The LEGOL 1 Prototype system and language†

R. K. Stamper
LSE Houghton Street, London WC2

Research into the problems of information analysis is being carried out using legislation as experimental material. A body of administrative law may define the tasks of a man/machine information system and, by attempting to construct a formalism (LEGOL), in which such law may be expressed, it is possible to study the logical structure of formal systems. A prototype version of the formalism and of its interpreter system are described. Theoretical problems are raised in the fields of data base semantics, very high level languages, use of relational data bases, design decision-making and of the methodology of research in the field of information analysis.

(Received April 1976)

The problems of developing computer applications will shift from their present locus, near the machine, as computer hardware falls in price and increases in reliability and as systems software and high level programming languages increase in power and efficiency. Increasingly they will be centred upon the organisation. The improvement of technology will make it feasible to attempt much larger and more complex applications than before. Unfortunately, the potential of computers will not be realised whilst we lack the necessary insight into the ways in which information is used to run organisations. It is now relatively easy to program the computer. What we find it difficult to do, is to decide what data processing must be performed. This, the information analysis problem, is increasing in difficulty with the complexity of the organisational tasks to which the computer is applied.

With the aim of understanding problems in information analysis, an attempt is being made to create a formalism (LEGOL) in which complex rules and regulations may be expressed. A prototype system has been constructed for interpreting LEGOL 1, a first subset of the formalism. LEGOL can be used as a very high level language for defining data processing operations and an interpreter can be used to simulate the logic of the system.

The project described below was conceived as a way of approaching problems in information analysis in a scientific manner. The hallmarks of a good scientific theory are that it is succinct whilst being of great generality and that it can be rigorously tested; the general theory must lead to particular statements about the real world against which they can be compared. The importance of this scientific approach is clear when one realises that information analysis is the task of ensuring that the formal data processing system is accurately and reliably linked to the real world which the organisation is supposed to monitor and control. Computer science has been preoccupied with the formal manipulation of symbolic data and with the machinery to perform these operations. Computer science now needs to develop an appropriate theory of the relationships between data and what they represent. Until it does so we shall continue to build efficient computer systems which are not effective organisationally (Land, 1974). The task which distinguishes the role of the systems analyst from that of a programmer is to establish these essential links between symbol and reality. The missing theory is that of systems analysis.

The LEGOL language and system, described below, might be seen only as an attempt to produce another high level language and corresponding interpreter. It is hoped that they are more. They are intended to show how we might shift the locus of our attention away from the relatively familiar problems of computer hardware and software into the murkier regions of formal organisational systems where most computers supposedly find their economic justification.

1. Data processing and legislation

After examining a number of alternatives, it was decided to use statute law and statutory instruments as the experimental material for research. The reasons are discussed in Stamper (1973). Legislation serves, in fact, to define what should be done in a formal data processing system. Legislation does not, as a rule, specify how the data are physically to be processed and stored, instead, it gives an almost minimal statement of the logically necessary data processing.

The people who obey the legislation, either to obtain their rights or to perform their duties, actually process the data. In doing so, they play their roles in an information system. In data processing terms such a system may be very complex but it is not defined by stating how the data should be processed. It is defined by rules which say what should happen to people, their possessions and their actions in the real world. Certain, necessary data processing is implied by these rules. Firstly, to apply the legislation, we must attach names, numbers or other symbolic labels to the relevant people, objects or activities. We must then manipulate these symbols as specified in the legislation itself. Finally, the results of the decisions must be communicated to the people who will act upon them, thus completing the cycle from object system, through information system, back to object system.

In taxation, social security, licensing and so on, a high proportion of these rules are cut-and-dried and therefore ideally suited to the computer. Where the rules are not entirely formal they imply an interaction between a formal system and some human decisionmaker. This interaction is itself an important feature of a typical data processing system. Thus by discovering a formalism in which legislation can be expressed, we shall learn how to define precisely one important class of data processing systems.

The aim is to produce a formalism sufficiently general to encompass any legislation. There is at least a reasonable chance of being able to produce a good, special formalism, hence there is a good chance of the research producing results of practical value. However, scientific discovery depends upon stretching a hypothesis to its limits to discover where it fails. At the point of failure discoveries are made.

Generality is of greater value if combined with brevity. Scientifically, a theory is obviously more informative if it is both brief and general instead of being either limited or diffuse.

†Supported by the Science Research Council with assistance from IBM (UK) Scientific Centre, Peterlee.

There are other reasons for seeking these quantities in the formalism being constructed. If the formalism is used for systems definition, it will be more valuable if it is general enough to encompass a wide range of data processing systems, and its power of exposition will be enhanced by its brevity. One has only to think of the length and diffuseness of a typical system definition to realise how valuable a concise problem statement would be. Hence, the chosen research goals combine a scientific ambition and an important practical objective.

