Ramesh KRISHNAMURTI
EdCAAD, University of Edinburgh

Modelling Design Descriptions

ABSTRACT

Une vue gènerâle de la representation schématique employeé par le systeme MOLE est repręsenteé. La relation du ''kind-slot-filler'' est schematiseé avec deux sortes de mechanisme enhéritant, pour la construction hierarchiquement structureé des descriptions des projets.

An overview of the representation scheme employed by the MOLE modelling system is presented. The kind-slot-filler relationship is outlined, together with two sorts of inheritance mechanisms for constructing hierarchically structured design descriptions.

MODELLING DESIGN DESCRIPTIONS

Ramesh Krishnamurti

EdCAAD
Department of Architecture
University of Edinburgh
Scotland

An overview of the representation scheme employed by the MOLE modelling system is presented. The kind-slot-filler relationship is outlined, together with two sorts of inheritance mechanisms for constructing hierarchically structured design descriptions.

Une vue géneràle de la representation schématique employeé par le systeme MOLE est representeé. La relation du "kind-slot-filler" est schematiseé avec deux sortes de mechanisme enhéritant, pour la construction hierarchiquement structureé des descriptions des projets.

1. Introduction

This paper provides a flavour of the representation scheme that underlie the MOLE modelling environment being developed at EdCAAD. It reports research that is directed at computer-based systems that can accommodate the idiosyncratic nature of design practice, without prescriptions to the form or content of designs. That is, systems that assist in the design process by enabling designers to make 'reasonable' statements about design objects; to ask 'reasonable' questions about these objects; and to perform 'reasonable' tasks on these objects.

Implicit in this approach is the view that designing is an activity dependent on designers' perceptions of design tasks and their resolution. In the context of computer-aided design, this view of design demands that the crucial element in any machine environment lies in the ability of the machine to accept (partial) descriptions of design objects. Moreover, these descriptions can be manipulated according to some (perhaps unanticipated) criteria that the designer may wish to apply.

A model for *intensional* descriptions of objects is presented. That is, a description that can be structured so that it can be used to recognise objects and can be compared with other descriptions. Such a description of an object should be organised around entities with associated descriptions, it must be able to represent partial knowledge about an object, and it must accommodate multiple descriptors which can describe the object from different viewpoints. Moreover, these descriptions should possess a quality of truth in that they *reflect* the (factual or otherwise) beliefs held by the designer. One way to treat these descriptions is to regard them as statements that belong to some logical framework.

2. The building blocks of a description

2.1. Kinds, slots, fillers

A representational structure that is akin to *frames* [1] or *semantic nets* [2] is employed. Where it differs is that there is no prescribed interpretation associated with the relational links between object descriptions. That is, entities have no connotation beyond the conventions one observes when using them. Of course, entities employed by specific decision procedures must have specific denotations.

The representation uses three sorts of entities: *kinds*, *slots* and *fillers*. A kind K corresponds to an object. A filler F can be any object and represents a value for some property or feature that a kind may have. A slot s denotes a relationship between a kind and a filler. There are no restrictions imposed on filler and slot types. Any filler can be associated with any slot. A filler may also be a kind.

It is clear that a hierarchically structured description for objects can be constructed from these basic building blocks. As a simple illustration consider the possible description of the kind DOOR given in the form of a relational table.

KIND	SLOT	FILLER
DOOR	type	PANNELLED
	material	RED_PINE
	construction	HOLLOW
	height	1600
	door_knob	DOOR_KNOB
	manufacturer	UNDERWOOD & CO
DOOR_KNOB	material	BRASS
	manufacturer	LOCKE

Pictorially, the description given in the above table can be drawn as shown in figure 1.

