPEAT FOREVER
     AB: On forward. speed 7.
     IF TS1 = 1
          AC: On reverse. speed 2. Continue 2 seconds.
          A: On forward. speed 2. Continue 1 second.
     END IF
END REPEAT
```

Listing 6.1 – An example robot program listing

Robot programs are developed on a computer using either a text editor or a specially designed programming environment such as the Instructive Portable Programming Environment (IPPE) [JKS02]. The IPPE allow commands to be built in a visual manner, thereby eliminating syntax errors. It uses a clipboard to store potentially useful commands before they are committed to a program. Commands can be added to a program from the clipboard in any order, and new ones can be placed on the clipboard at any time. In the IPPE, commands are created using dialog boxes such as that shown in Figure 6.10, where the command is 'Turn on motors A and C with speed 7' [JKS02].

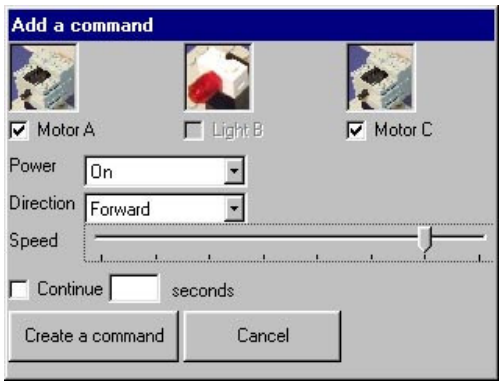


Figure 6.10 – Using the IPPE to create a command

Complete programs can be transferred to the programmable brick using an infrared communication device. When the program has been uploaded to the robot it can be run and the behaviour of the robot can be observed. Following observation of the robot's behaviour, the program can be debugged if required by considering the problem, making changes to the program, transferring the program to the brick and running it. This leads to an iterative program development cycle comprising programming the robot, testing its behaviour and conceptual debugging (see Figure 6.11). Each of these phases occurs in a different context: programming on the computer, testing in the world and conceptual debugging in the mind.

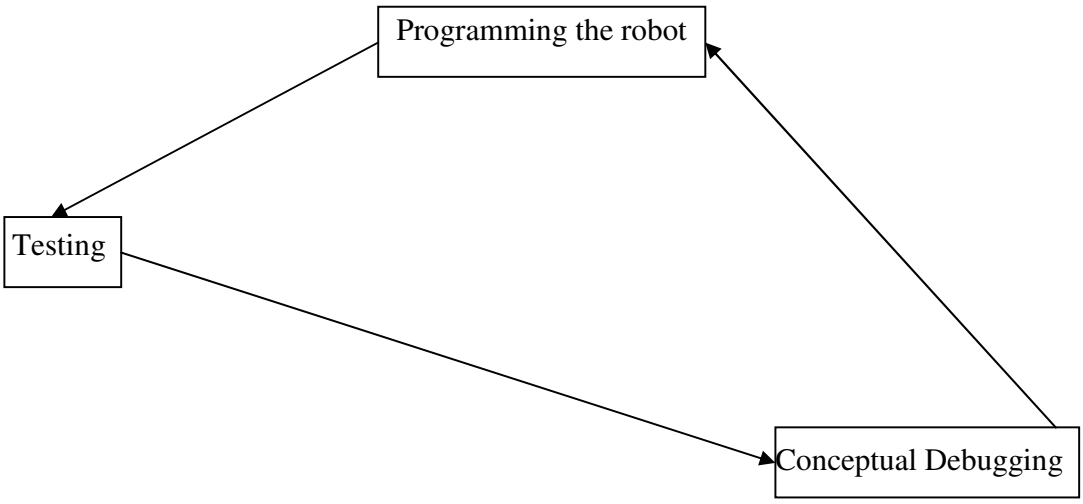


Figure 6.11 – The iterative robot programming cycle

The problem with the current programming environment, and the iterative developmental style outlined above, is that significant cognitive demands are placed on the learner. This becomes apparent when we consider the steps involved in a typical robot programming scenario.

In practice, each iteration of the development cycle outlined above can take several minutes. During this period, the learner has to remember the modifications they had made and what they were trying to correct, and at the same time check whether their

revised program solves the problem through analysing the observed behaviour. If a shorter amount of time were required to complete a cycle, then – in addition to the intrinsic timesaving benefit – the cognitive demands on the learner would be drastically reduced.