A fully-fledged, legally orientated language will take many years to evolve. So far, a rudimentary version of the language has been defined and an interpreter for it has been constructed. In doing this, some fundamental problems of information analysis have been exposed with a clarity which gives us a good chance of solving them. Currently, a more elaborated version of the language and prototype system are being developed. The first prototype version of the language and system are described below, key theoretical problems encountered and new directions of work are also discussed.

2. Structure

Structure is the key to economy of expression. To interpret ordinary statements and commands in prose, a person relies upon his knowledge of the world or, to be more precise, upon his knowledge of how the names ascribed to things in the world are used in relation to these things and to one another. In ordinary discourse we use language with a richness of metaphor and allusion which are not employed in legislation. Legal prose is intended to express ideas which are relatively simple from a semantic point of view. Despite its apparent complexity, legal prose in the hands of skilled draughtsmen is purged of those logical complexities and ambiguities which make the rigorous interpretation of ordinary language difficult, if not impossible. The logical and semantic structures employed in legal prose being relatively simple ones, it is hoped to explicate them fully and employ them within the LEGOL system for interpreting statements in the formalism. This is much less ambitious than attempting to model in a computer the understanding of ordinary language.

The explicit structure of a body of rules consists of first order rules which say what should happen in the 'real world'; these are put into a sequence (which may be conditional) by second order rules. These denotative rules are supported by effective prescriptions telling of the inducements to conform to them. Implicitly, there is also the semantic structure. Of these four components, the first prototype language and system handle only the first order rules and a part of the semantic structure concerned with the entities spoken about in the first order rules. These will be described.

The description begins with an account of the semantic structuring. To use the formalism in practice one would also begin with the semantic part of the problem. This corresponds to solving the problem of the data base controller setting down a conceptual schema for an application (ANSI-SPARC, 1975). Judging by the work done on LEGOL so far, the problems of conceptual data definition are in danger of being misunderstood if one adopts the computer programmer's frame of reference; this is not appropriate for information analysis. Programming is concerned with symbols and their manipulation by the machine. The data base problem, from this point of view, is about how to label, store, retrieve and manipulate these data. Information analysis is quite different as it concerns the people, things and actions to which the data refer. From this point of view, the data base problem is one of locating people, things and actions so that the data processing system can monitor and influence them. In the LEGOL project the problem of a conceptual schema has been approached as a problem of information analysis. We must define what may or may not be said about the world within the formal system, so that it will always employ data in meaningful and self-consistent ways. The problems of labelling, storing and retrieving the data are not examined. That is not to dismiss the programming problems but to say that they are of a different kind from those of information analysis and that we simplify the overall problem by distinguishing two subproblems. By doing this we also hope to highlight aspects of the conceptual schema ignored by concentrating on the programming problem.

To illustrate the way in which LEGOL handles the semantic model or conceptual schema, consider legislation intended to encourage the construction of factories and other buildings for industrial use. Some allowance against taxation will be made to certain companies in relation to the industrial buildings for which they are responsible. Whatever rules we devise to obtain the desired economic effect, we shall need to be able to talk about certain things. The concepts that we shall employ vary considerably in their semantic standing (Stamper, 1973). Perhaps the simplest concept in this example is that of a *building*. A physical object which is large enough to be seen clearly yet not so large that one cannot discern it on a single occasion, can be related to the data which name it and describe it in a fairly simple way. If necessary the object may be defined ostensibly: we may take someone to see the building. That is a reliable way of establishing the relationship between a word and a thing. The words associated with the building will serve two major functions. One of these is to identify the building, the other is to report properties of the building. The name of an object such as a building is a unique label which may be used for identifying it. *When we talk about an identifier in the context of LEGOL, we mean an element of data which may be used by a person to find the real thing, an actual factory for example, whereas in the programming sense an identifier is an element of data used to locate other data elements.* (It may seem eccentric to a non-computer specialist to be turning language onto its feet in this way but, writing for the computer world where language so frequently stands on its head, one must be explicit when turning it the normal way round.)

Not all concepts that are semantically treated as objects will correspond to physical objects like the factory. For example, the owner of the factory may be a person whose hand you might shake and whose name you might be taught by ostensive definition, but the owner is more likely to be a *company* and a company is a legal fiction. A company will have a name and it will be treated conceptually as though it were a person. Physical objects such as buildings, and by analogy objects such as companies, will be identified by unique names.