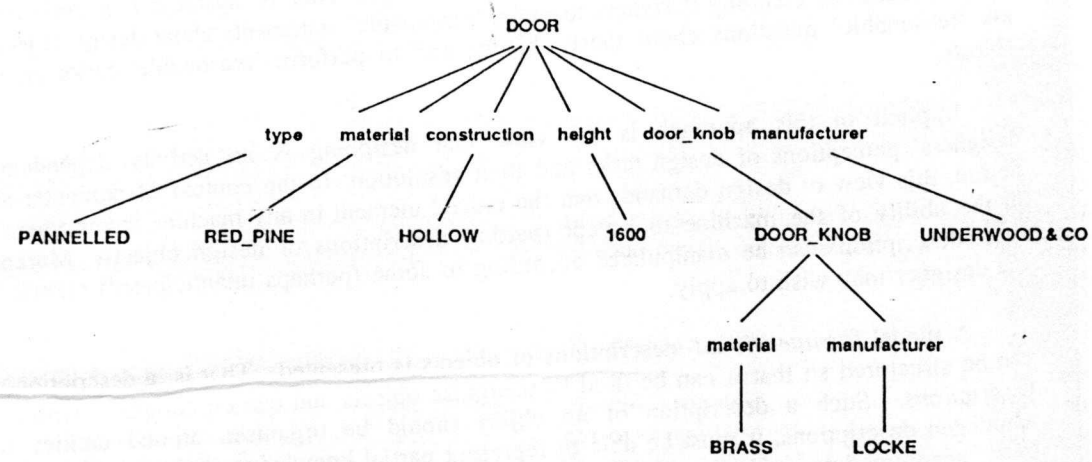

Figure 1 Pictorial representation of the description of a kind:
Kinds and fillers are represented by vertices and slots by labelled edges. A description corresponds to a rooted labelled graph.

2.2. Fillers

In principle anything can be a filler, some possibilities of which are indicated by the following table.

CATEGORY	FILLER
kinds	DOOR, RED, TUESDAY, *
numbers	200, 3.1415923
beliefs	yes, no, dont_know
statements	'follows from rule 5(a)'
indirections	SEM1:party_wall, SHAPE:origin
undefined	[ ]
exclusion	χ

The above table presents some interesting examples. For instance, RED and TUESDAY that do not appear to be kinds can be considered as kinds with no explicit description. The asterisk symbol ' * ' is a special filler to denote an unnamed kind. It serves as a useful device when we want to describe kinds without having to explicitly label every kind that occurs in a description. The empty list symbol '[ ]' is a special filler to denote unspecified fillers. Since designing is a constructive activity of creating and updating design descriptions, there may be situations when one is undecided about what fillers to employ for a given slot. The 'χ' symbol is a special filler that is useful when we want to stipulate that some property or feature is to be excluded from the description of a kind. A case in point is when a kind inherits (see next section) part of its description from another kind and some of the inherited description is to be excluded.

Indirections act as permanent pointers to descriptions that occur elsewhere and which are independent of possible changes to the referred kind. For example, consider the description of a semi-detached house. It has two houses, say one on the left and the other on the right. Clearly, right wall of the left house and the left wall of the right house refer to the same entity, namely, the party wall of the semi-detached house. Moreover no matter what changes are made to the description of the semi-detached house we require the two walls to continue to refer to the same party-wall. Indirections provide the means to ensure this.

The above list is necessarily incomplete. Other possibilities include ranges, procedures, decision rules, sets of values etc that can be fillers. For instance, suppose that a certain type of manufactured bath comes in three colours, say avocado, white and grey. Then, the colour part of the description of the kind BATH could be the set { AVOCADO, WHITE, GREY }.

It should be noted that the list of acceptable fillers for any modelling system depends on the available interpreters to evaluate the descriptions.

3. Inheritance of partial descriptions

Often in the course of designing one is interested in a collection of similar objects - for example, all the doors on a new housing estate. There are things that can be true of each - for example, that they are all made by the same manufacturer. Within this collection of doors there may be things that are different from one another- again for example, some may be internal doors and others external doors.

This similarity between partial object descriptions can be considered as an *inheritance* of properties from one object to another. Two sorts of inheritances are allowed. The first is termed a *variant* of a kind which provides a view of an object as seen with respect to the attributes of another kind. That is, a form of (super)kind to (sub)kind relationship. The second is termed an *instance* of a kind which occurs when two kinds contain references to the same object, and when the description of one of the kinds is altered without changing the other. The distinction between the two forms of inheritances lies in the roles played by the slots in the description of a kind.

3.1. Inheritance through variants

Consider the archetypal kinds MAN and MUSICIAN. Clearly, some individual, say BEETHOVEN, is both. It is possible for both man and musician to have slots with the same name - for instance, 'preferences', where for the former it denotes manly preferences and for the latter specifically to musical preference. Thus, in the description of BEETHOVEN, the preference slots are disambiguated only when the context in which they occur are specified. This process of disambiguating the slots is termed as *tagging*.