A related problem in understanding a robot program is that the program code and the resulting behaviour are studied in separate environments. The learner's task – trying to grasp why the robot is not performing as it should, and working out how to debug the code – combines observation of the robot with analysis of the program code. This requires an understanding of how the available robot commands relate to primitive robot behaviour. If the learner has an inadequate construal of this relationship, the programming task is exceedingly difficult.

A further problem with the current robot programming approach – with its associated long feedback loop – is that it is difficult to perform experiments to determine the reasons for a robot's undesired behaviour. An environment in which commands and programs can be easily tested, that allows users to interact on a level appropriate to their understanding, promotes exploratory learning. This accords with diSessa's observation that a key factor in the design of learning environments that promote active learning is that it should be easy to explore personal hypotheses [diS01].

The demands discussed above can only be met if the learner has sufficient experience of the relationship between robot programs and robot behaviour. The construction of a robot simulation environment is motivated by the fact that novices do not possess the experience that is required to program successfully. A visualisation to make the relationship between the program code and the robot's behaviour more explicit would simplify the understanding of robot programming. This would reduce the cognitive load on the learner by bringing together the elements of programming, testing and conceptual debugging into a common environment. From an EFL perspective, robot programming in the existing environment assumes that the learner understands the relationship between robot program and robot behaviour and does not require support

for the experiential learning activities that inform this understanding. This support can be established by providing features that enable exploratory learning to establish a solid construal.

In the next section we discuss a prototype EM robot simulation environment targeted at solving the problems associated with learning to program robots in the Kids' Club.

6.4.2 The Empirical Modelling Robotic Simulation Environment

The primary aim of the EM Robotic Simulation Environment (RSE) is to reduce the cognitive demands in learning to program the robots. The work that is reported in this section originated from group work with Pasi Eronen, Järi Järvela and Marjo Virnes at an Educational Technology summer school held in Finland in August 2002 [EJR+02].

The RSE supports the development of a child's construal of robot behaviour by targeting a better understanding of how program commands are related to the behaviour of the robot. This is achieved through a layered environment that eliminates the long feedback loop associated with the current programming approach. With reference to the EFL, the RSE permits learners to undertake activities situated at the experiential end of the EFL, by providing different perspectives that can support the development of objective knowledge.

The RSE provides cognitive support for the key feedback loop that is driven by the problem that the learners are trying to solve (see Figure 6.12). Through observations about the robot behaviour in the environment they can test hypotheses and gain feedback on them. Solutions to their problems are the catalyst for successful concrete robot building. Learners' construals are used as the basis for solving new problems that are refined to suit the context of their new understanding.

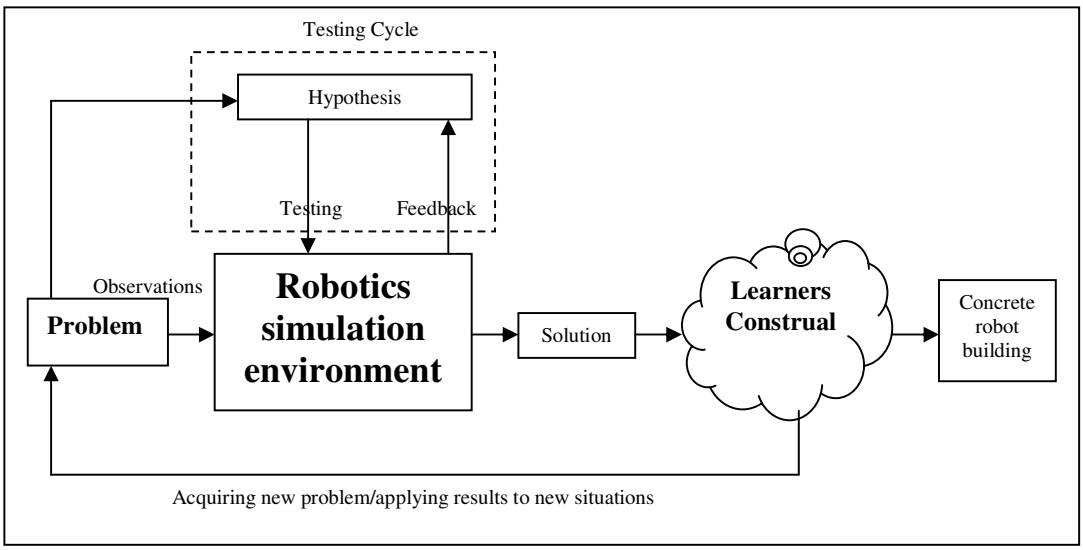


Figure 6.12 – Learning and the Robotic Simulation Environment [EJR+02]

The complexity and duration of the feedback loop is dramatically reduced in the RSE because of the computer-based environment and the quality and type of feedback provided. With the current approach, a large proportion of the time spent debugging is taken up with physical manipulation of the robot and observation of its behaviour in the real world. In a computer-based model, the resultant behaviour can be more easily correlated with the program that defines it through tracking the program commands as they are executed. For this purpose, the behaviour of the simulated robots must match that of their real-world counterparts, otherwise the environment would not allow for the transfer of knowledge from the RSE to robot programming. Ideally, the simulated robots should be programmed using the same programming language that is used to control the physical robots (cf. Listing 6.1). This feature has not yet been implemented in our prototype RSE but it would be possible to specify the programming language for the robot using the AOP (see section 5.4.1). In essence, the RSE allows the learner to concentrate on the essential features of programming the robots and supports experimental activities targeted at establishing or reinforcing their construal of robot behaviour [EJR+02].