In the taxation example, it is necessary to identify more complex entities such as concrete and abstract relationships, e.g. *use* and *ownership*. Each of these links a building with a company. One may observe the relationship *use*, whereas the relationship *ownership* is a legal fiction, an abstraction, which effectively sums up the consequences of a body of law. To identify a relationship, we must supply the names of the objects which are related and the name of the manner in which they are related, but these identifiers will not suffice. Time plays an important role in the identifying of certain relationships. Most of the entities which are the subject matter of the law will exist for a definite period. In the case of an object, the name in the LEGOL system will be unique for all time. Hence it is superfluous to know the period of existence of the object, if we are given its unique name. For a relationship, however, it is conceivable that a company might use a building for the same purpose for two quite separate periods. We may wish to give tax relief for each of these periods but not for the intervening time. To identify the relationship *use*, we must therefore specify the start and end times for the existence of the relationship.

Given that we have succeeded in identifying an entity, whether object, relationship or some other type, we can discover some of its properties by subjecting it to operational procedures.

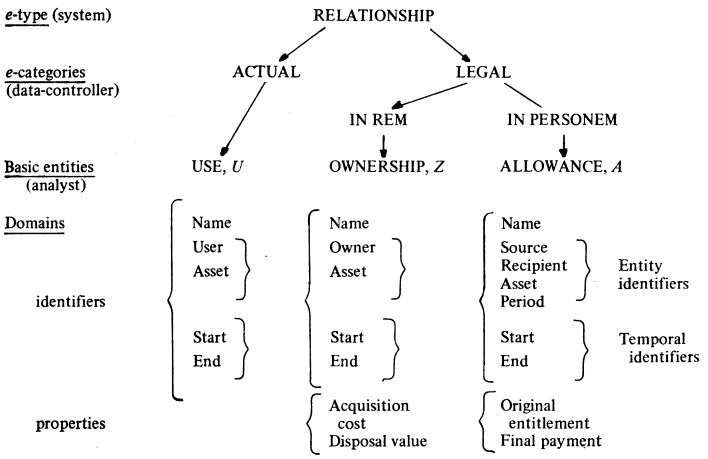

Fig. 1 A hierarchy of entity structures

Some properties change with time but these are semantically more complicated than those which are invariant. For a given entity the properties that are relevant will vary from application to application. At the most general level of analysis however, we are able to distinguish what we call *entity types*. Examples are *object* and *relationship*. Each of these *e*-types defines a particular pattern of identifying information associated with an invariant set of properties. The LEGOL formalism compels the data base controller to define every entity using one of the built-in patterns called entity types.

The data controller is permitted to define subclasses of any entity type which are relevant to the application with which he is dealing. These subclasses are called *entity categories*. Within the framework of entity categories, the application analyst may define his own *basic entities*. An example is shown in **Fig. 1.** The hierarchy of categories fills out the bare framework of an entity type until basic entities may be defined at the terminal nodes.

In relational data base terms (Codd, 1970), each basic entity may be regarded as a relation. The property domain structure for a relation representing a basic entity results from the analyst's choice of *e*-category, whilst the identifier structure is ultimately supplied by the *e*-type chosen by the data controller. The definitions of *e*-categories and basic entities apply only within some prescribed context, such as an area of law, whereas the *e*-types impose their patterns upon all entities. Thus the objects to which the data processing system may refer are constrained by the more general structure evolved by the data controller. This structure can then be employed by the formal system to interpret expressions involving the entities referred to by the analyst.

The entity structure is one part of the semantic information which is included in the first, prototype, LEGOL system. This early system employs a set of entity types which is now known to be inadequate. Nonetheless, the method of building this structure appears to be valid. A major objective in the continuing research is to discover an adequate set of entity types.

3. First order rules

Using a set of definitions provided by the data controller, the application analyst can translate legal rules into the LEGOL formalism, or express himself directly in the LEGOL formalism. The law consists of a series of Statements saying what should happen in certain well-defined conditions. As an artificial example consider the following:

> 'Capital Depreciation Act 1984 c. 11, Sub-Section 23.—(1): *Where a company owns an entitling interest* as mentioned in Paragraph (3) below *in a building which is occupied for an industrial use* as specified in Paragraph (4) below *by that company or by another company, there shall be made to the company owning the entitling interest*, for the chargeable period mentioned in Paragraph (5) below, *an allowance* (*to be known as 'a capital allowance'*)'.

The rule itself has been italicised but it starts with a reference which may be treated as a label for a particular context and it includes references to other rules indicating the sequence in which they should be applied.

The LEGOL expressions used to translate the example above are:

$$Z'(x, b) \,\&\, U'(y, b) \rightarrow A(g, x, b, t)$$
$$A <= \text{'Capital Allowance'} \ .$$

Where Z' is a relation containing details of property interests which attract the allowance, A, for the periods of recognised industrial use, U'.