There are two rules of inheritance which are stated as follows:

1. *A kind inherits the parts of each of its super-kinds*

Suppose P $\Rightarrow$ Q $\Rightarrow$ R where $\Rightarrow$ denotes the direction of inheritance between kinds. Then, by the above rule, Q inherits the slots (and their associated fillers) from P, and R inherits the slots from both Q and P.

2. *The parts inherited from its nearest super-kind masks those inherited from any other kind.*

Consider figure 2 which has two distinct inheritance paths: P $\Rightarrow$ Q $\Rightarrow$ R and S $\Rightarrow$ R. Both Q and S have parts identified have parts identified by the slot named b. The part in Q identified by c is actually inherited from P but has been altered by associating it with a different filler G $\neq$ F. By rule 1, R inherits the b slot from both Q and S. To distinguish between them, the slots are tagged as b(Q) and b(S) respectively. In the second case, Q's altered inherited slot c(P) masks the c slot of P that R would otherwise inherit since Q is an immediate super-kind of R whereas P is a super-kind at one remove. Thus, 'nearness' in the second rule is a measure of the distance of inheritance between a kind and its variant. The complete slots - tagged with the name of its parent kind - of R are:

a(R) - R's own a slot
b(Q)

c(P) - slot c from P with filler G as a result of a change to the description of Q and so, masks slot c with filler F in P's own description

f(P)

b(S)

The exclusion part d = χ in the description of R ensures that the d slot in the description of S is not inherited.

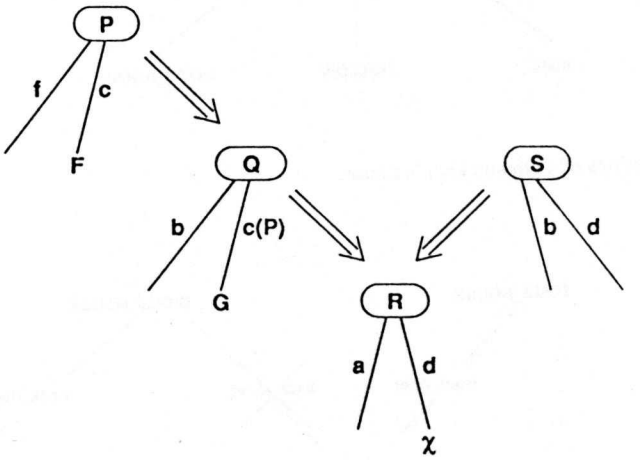

Figure 2 Illustrating the inheritance between kinds:
R inherits all slot-filler parts from the other kinds except c = F in the description of P which is masked by the part c(P) = G in the description of Q, and slot d in the description of S which is excluded by d(R) = χ.

3.2. Inheritance through instances

Instances occur when the description of a kind is updated. Consider figure 3 which represents the description of Tom and Dicks' houses. Notice that the kinds TOMS-HOUSE and DICKS-HOUSE both contain references to the same kind DOOR.

Suppose the material of the door-knob in Dick's house is stated to be made of brass but at the same time the description of Tom's house is required to remain unaltered. This is achieved by creating instances of DOOR and DOOR-KNOB as illustrated in figure 4.

Here the back_door slot of Dick's house now refers to the instance DOOR-1 whose door_knob slot masks that in DOOR (by inheritance rule 2 above). The material slot of the door_knob is inserted in the description of a new instance of DOOR_KNOB, namely DOOR_KNOB-1. Instances are identified by the name of the parent kind together with an instance number separated by the instance operator '-' The arrow '$\Rightarrow$' is used to denote inheritance between a kind and its instances. Instances inherit all but the altered property from their parent kind. That is, an instance masks all inherited slots with the same name. In other words, the tag of any slot in the description of an instance is the tag of any slot in the