6.4.3 Layering in the Robotic Simulation Environment

The RSE is designed to support learners at many different levels of competency [EJR+02]. Some learners will use the environment to learn about the basic concepts involved in robot programming. For instance, the learner may wish to explore the relationship between primitive robot movement and the programming commands needed to produce that movement. Other learners will already understand the basics of robot programming and will use the environment to test hypotheses. For instance, the learner may wish to confirm that a particular program successfully achieves its goal irrespective of the initial orientation of the robot. Both these agendas can be satisfied in a single layered environment. We now illustrate different types of use within the RSE by describing two different layers. For a fuller description, see [EJR+02].

The basic layer of the RSE allows learners to manipulate the robot using predefined controls to establish how robot movements are related to the underlying program code that produces them. This of course makes fundamental assumptions about the environment such as: the terrain is flat, the friction is uniform, and there is no wind. It also assumes that the robot has already been configured so that the touch and light sensors are attached. As depicted in Figure 6.13, the robot is controlled using a set of buttons labelled 'Forward', 'Backward' etc, to specify an intended movement of the robot. If the learner presses 'Forward' then the robot will move forward in a straight line until it receives a further command or until it hits an obstacle. By using these buttons and referring to the current status of the components, the learner can explore how the movement of the robot is related to the internal state of its motors. For instance, when the robot is moving forward, both motors are running in the same direction at the same speed. To provide scaffolding for writing robot programs, each time a button is pressed, the equivalent robot command for that movement is displayed using the syntax of the IPPE language. For instance, in Figure 6.13 pressing the forward button has set motors A and B to '`RUNNING`' and the code output window shows '`A,B: on forward. speed 2`'. The important activity in this

layer is concerned with understanding the basic repertoire of movements of the robot and the dependencies between commands and movements. The robots are assumed to possess some automatic stimulus-response mechanisms. For instance, if a robot touches a wall then the touch sensor will return a true value, which is registered in the code output window.

