

TOWARD PERFECT INFORMATION MICROSTRUCTURES

October 5, 2003 DRAFT

*Ed Lowry
7 Alder Way
Bedford Mass 01730
781 276-4098
eslowry@alum.mit.edu
users.rcn.com/eslowry*

Eliminating needless complexity from formal languages and their applications leads to data objects which are: purely connective, asymmetric, and all the same. The supporting rationale is simple, and similar to the reasons for making wheels round, pillars vertical etc. It is proposed that further simplification leads to a optimum data object structure, an hierarchically interconnectable pointer. A series of hypotheses supporting the proposed optimum is presented and defended. Understanding data object structure can be used to improve the quality of information, the scientific foundations, the generality of language, and the ease of working carefully with many kinds of information with or without computers. Overspecialization of data objects appears to be a major source of needless complexity for software and technical education.

In both computing and technical education the same basic need is being addressed -- to increase the productivity of mental effort in working carefully with various kinds of information. The general engineering goal of getting more for less takes the form of expressing more information with less complexity. That focuses on the problem of eliminating excess complexity. A basic engineering method is to press the limits of getting more for less and try to understand the consequences. Here the focus is on the consequences for the structure of the most basic building blocks of information, data objects, as elimination of excess complexity approaches completion. We examine the structure of "atoms" of information for which further decomposition has no simplification value.

Pointers which have some additional connections for hierarchical organization appear to offer an enduring theoretical and practical optimum primitive data object structure. The optimum applies broadly across rich applications and technical descriptions written in non-trivial formal languages intended for execution by machine or for understanding by people. The analysis raises doubts that there can be long term justification for using any other kind of data object for substantial applications. A simpler analysis argues that long term use of at most one kind of data object can be justified.

What is a reasonable structure for data objects? How is the quality of information affected by the fine structure of its representation? While data objects are among the most important basic structures in any technology, they are so far among the most poorly understood. From arrowheads to turbine blades, the developers of technology have had good reason to give meticulous attention to the structures of the basic components of their artifacts. Reasons for trying to understand the fine structure of information and its representations include:

- ! Knowledge workers succeed by working *carefully* with various kinds of information. Learning to do so pervades education. Careful work of almost any kind requires control of fine structures.
- ! Foundational knowledge in most fields of science and technology consists mainly of knowledge about fine structure of the subject matter.
- ! The quality of most artifacts is strongly affected by their fine structure. Poorly chosen data object structures can degrade the quality of massive quantities of technical information including its accessibility, usability, clarity, cohesiveness, ductility, and durability of usefulness.
- ! Teachers mainly teach people how to arrange pieces of information, They need to understand the structure of the pieces to fix a basic problem. At present, technical knowledge is represented in ways that prevent much precise information from being presented precisely.

If the design of a vehicle is optimized to minimize vibration, there will be many consequences, including a requirement that the wheels be round. There are at least a few dozen similar cases where progressively optimizing an engineering design terminates with a sharply defined structural constraint rather than by design choice in a tradeoff. In each case the structural constraint eliminates a class of design deficiencies. Other examples of such “**irreducibility optima**” are that constraining pillars to be vertical eliminates shear forces, constraining mirrors to be flat eliminates image distortion, constraining the top and bottom of building blocks to be horizontal eliminates sliding forces. Such constraints often have no adverse side-effects which raise significant tradeoff issues.

A series of constraints on data object design is proposed. The first three of those constraints eliminate design deficiencies of information systems in a way that is similar to the irreducibility optima. The rationale is mainly plausibility arguments which are fully supported by the available empirical evidence. Design choices for data objects are inherently discrete. They have no continuously varying properties which could be balanced in a tradeoff. Like masonry building blocks, the optimum structure for building blocks of information is affected much more by the interaction between the building blocks than by the larger architecture they are part of. The specialization of data objects appears to have developed

accidentally over time as a result of making poor initial choices and then adding more data object structures to compensate for the limitations.

EMPIRICAL OBSERVATIONS

About 30 calendar years and 70 person-years of experience with a series of languages [1,2] (now called Shannon) developed mostly at IBM and DEC have yielded the following empirical observations:

1. Moving to very simple structure for data primitives simultaneously improved:
 - simplicity of expression for large applications,
 - simplicity of the definition of language semantics for given capability, and
 - generality of language semantics.
2. A language with 2 data object structures could be redefined as having a single data object structure (by combining the structural features) and the result had the same simplicity of expression and slightly improved simplicity of language.
3. When all functions from different problem domains operate on data made from the same kind of data objects, combining them in one language is straightforward. A longstanding dichotomy between "structurally expressive" and "functionally expressive" languages (discussed later) was eliminated.
4. Changes to the data object structure had only small effects on simplicity after a high level of simplicity had been achieved. This suggested that a true optimum was being approached. In addition, the variety of data object structures which seemed to have any realistic prospect of improving simplicity was shrinking to just a few possibilities.
5. The leading edge of simplicity of expression for large applications appears to have changed only very slowly over the past 25 years, suggesting that an optimum functional semantics is being approached.
6. In addition to enhancing specific language design, the improvements give language design in general a more clearly defined engineering envelope and allow orderly expansion of the envelope. Simplification clarifies - - including how to simplify further. The rich space of possible language designs has made it difficult to use engineering design disciplines directly. However, focusing on the structure of data objects leads to opportunities for dependable engineering analyses, and comparisons which are not difficult to pursue. The increased language generality helps to improve the uniformity of comparisons and as a result the leading edge for simplicity of expression is more sharply defined.
7. Such language could significantly combine the precision of formal language with the readability of natural language.