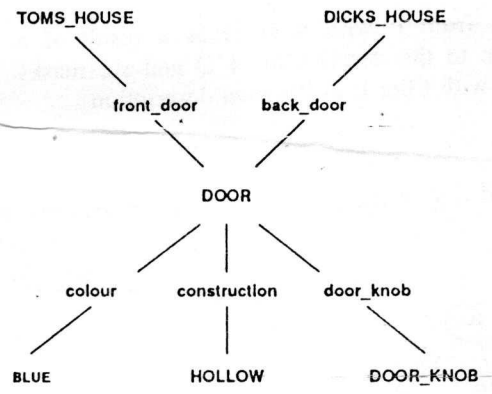

Figure 3 Partial descriptions of Tom and Dick's houses

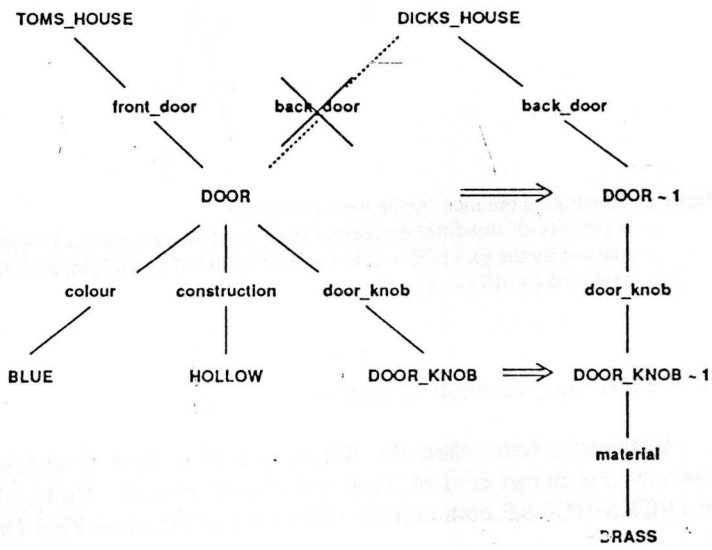

Figure 4 Updating the database shown in figure 3
× marks the deleted slot.
⇒ indicates inheritance between a kind and its instance.

description of its parent super-kind.

A similar situation occurs when a slot inherited from a kind is updated. Figure 5 illustrates this where the front door of Tom's house and back door of Dick's house are both variants of an archetypal door. The figure also illustrates the use of the special '*' filler. Notice that door_knob slot added to the back door of Dick's house is explicitly tagged with the name of its parent kind, namely DOOR. This ensures that unwanted door_knob part

which would otherwise be inherited from DOOR is masked and thus maintains part update consistency.

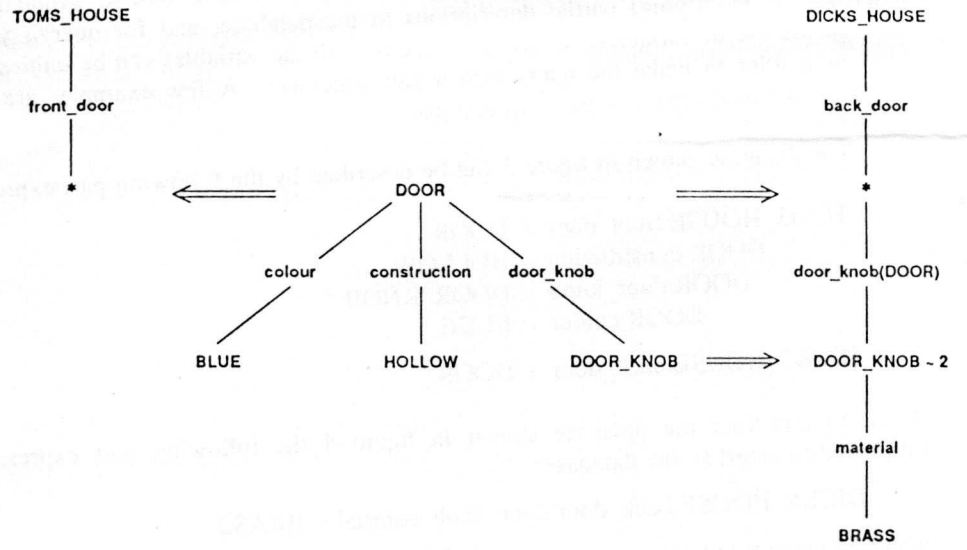

Figure 5 Updating the filler of a slot inherited from a variant.
The updated door_knob slot has been tagged by the parent kind DOOR.
* indicates an unnamed kind or filler.

4. Creating and updating a model

4.1. Part expressions

Recall that any description can be represented as a labelled digraph in which a *path* corresponds to a conjunction of slots and kinds relating a kind and a filler. A model is a collection (possibly disjointed) of such labelled digraphs. It is convenient to treat the model as a database consisting of *parts* expressed as identities of the form:

K:s = F

Part identities are tag-wise parametric. That is, the identity K:s = F is in the database whenever kind K has in its description a slot s (inherited or otherwise) with filler F. K:s stands for K:s(T) where tag T need not equal K.