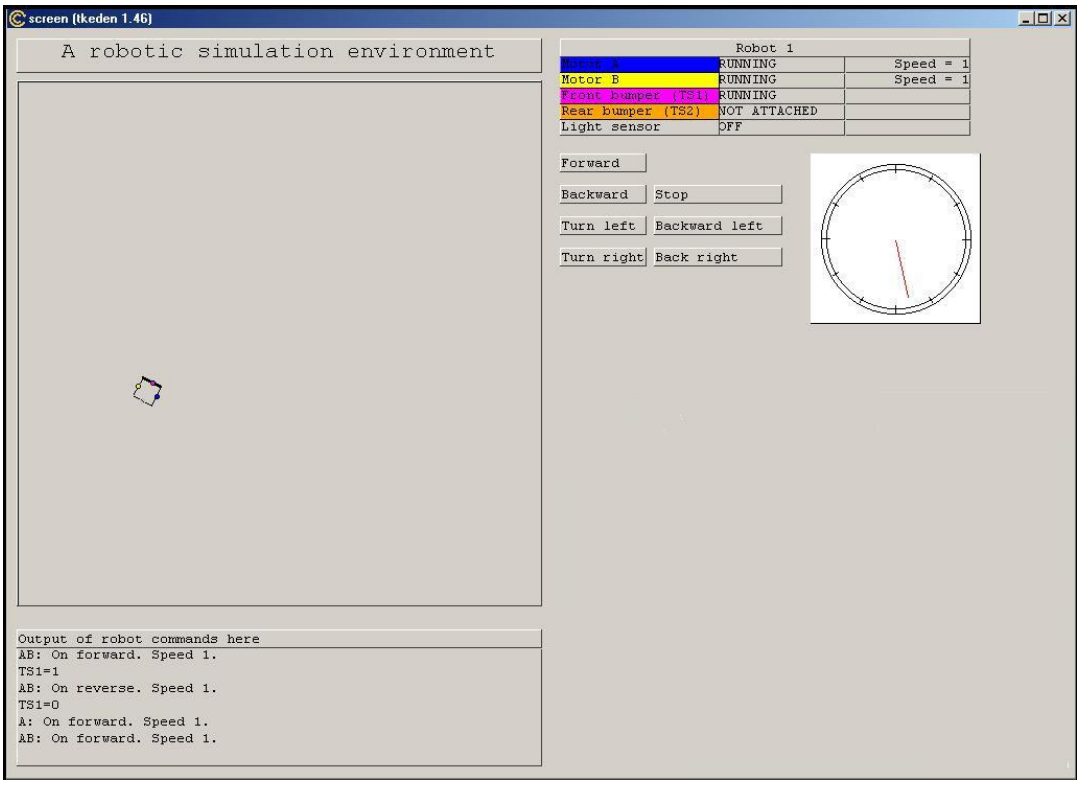


Figure 6.13 – The RSE being used to investigate the relationship between the motors and the robot's movement

The exploratory nature of the RSE helps to develop the learner's construal of the relationships between the movement of the robot, the setting of the motors attached to it and the equivalent program code needed to generate that behaviour. The learner needs to understand the fundamentals of robot behaviour in order to program the robots effectively. This support for establishing and reinforcing a learner's construal is a major advantage that the RSE has over the IPPE.

A layer of the RSE to support more advanced learning gives learners the power to simulate the physical construction of a robot and write programs in the IPPE language. In programming the physical robot, sensors and motors must be attached to the robot, and wired to the programmable brick in order for program instructions to be passed to the motors and sensors. Forgetting to connect devices is a common mistake that children must learn to avoid. To support this in the RSE, attaching and connecting the motors are two separate actions, which reinforces the need for this to be carried out in the real world. To program the simulated robot the learner must enter code into the robot commands window (see Figure 6.15). In the current prototype, the robot is programmed using direct EDEN counterparts of the primitives and control structures available in the IPPE language. Because of this close syntactic similarity between EDEN and IPPE code, providing a translator from IPPE into EDEN using the AOP should not prove too technically demanding. By entering code fragments into the input window, it is envisaged that children will be able to write simple commands to replicate single operations, or construct programs using the full range of commands and high-level language constructs available in the IPPE language. When commands are being executed in the RSE, the status of the motors and sensors is kept up to date by dependency so that learners can use the environment to debug their real-world programs, or determine how their program influences the internal state of the robot.

The layers of the RSE are not intended to be completely separate and we envisage that learners writing their own programs could also use the direct manipulation controls to test a hypothesis about a situation they are trying to understand in their program. With reference to the EFL, this reflects the way in which learning activities migrate from the concrete to the abstract and vice versa as a learner consolidates his or her advanced understanding by exploring areas of uncertainty. The intention is that as learners become more competent they gain a better conceptual understanding of programming robots and more of their time is spent testing hypotheses and experimenting with real-world programming.

We now describe a practical example of how the RSE can be utilised by learners in trying to solve a programming task. The scenario described here is originally from [JKS02]. The objective is to write a program to navigate a robot around a rectangular obstacle, as shown in Figure 6.14.