Each of these expressions contains an arrow sign. The arrow means 'if . . . then do . . .'. In the first expression, the condition is that the relevant ownership should coincide with a recognised use for the building. The expression looks like a conjunction of two statements in predicate algebra but it is more complicated as it must take account of the times when the ownership and the use exist. The action to be taken when these conditions are satisfied is to create the allowance, A, to which the prescriptive arrow points. What happens is that we give values to instances of the abstract entity type called an *allowance*. The allowance involves all the domains listed in Fig. 1. In the second expression the allowance, A, is referred to in a different sense; here we are only concerned with the name of the allowance which we give the literal value 'capital allowance'. This is an example of one of the many ways in which symbols in LEGOL expressions may be used with more than one meaning, the correct one being resolved by the context.

As a further example, consider the rule which establishes when a building is used industrially. The rule is subsection (4) which might read as follows:

> 'For the purpose of this law 'a building occupied for industrial use' means a building occupied for the purposes of one or more of the trades listed in Schedule D.'

Schedule D reads:

> 'Subject to amendment by the Secretary of State under the provisions of section 30 of this Act the trades regarded as "industrial" for the purposes of this Act are
>
> Baking
> Milling
> . . .'

The LEGOL expression for subsection (4) will be:

$$U(x, b) \in \{I\} \rightarrow U'(x, b) \ .$$

'I' denotes a set of industrial uses which are listed in Schedule D. In this expression it appears that U is not a predicate formula but represents something which is the member of the set $\{I\}$. In fact, the context indicates that we are talking about the name of the type of use and the formula says that if this name is a member of the set of names of approved uses, $\{I\}$, then we are to call the use U', an industrial use.

These methods of exploiting context are not unlike those evolved in natural language. By using context to resolve ambiguity, the formalism can be made more compact and, by imitating natural language, it should be easier to translate from natural language into formalism.

4. Local or temporary entities

Every entity category or basic entity must be defined within some specified context. A context may be a branch of law, it may be a certain Act or Statutory Instrument but it may be only a Section of an Act. Within the context for which it is defined, an entity must be employed with its limited meaning

and one of the objects behind the system is to ensure that this is so.

Normally first order rules are written in terms of basic entities but this would be a severe limitation. One may wish to refer to some more general entity. This can be done by using an entity category to form a temporary entity for use as a working variable. In the example we are considering, subsection (5) defines a period for which the allowance is to be made and specifies with what company, what building and what set of accounts this period shall be associated. The resulting prescription:

$$\langle \text{expression} \rangle \rightarrow p(x, b, t)$$

combines a number of entities in a complicated structure called ⟨expression⟩ from which some part is abstracted and called a period, p, where the entity category *period* is employed in the context of this section. Another subsection explains how the amount of the allowance is to be computed according to the duration of this period, p. The variable p may be used to transfer a meaningful structure from one rule to another but, because it is a temporary entity, it has no meaning outside the immediate context specified by the analyst. The label 'p' is supplied by the analyst.

A similar kind of local entity is illustrated by the formula:

$$U(x, b) \in \{I\} \rightarrow U'(x, b)$$

in which U' is defined in terms of U. The semantic structure of this local variable is taken from a basic entity. Its symbolic label is obtained merely by attaching a superscript to the label for the basic entity. Once again, this kind of entity has no meaning outside a narrow context.

Superficially, this may look just like the handling of temporary variables in a program, but it is much more elaborate. Remember that the symbols in the formalism do not refer to data inside a computer but refer to actual things in the world. Temporary entities are components of the world of physical objects or legal fictions which we do not wish to distinguish by permanent labels, yet they exhibit semantic structures assigned to permanent entities in our universe of discourse.

5. Limitations upon the first prototype

The first prototype version of the LEGOL formalism and its associated system is intended to answer certain questions of feasibility. The major features incorporated are those described above: the entity structuring and the writing of first order rules. These features of the formalism correspond, on the systems side, to the structuring of a data base and to the data manipulation implied by the legal prescriptions. Other features of the formalism and system have been left for the second prototype. The most important of these concern the semantic structures for domains and the treatment of second order rules, both of which were only mentioned above.

Whilst the first prototype system handles the semantics of identifiers, it does not include any means of specifying the semantics of properties. Once one has located the entity, object or relation in the real world, one assigns words or numbers to describe the entity. The formalism and the system should control the meanings of these property domains. This part of the semantic structuring will eventually handle such things as scales, units, operational definitions used in different contexts, ranges, accuracy and so on. This semantic information plays a less direct role in the problem of interpreting LEGOL expressions, hence its omission from the first prototype system.

Second order rules, which determine the sequence in which first order rules are applied, will be an important problem when handling a large number of first order rules. It was quite unrealistic to construct a first prototype system to handle anything but fairly small numbers of first order rules because of the complexity of the interpreter. Hence the omission of second order rules.