8. Substantial searching showed no language giving greater simplicity for large applications, no effective challenge to hypotheses about optimum data object structure, and no evidence of any other leading edge investigation of how data object structure affects information quality. An offer of \$20K posted on the author's website starting in January 1999 produced responses consistent with the view that the technical community has neglected the issue of data object structure. No one has given reasons supporting long term use of any alternative to the proposed optimum in substantial applications.

HIERARCHICALLY INTERCONNECTED POINTER DATA OBJECTS: NEEDLES

A series of 8 hypotheses (H1 to H8) are stated below. To provide clearly defined hypotheses, we describe more specifically what complexity is being minimized. The rationale supporting the hypotheses suggests that their soundness is unaffected by the way complexity is measured as long as adding text (other than extending name lengths) also adds to complexity.

Rigorous minimization of one kind of complexity can result in eruptions of complexity of a different kind. Pressing the limit of eliminating complexity from expression of applications by itself would excessively shift complexity into the language. More clarification results from considering minimization of complexity of the combination of language and expression of applications.

It is proposed here that maximizing simplicity favors language semantics based on hierarchically interconnected pointers. They are referred to here as *needles* by analogy with pine needles which are pointed at one end and connected to trees at the other. The 8 hypotheses of this paper have a common premise. Two of them summarize the others. They are:

For every sufficiently large and diverse set of procedurally defined applications, whenever the total complexity of deterministic language definition plus the expression of the applications in the language(s) is minimum, then:

Hypothesis 4:

All of the data objects will be purely connective, asymmetric, and have a common unspecialized structure.

Hypothesis H8:

The data objects will form hierarchies where each object is a "pointer" (or needle) which points away from a parent in the hierarchy possibly toward a remote object and which has secondary connections to at most two siblings and two children in the hierarchy.

Summarizing H1, H2, H3, hypothesis H4 is supported with simple arguments similar to those which lead to irreducibility optima. Its implications are large independently of the more subtle arguments leading to H8. The hypotheses are expected to hold even when diversity of the applications is fairly

limited. Diversity requirements are implied by the supporting rationale. Great breadth of subject matter does not seem necessary. The hypotheses do depend on occurrence of some dynamic creation of objects in sequences, tightly nested iteration and use of various kinds of references such as to sets of sets. The set of applications may be biased in favor particular kinds of application in numerous ways as long as the needed patterns are included.

Other articles [3,4,5] describe earlier explorations of data object structure. The only properties of a needle are its identity, what it points from (its parent), what it points to, and its connections to siblings and children in the hierarchy. The needles only have connections to needles (not any machine storage objects). The choice of which neighbors in the hierarchy a needle is connected to is somewhat arbitrary, because some can be derived from information about others. Access to a predecessor sibling or a final child in the hierarchy can be derived from other available connections in a data structure.

An expression in a language based on needles could be presented in a textual form such as:

82 = count every element where some isotope of it is stable;

There are more short examples in the Appendix and longer ones in [6]. Statements can be represented in conventional text (which can be very readable) and the text in turn can be represented using needles. While the subject matter of the expression can be interpreted by a human to refer to ideas in physics as in this example, from the viewpoint of the language definition, the subject matter of the statements is always interpreted purely in terms of structures made from needles.