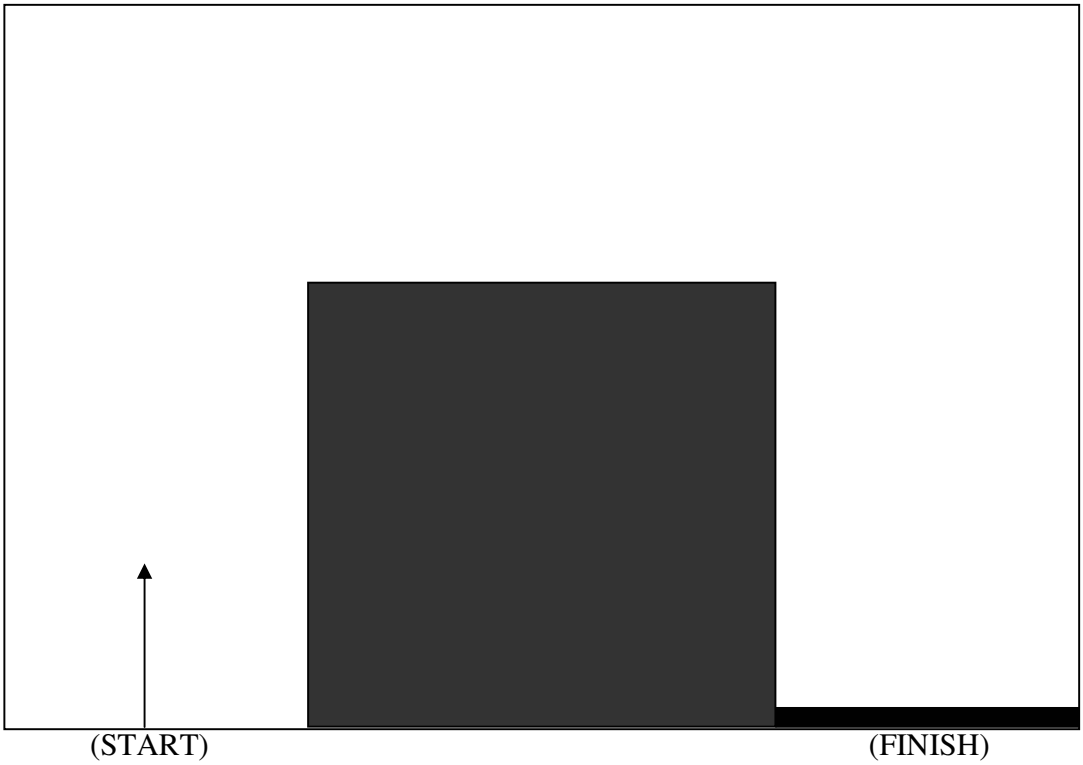


Figure 6.14 – An example task for a robot program to solve (from [JKS02])

Setting up the problem scenario in this example requires specifying the world in which the robot will interact. This simply contains four boundary walls to represent a square room and three interior walls to represent the obstacle. These walls are specified as definitions and can be changed whilst the robot is moving as when simulating a moving obstacle.

Learners can interact at either a direct manipulation level, or at a programming level in order to solve the problem. Direct manipulation may be used to understand the types of commands that are needed to solve the problem and to identify a starting

point for writing a program to navigate around the obstacle. Alternatively learners can write a program for the robot. When the program is run, the behaviour of the robot can be observed and debugged as required. Figure 6.15 shows the RSE in use in solving the given problem.

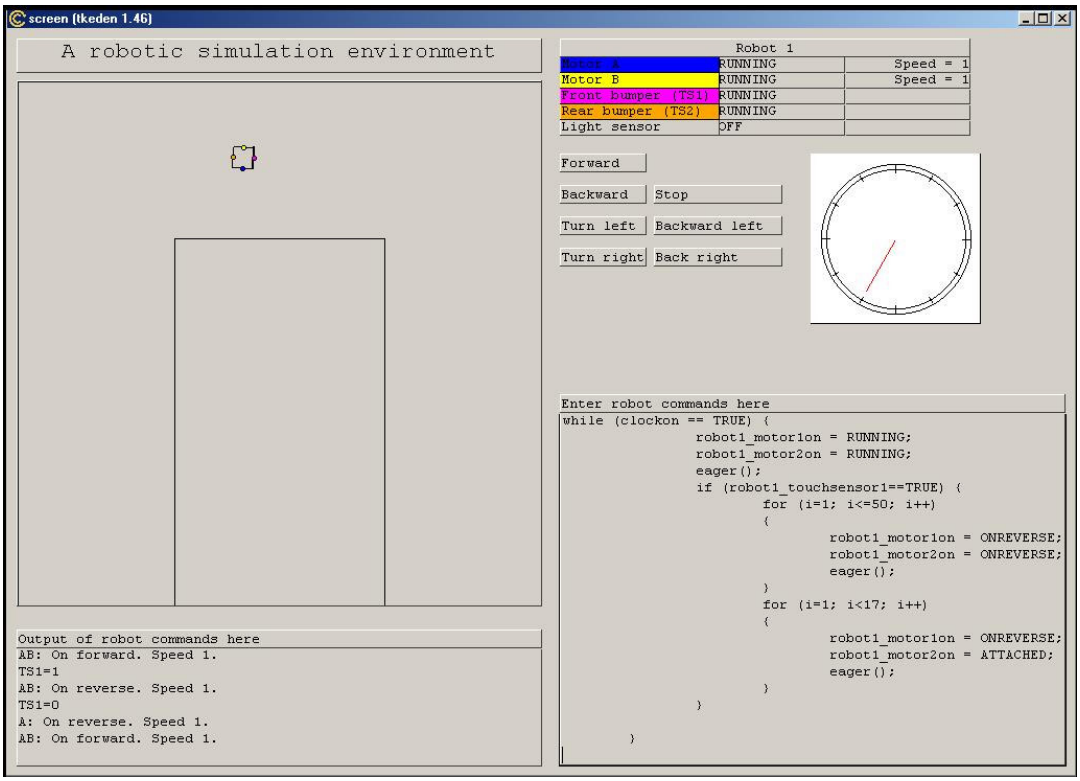


Figure 6.15 – The RSE in use in solving the task from Figure 6.14

This example shows that the RSE can be used to solve problems that are encountered in robot programming in the Kids Club. There are many ways robots can be constructed, and many tasks that the robots could be programmed to perform in the real-world. The RSE will not be able to support them all, but this example illustrates that it can play a role in establishing a conceptual understanding of robot programming.

In its present form, the RSE is not sufficiently developed to deal with robot programming in its full generality. However, the above example establishes proof-of-

concept, and illustrates that the RSE can be applied to at least some of the problems that learners face when programming robots in the Kids' Club. There are many things that still need to be added to make it more suitable for its intended audience. Providing an interface for the IPPE language is one of the principal concerns. However, there are a number of other features that could be added to the RSE in order to enhance its usability. These include:

- a graphical interface to enable learners to create and manipulate the robots' environment interactively.
- a more realistic visual representation of the robot, potentially in 3D using the Sasami notation.
- support for developing and simulating different robot designs.

The RSE has further potential as a test bed for designing new features that enhance the robots. For example, it would be of interest to introduce communication sensors so that teams of robots could work collaboratively or competitively to achieve a task [EJR+02].