It is easy to show how - by a simple traversal algorithm - a kind is related to a filler. In fact, updating the model corresponds to following paths in these digraphs coupled with the appropriate addition and deletion of labelled edges. A path in a labelled digraph can be expressed as *part expressions*:

$K:s_0: \ldots :s_n = F$

A part expression is deemed true whenever there are fillers $X_1, \ldots, X_n$ such that part identities $K:s_0 = X_1, \ldots, X_n:s_n = F$ are in from the database.

Part expressions are interesting because they provide a unified syntactic format for asserting (or modifying) partial descriptions to the database, and for querying the database through the use of variables. A query succeeds if all the variables can be *unified* to slots or a kind or a filler to make the query into a part statement. A few examples are presented to illustrate this dual aspect of part expressions.

The database shown in figure 3 can be described by the following part expressions.

```
TOMS_HOUSE:front_door = DOOR
      DOOR:construction = HOLLOW
       DOOR:door_knob = DOOR_KNOB
           DOOR:colour = BLUE

DICKS_HOUSE:back_door = DOOR
```

Thus, to construct the database shown in figure 4 the following part expression can be employed to assert to the database:

DICKS_HOUSE:back_door:door_knob:material = BRASS

which is made true by creating new instances of DOOR and DOOR_KNOB.

In a similar vein part expressions can be employed to query the database such as:

TOMS_HOUSE:front_door:construction = ?

where ? denotes a variable which is instantiated to HOLLOW. And,

DICKS_HOUSE:?:door_knob:material = BRASS

will instantiate the variable ? to the slot back_door. More complicated examples are given in [3].

5. Modelling algorithms

In this section a brief outline of some of the algorithms for maintaining the modelling enviornment are described. A rigorous treatment is given in [4].

The modelling environment, at any given time, maintains the current list of known kinds, their slot-filler pairs, the lists of pair-wise kind-variant and kind-instance relationships. The slot-filler pairs are partitioned into two sets, (1) corresponding to simple slot-filler relationships; and (2) corresponding to tagged slot-filler relationships. The latter corresponds to inherited slots that have been modified in the description of the variant. All other slots inherited by a kind can be recovered from the kind-variant inheritance relationships and hence need not be stored. All this information is kept essentially in the form of relational tables. Initially the model is empty. The modelling environment also maintains a collection of evaluators to process fillers of various sorts. While in practice the list of available filler evaluation procedures is finite, for the purposes of explanation it suffices to assume that evaluators exist for each sort of filler.

5.1. Extracting descriptions

The algorithm is essentially a backtrack search performed on a labelled digraph rooted at the vertex corresponding to the given kind. There are two sorts of descriptions: unevaluated and evaluated descriptions. In an unevaluated description a search path terminates if the filler is not a kind. In an evaluated description all fillers are evaluated. For instance some fillers may represent messages to invoke procedures that return values. Others, such as indirections, represent references to kinds. Whenever the search finds a filler to be evaluated it does so, and if the value returned is a kind it continues the search, otherwise the particular search path terminates with the evaluated filler.

The description algorithm can be tweaked to provide various forms of partial descriptions of a kind such as the listing the parts explicitly attached to a kind, the description of a kind as seen with respect to the parts inherited from another kind and so on.

5.2. Updating the database

We consider the following four operations: (i) adding a simple part identity; (ii) removing a simple part identity; (iii) altering an inheritance relationship; and (iv) altering a composite part statement.

5.2.1. Adding a simple part identity

If the referred kind K is not in the database, then the list of kinds is updated, and the K-s-F relation appended to the slot-filler table. Otherwise, there two cases to consider according as the tag of s equals parent kind of K or not. It is important to test against the parent kind since only instances have parents that are not themselves. In both cases replace any existing K-s-G relationship by the new part. If equality does not hold, then the part corresponds to a modification of an inherited slot and is tagged as such.

5.2.2. Removing a simple part identity

If F is specified removing a part identity is straightforward. The only thing to note is that if s refers to an inherited slot then the exclusion relationship K-s-χ must be added to the database provided $F \neq \chi$. If F is unspecified then all K-s-? relationships are determined and each in turn is removed.