Ultimately, it will be important to make the formalism and the system easy to use. These objectives have however been given low priority in the design of the first prototype. The formalism will strike the reader as esoteric. It was felt that by concentrating initially upon the logical structure required by the formalism and leaving its appearance until later, priorities would be in the right order. Similarly, the system has been designed to be easy to evolve and modify rather than easy to operate. It has been implemented using a powerful macroprocessor system (Mandil, 1974) and a relational data base system (Todd, 1976). The interface with the user is not in a unified LEGOL formalism because it also exploits features of both the underlying systems. By accepting these limitations, the available resources have been used to demonstrate the feasibility of constructing, for the first crude version of the LEGOL formalism, an interpreter which is capable of being readily extended to handle the anticipated refinements in the formalism.

6. The relational model for data

Although the LEGOL formalism refers to things in the real world, it also implies how the data for identifying and describing these things should be organised and manipulated. It is the task of the LEGOL system to give effect to these implications. The strategy for achieving this is now described.

The relational model of data is the natural foundation for a part of the system dealing with entities. When one refers to an entity in natural language to tell someone about it or ask them to perform some action in relation to it, one refers to the whole entity not merely to its name; the formalism was therefore conceived on the assumption that a symbolic label for an entity would designate the entity as a whole. The implication of this approach for data processing is that within the LEGOL system, an entity label should stand for all the data relevant to that entity, encompassing all the relevant domains. Similarly, in the context of legislation, one lays down rules which must be applied to all buildings, say, or all companies, not to isolated instances of them. For the system, this implies that the relation label should refer, not to an individual entity but to all the entities of a given type, to all buildings or to all companies. It was therefore envisaged that in the data processing system, the surrogate for an entity should be a relational structure encompassing all the relevant domains, containing a tuple corresponding to each individual entity. The relational model for the data is never abandoned even when we wish to refer to particular properties. As in ordinary speech, when we refer to a particular property, such as a cost, the LEGOL formalism carries with it the implication that it is a cost pertaining to some entity and that entity has various other relevant properties. This is accomplished in the system by retaining all data within the relational structure and simply *pointing* to those aspects which are of immediate relevance in the expression currently being interpreted.

From a data processing point of view, this approach may seem very wasteful, but the object of the LEGOL formalism and system is not to be able to process data efficiently. It is to specify real-world activities which imply that certain data processing should be carried out. The LEGOL definition of the problem is only the starting point for the design of a data processing system. The LEGOL system will be adequate if it is capable of simulating the behaviour of the required data processing system.

7. The semantic data base

An information analyst who is defining a data processing system, is more concerned with information about the data than with data themselves. The LEGOL system might be used to assist him by storing this information. Some would be held in the hierarchical structure extending from the few entity types, becoming increasingly specific through the entity categories,

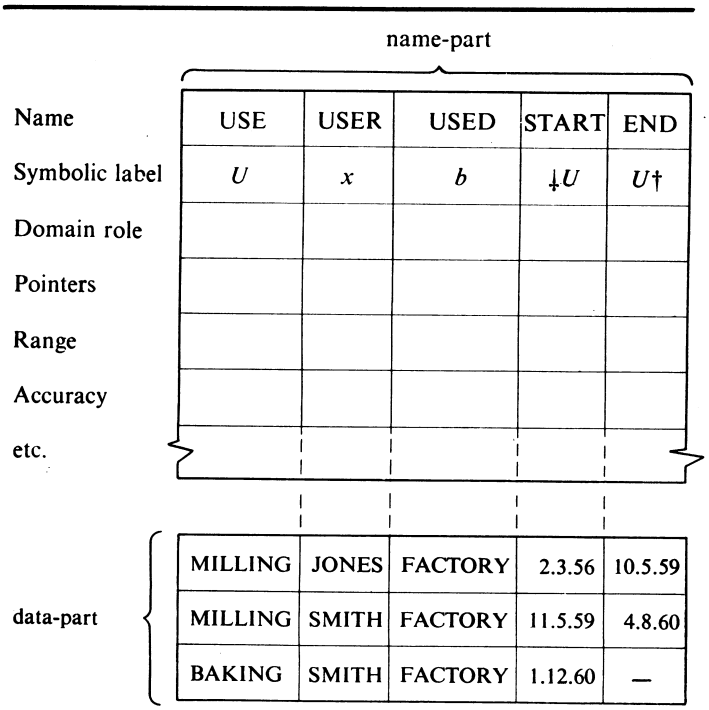

Fig. 2 The two-part relation, $U(x, b)$ in the semantic data base

terminating with the basic entities which may be associated directly with relational structures containing data. Then, at the level of basic entities, more information will be organised as another relation. This is shown in **Fig. 2.**