In this section, we have shown that the RSE can allow learners to gain an understanding of the concepts underlying robot programming without having to concentrate initially on specifying the behaviours of the robot in abstract program code. This approach allows the learner to intersperse concrete manipulation and abstract programming according to their current task and learning needs.

6.5 Chapter summary: Supporting learning across the EFL in Empirical Modelling

In this final section, we draw together the three characteristic case studies described in this chapter and relate them to the EFL. We argue that model building and model use – from the perspectives of many different learners – requires support for learning activities across the whole range of the EFL.

By way of illustration, in section 6.2, we discussed the MBF4 model from the perspective of a learner and a mathematical researcher. For a learner, the primary learning activity is in understanding the mathematical concepts embedded in the model. Learners require support for formal concepts through the visualisation of concrete examples, allowing them to move from the abstract end to the concrete end of the EFL. For the mathematical researcher, the primary learning activity is investigating potential connections between various mathematical objects. This task involves adding to the model in order to test out conjectures through experimentation with concrete examples.

Both model building and model-use can be associated with learning activities throughout the EFL. As discussed above, in respect of model building, the MBF4 model is primarily focused on the learning activities at the concrete end of the EFL. In respect of model use, the primary focus of the RSE in section 6.4 is to give learners access to an experimental environment in which they can seek to consolidate their understanding of programming LEGO Mindstorms robots. With reference to the EFL, this is associated with enabling learners to connect the formal symbols in robot programs with their concrete meanings in the world. The Heapsort model in section 6.3 – which integrates aspects of model building and model use – seeks to integrate the formal characteristics of the heapsort algorithm with the experiential elements that inform it. With reference to the EFL, this gives learners the scope to move either towards the abstract or towards the concrete in their exploration of heapsort.

The case studies described in this chapter – and more generally throughout the thesis – are practical evidence of the claim that EM can support learning across a wide range of learning activities and domain contexts.