5.2.3. Altering an inheritance relationship

Adding an inheritance relation is trivial. In removing an inheritance relationship, the effect is to 'freeze' the description inherited from K by V. That is, each V-s-F part inherited from K must be asserted as a tagged slot-filler relationship to the database before the K-V inheritance link is severed.

5.2.4. Altering a composite part statement

Instances are created when a composite part is added or removed from the description of a kind. A composite part statement corresponds to an expression of the form $K:s_0: \ldots :s_n = F$, $n > 0$. Here a backtrack search with filler evaluation has to be performed in case the

expression implicitly refers to inherited slots or to fillers that evaluate to kinds. The simple case is when each intermediate filler is a kind. In this case a new instance of each intermediate filler is created to which a slot-filler relationship is added. The last part identity say $K_n^- : s_n = F$ where K_n^- is the newly created instance is either added or removed according to the nature of the alteration. The operations involved correspond to one of the three described above.

Clearly, updating the database may cause an explosion of instances and slot-filler relationships and checks can be devised to minimise this. Beside this and the case when intermediate fillers are not simple kinds, other problems can arise. For example, when descriptions of kinds are recursive; that is, the search may revisit a vertex on the search path. For a fuller discussion on updating composite parts the reader is referred to the paper cited above [4].

6. Drawings

The visible effort in any design activity is spent on design drawings. This implies that CAD environments must provide graphics facilities that are capable of carrying and manipulating descriptions of the geometry of spatial objects. Space considerations do not permit further elaboration. However, it has been [5, 6] shown that parametrised spatial descriptions can be represented as kind-slot-filler structures. Further, it is possible to describe geometrical operations as sequences of part expressions.

7. Implementation

A prototype modelling system MOLE [7] based on the ideas presented here has been implemented in an enhanced version of C-Prolog dynamically linked to C routines for faster i/o and graphics. MOLE has been applied to shape grammars, to kitchen space planning problems and to space recognition problems [8].

8. Conclusion

Some of the salient features of a representation for modelling design descriptions has been presented. Design descriptions can be constructed from simple building blocks, through part expressions. One of our motivations for this work rests on the belief that it is possible to place such a modelling environment within systems that are capable of handling the conventions and rules that designers *choose* to adopt. While it is not suggested that designers produce designs by some sort of top-down iterative process of creating and altering design descriptions, the use of computers to assist in the design activity does impose a certain discipline on them. A consequence is that designers should be be aware - self-consciously or otherwise - of the kinds of 'knowledge' they possess (and employ) for their designs. Equally, as designers become progressively 'computer literate', it becomes imperative that appropriate software is available to suit their needs. One result of this work is the raised hope that, in the near future, aspects of design activity can be ably assisted by 'design literate' software.

Acknowledgements

I am grateful to Aart Bijl for his encouragement and support, and for his many constructive discussions during this research. The research reported in this paper has been carried out within the context of research programmes at EdCAAD funded by the Science and Engineering Research Council and the ACORD project funded by the CEC/Esprit programme.

References

1 Minsky, M., "A framework for representing knowledge", in P. Winston (ed.), *The psychology of computer vision*, McGraw-Hill, New York, 1975.

2 Woods, W. A., "What's in a link?", in D. G. Bobrow and A. Collins (eds.), *Representation and understanding*, McGraw-Hill, New York, 1975.

3 Krishnamurti, R., "The MOLE Picture Book: on a logic for design", *Design Computing*, Vol. 1 (1986), forthcoming.

4 Krishnamurti, R., "Representing design knowledge", *Planning and Design*, Vol. 13 (1986), forthcoming.

5 Szalapaj P. J., and A. Bijl, "Knowing where to draw the line" *Proc. IFIP Working Conference on CAD*, Budapest, Sep. 1984.

6 Bijl, A., and P. J. Szalapaj, "Saying what you want in words and pictures", *Proc. INTERACT'84*, London, Sep. 1984.

7 Tweed, A. C., "The MOLE User's Manual", Technical report, EdCAAD, University of Edinburgh, 1985.

8 Tweed, A. C., "The MOLE Exercise Book", Technical report, EdCAAD, University of Edinburgh, 1986.

Author: Ramesh Krishnamurti

Address: EdCAAD, University of Edinburgh
20 Chambers Street
Edinburgh EH1 1JZ
Scotland

Phone: (031) 667 1011 x 4566 Telex: 727 442 UNIVED G