Each domain of the *d*-part of this two-part relation is matched by a tuple of the *n*-part containing names, labels and other semantic information. One of the functions of the more general part of the semantic structure is to constrain the entries which the analyst may insert into the *n*-part associated with the basic entity. This can be illustrated in terms of the symbolic labels. The system restricts the user to one suitable entity type called a *relationship* which is identified by a name for the character of the relationship plus a list of names of the entities related plus the start and end times for the duration of the relationship. This may be indicated by the 'template':

e-type (*, *n* entities, ↓*, *†) Relationship

where the asterisk corresponds to the name of the relationship and the asterisks accompanied by the dagger symbols stands for the two time identifiers. The data controller, in the context of this law, divides relationships into two categories *actual* and *legal* and indicates that legal relationships have two properties associated with them being sums of money associated with the start and the end of the relationships, whereas actual relationships have no properties relevant to the context of application. These would be represented in the entity category hierarchy by 'templates' such as:

e-categories (*, *n* entities, ↓*, *†, £↓*, £*†) Legal Relationship
(*, *n* entities, ↓*, *†) Actual Relationship

Finally, the application analyst will define a basic entity called 'USE' as an actual relationship involving two object identifiers. Thus he fixes the actual size of the relation by creating the 'template':

basic entity with (*, 1, 2, ↓*, *†) USE
generic labels

which is a list of what are called 'generic labels' which are, roughly, role indicators for the domains. The analyst also supplies a symbolic label for the relationship, in this case, *U*. This creates the symbolic labels in the *n*-part thus:

basic entity with (U, 1, 2, ↓U, U†) Use, U
specific labels

which takes the relation label as the symbol of the name domain without alteration and, suitably modified, to create symbolic labels for the time identifiers. Notice that the labels for the entity identifiers are not entered when the basic relationship is defined. This happens later when it is used in some expression. It might be used as $U(x, b)$ and generates specific labels:

$$(U, x, b, \downarrow U, U\dagger) \quad U(x, b)$$

but we could have written U(COMPANY, BUILDING).

Of course, the sequence of the identifiers is significant, the first being the user and the second the thing used. Semantic information of this kind will be available in the *n*-part of the relation of the second prototype where its significance will be supplied by tables of domain semantics analogous to those for entities in the first prototype. This deficiency in the first prototype is supplied by giving clear language names to the domains such as 'User', 'Used', the implications of which are clear enough to a person.

As well as the identifier labels, another item of semantic information which has only local significance is the pointer. Although in LEGOL when we use a relation label we are referring, by implication, to the whole relation we can select specific domains for attention. Thus in the expression

$$A <= \text{'CAPITAL ALLOWANCE'}$$

the arrow indicates that we are referring to a domain and, in this context, the '*A*' would be interpreted as the label for the name domain of the allowance rather than the entity, *A*, as a whole. The system would represent this by inserting a pointer '1' in the *n*-part tuple of the name domain.

8. The interpretation of LEGOL expressions

The above section has shown that much of the semantic information held by the system is embodied in a carefully structured system of names for the data. This section explains how the names given to the data items are used to control the interpretation of LEGOL expressions. The problem is to translate a LEGOL expression into a series of operations upon relational data structures. As these operations proceed, it becomes more difficult to associate meaningful names with the resulting relations and their domains. Therefore, there is always the important final step of endowing the resulting expression with clear meaning. Until this is done, the result cannot be stored within the highly structured semantic data base.

As an example of how the labels and pointers are used to control an operation, consider the expression:

$$Z'(x, b) \,\&\, U'(y, b) \rightarrow A(g, x, b, t) \ .$$

The operation '&' is called a 'time intersect'. It generates tuples for all those periods during which x owned the interest in the building, b, which carries an entitlement to the allowance and also during which b was used for some approved industrial purpose. The relations Z' and U' are subsets of all ownerships, Z, and uses, U; they are set up by the system as temporary relations during the interpretation of the relevant Section 25. Suppose we have the following data for dealing with a case:

Entitling interests Z'

(Z	x	b	↓Z	Z†	£↓Z	£Z†)
FREEHOLD	JONES	F	1	12	20	25
FREEHOLD	SMITH	F	12	18	25	26
FREEHOLD	JONES	F	18	24	26	30
FREEHOLD	JONES	N	19	30	15	30

Industrial usage U'

(U	y	b	↓U	U†)
MILLING	JONES	F	2	5

Industrial usage U'

(U	y	b	↓U	U†
MILLING	SMITH	F	7	8
BAKING	SMITH	F	8	24

Then the expression

$$Z'(x, b) \ \& \ U'(y, b)$$

will yield the data about all those periods during which some entitling interest overlaps a period of industrial use of the same building. It will not matter if the owner, x is the same as or different from the user, y. Thus we obtain, in this case, the result:

(Z	U	x	b	y	↓*	*†	£↓Z	£Z†)
FREEHOLD	MILLING	JONES	F	JONES	2	5	20	25
FREEHOLD	MILLING	JONES	F	SMITH	7	8	20	25
FREEHOLD	BAKING	JONES	F	SMITH	8	12	20	25
FREEHOLD	BAKING	SMITH	F	SMITH	12	18	25	26
FREEHOLD	BAKING	JONES	F	SMITH	18	24	26	30

Note that we have a problem of naming the time domains which may be derived from $\downarrow Z$, $Z\dagger$, $\downarrow U$ or $U\dagger$; we therefore reintroduce the asterisk and symbolise the start time as $\downarrow *$ and the end time as $*\dagger$. From these data we are only interested in knowing the circumstances which give rise to some entitlement of a capital allowance. This is symbolised by:

$$\rightarrow A(g, x, b, t)$$

which, by referring to x and b, says that we are looking for occasions when x and b are identifiers. All the other data are thrown away and the prescription results in:

Allowance, A

(A, g, x,	b, t, ↓A, A†, £↓A, £A†)
JONES	F
SMITH	F

which says that JONES and SMITH are each entitled to an allowance with respect to the building A. The times during which these entitlements are valid and the sums of money involved are blank, or unknown, as yet, nor do we know the precise tax account, g, and tax period t. The second rule:

$$A <= \text{'Capital Allowance'}$$

at first attaches a pointer to the name domain of A

(A, g, x,	b, t, ↓A, A†, £↓A, £†A†)
(1	)
JONES	F
SMITH	F

So that it can then transfer the literal to this selected domain:

(A,	g, x,	b, t, ↓A, A†, £↓A, £A†)
CAPITAL ALLOWANCE	JONES	F
CAPITAL ALLOWANCE	SMITH	F

The remaining blank domains would be filled in by the application of other legal rules.

These examples indicate how the symbolic labels given to domains, among other things, control the transfer of information from an evaluated expression into the entity which appears at the head of a prescriptive arrow. Earlier, two kinds of temporary entities were introduced. They were exemplified by a *period* defined using an entity category and the sub category of *industrial uses* defined in terms of the basic entity USE. The first kind of temporary entity enables one to select data representing something more abstract. The other temporary entity is used to define another at the same semantic level as the entity in the original expression. Thus, in the case of the rule:

$$\langle \text{expression} \rangle \rightarrow p(x, b, t)$$

although the expression on the left might contain indefinitely labelled domains, the semantics of p will indicate to the system that it should use generic labels where they result in no ambiguity. The other kind of temporary entity, which is named by adding a superscript to the name of a basic entity, (e.g. U' obtained from U) enables one to select only those facets of a complex expression which carry the same specific domain labels as those for the domains in the original, basic entity. These domains may be both identifiers and properties.

From the above account of some of the principles underlying the system, the problems of constructing a suitable interpreter may be appreciated. Implementation would have been difficult had it not been for the opportunity to use some new, powerful software systems.

9. The development of the system

The system has been implemented, first on a 360/44 and now a 370/145 at the IBM(UK) Scientific Centre, Peterlee. It has been possible to use two experimental systems developed at the Scientific Centre. One of these, PRTV, is a relational data base system with an extensible data manipulation language based on a relational algebra (Todd, 1976). The other system, MP/3, is a powerful macroprocessor which is used to build a flexible top-end to the relational data base system. (Mandil, 1973 and 1974).

The macroprocessor is first used to translate the LEGOL formalism into an intermediate language which takes account of the context of each expression. The intermediate language employs a repertoire of operators which have, as their operands, relations in two parts, data in one, semantics in the other. The macroprocessor is used to interpret these operators, effectively transforming the relational data base into a *semantic data base*. At the present only the d-part relation is being handled in the relational data base. The n-part relation, containing semantic information, is handled in the global variable storage of the macroprocessor. It is intended to overcome this restriction in a more developed version of the prototype.

More detailed accounts of the formalism and the system are given in two other reports available from IBM(UK)SC, Neville Road, Peterlee, Co. Durham and the London School of Economics and Political Science. It is hoped that the second prototype will be an approximation to a system with practical usefulness for drafting and testing of bodies of complex regulations.

10. Theoretical problems

By way of a summary, it may be useful to indicate again some of the theoretical problems encountered in this research. These problems are severe ones which account for the long term nature of the project. It is not expected to produce a general purpose system of practical value in less than several years; nevertheless, systems of limited scope may be produced more quickly. This is a realistic rather than a pessimistic projection of the progress of the work; the problems encountered, although difficult, do not appear to be insuperable. More immediate research results of a theoretical character are available. They relate to problems in information analysis, the systems work which leads up to the definition of the data processing system. If it is only possible to characterise this branch of computer science with more clarity and exactitude, then the LEGOL project will be worth pursuing.

The first task is to establish a methodology of research in information analysis. From the point of view of scientific method, the project may be viewed as an attempt to formulate a series of hypotheses about the description of a formal information system and to subject these hypotheses to destructive testing.

A key part of each of these hypotheses is the model within

which semantic information must be structured by the information analyst. There are many subproblems here, some of them very complicated. There are problems of understanding identifiers (for example, compare Stamper, 1977 with Grindley, 1974). Other problems include those of finding an adequate set of entity types, discovering more about the use of names and building a suitable generic specific structure of entity categories. The semantics of individual domains presents as many problems as semantics of entities. These problems are equivalent to those data dictionaries and conceptual schema for data bases (Mason, 1975).

Finding a way of expressing legal prescriptions amounts to a search for an economical repertoire of very high level operators. These operators correspond to fairly simple seeming forms of expression in natural language but in data processing terms they are very complicated. The research suggests that such operators may usefully be defined if they can exploit an extensive semantic structure. The semantic structuring limits severely the number of things one might meaningfully do with data. This should enable us to overcome one of the worst deficiencies of programming languages. These permit one to perform a host of operations on logical variables, numbers and character strings, that make mathematical sense but are not necessarily sensible in terms of entities with which they are associated in the real world.

There are problems of how to represent semantic structures within a computer system so that they may be augmented, modified and used by the information analyst. More general subproblems are those of checking the logical consistency of a problem definition and of simulating the functioning of a system constructed according to that logical definition (compare Grindley, 1974; Teichrow and Sibley, 1972; Langefors, 1970). A problem for the designer of data base software is to construct systems which will represent and handle semantic information. An important theoretical question here is how to distinguish between semantic and purely syntactic features in the system.

Finally, there is the problem of how the designer of a computer system should exploit the semantic information assembled by the information analyst. He must obviously ensure that data are manipulated within the semantic constraints or the results will be meaningless but in the data processing system the semantic constraints are not made explicit, they are implicit in the actual programming structures which are chosen on grounds of efficiency. This brings us to the theoretical root of a major fault in many data processing systems: their inflexibility. The system when first created probably processes data in a meaningful way but, because the underlying semantics are not explicit, changes or extensions are likely to introduce inconsistencies or result in outputs that are misleading.

Acknowledgements

For their willingness to discuss critically aspects of this research, I should like to thank my colleagues at the LSE—Susan Jones and Peter Mason, and those at the IBM Scientific Centre, Peterlee, in particular Salah Mandil (now at the WHO, Geneva), Terry Rogers and Stephen Todd.

References

ANSI-SPARC (1975). Interim Report of DBMS Study Group, February 1975, American National Standards Institute, Washington DC.

CODD, E. F. (1970). A relational model of data for large, shared data banks, *CACM*, Vol. 13, pp. 377-387.

GRINDLEY, C. B. (1974). *Systematics*, McGraw Hill, London, New York.

LAND, F. F. (1974). Criteria for the Evaluation and Design of Effective Systems, *Proceedings, International Symposium on Economics of Informatics, Mainz 1974*, IBI-ICC, Rome, pp. 116-127.

LANGEFORS, B. (1970). Papers in *Systemering* 70, (Bubenko *et al.*—eds.), Studentliteratur, Lund.

MANDIL, S. (1973). A Macro-Processor as a Generalised Top-End to Computing Applications, *SOFTWARE'73 University of Loughborough*, Leicestershire.

MANDIL, S. (1974). The MP/3 Macro-Processor, *Proceedings of European Computing Congress*, Brunel University, 1974.

MASON, P. J. (1975). The Legol Semantic Model, Discussion paper for BCS Data Dictionaries Working Group.

STAMPER, R. K. (1973). *Information*, Batsford, London: John Wiley, New York.

STAMPER, R. K. (1973). The Legol Project and Language, *Proceedings Datafair Conference, Nottingham 1973*, British Computer Society, pp. 269-276.

A more detailed account of the LEGOL Project is available from IBM (UK) Scientific Centre, Neville Road, Peterlee, Co. Durham:

STAMPER, R. K. (1976). *The LEGOL Project: A Survey* UKSC 0081.

STAMPER, R. K. (1977) Identifiers of Physical Objects, Proc. IFIP TC2 Workshop on Data bases, Nice, Jan. 1977.

TEICHROW and SIBLEY, (1972). PSL, a Problem Statement Language for Information Systems Deisgn, ISDOS Working Paper, University of Michigan, Ann Arbor.

TODD, S. J. P. (1976). The Peterlee Relational Test Vehicle—a system overview, *IBM Systems Journal*, Vol. 15, No. 4, pp. 285-308.