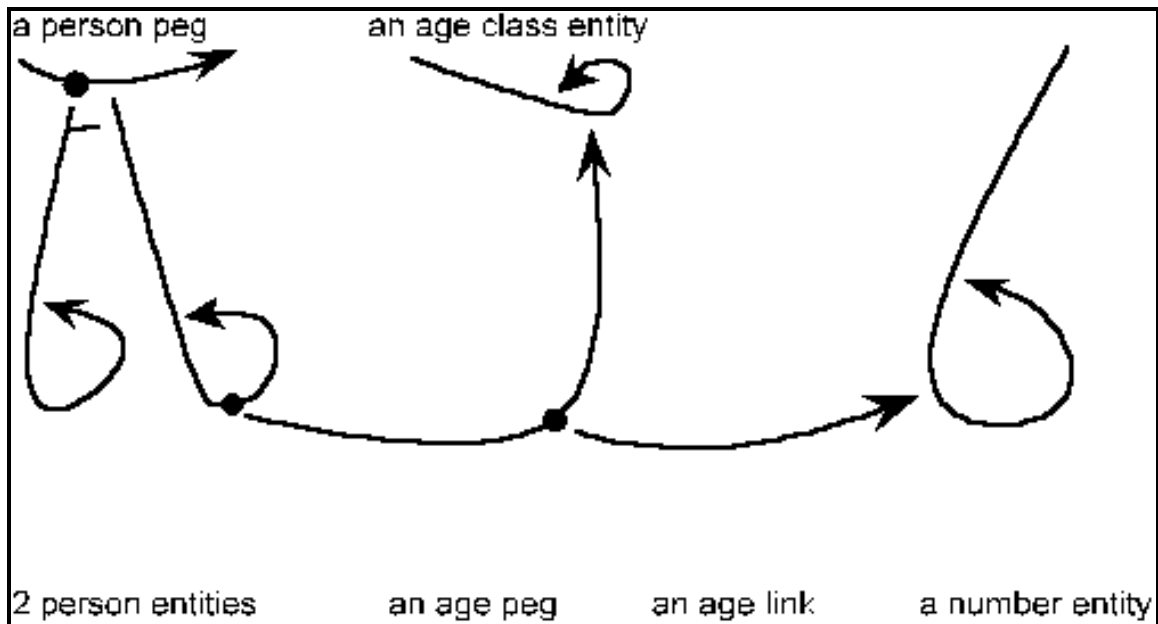


Figure.1 Representation of the age of a person

Figure 1 illustrates a possible way that needles could be used to represent the age of a person. It is practical to use needles in 3 roles:

- Entity needles, such as those representing the two persons, the number, and the age class in the diagram.
- Link needles, representing relationships usually between entities, such as the age relationship shown.
- Peg needles, providing class information for selecting from among siblings in an hierarchy. The age peg shown points to an age class entity and determines the class of the link. Multiple objects with similar patterns of descendants can hang from a peg.

The two entity needles representing persons also hang from a peg which points to a person class entity. Entity needles do not need to point to anything, and are shown here as pointing to themselves. Besides the non-hierarchical connections, shown as arrow heads, there are three kinds of connection to neighbors in the hierarchy: parent connections (rear end of the arrow), a next sibling connection (a side branch), and first child connections (dots).

The needles may be interpreted to play a variety of roles but that does not add to the structural complexity of individual needles. The labeling in the diagram is not part of the needles. That information is derivable from the other connections among the needles. The use of needles to represent individual numbers implies using huge numbers of needles, but applications can be translated into efficient computer oriented forms for execution where only a small subset of the numbers are explicitly represented.

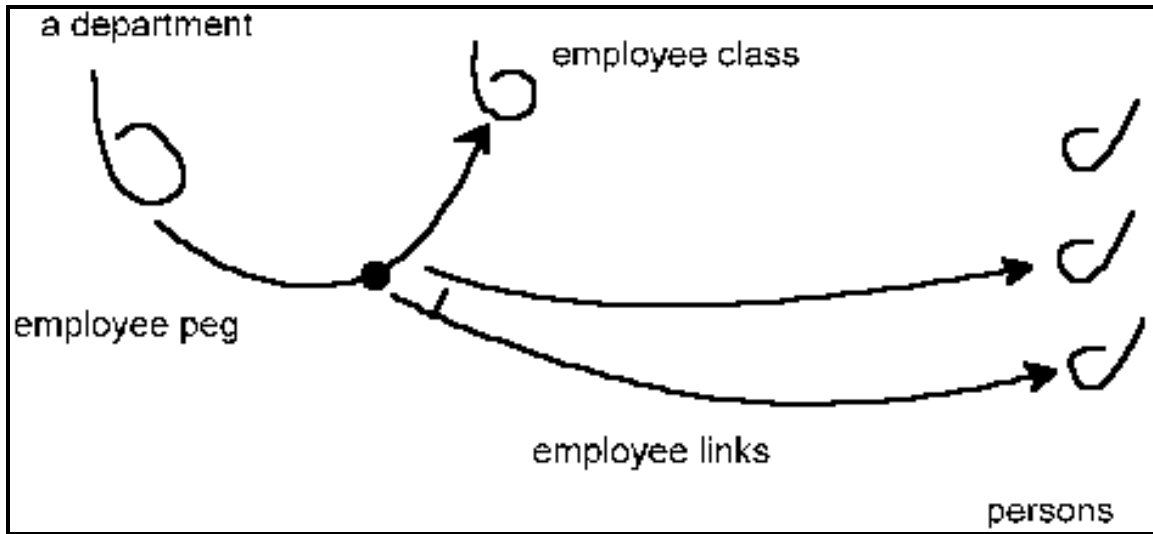


Figure 2. A multi-valued relationship

The full advantages of using needles rather than other objects are best illustrated using more complex examples. Consider instead a multi-valued "employee" relationship between departments and persons (Figure 2). In that case, an employee peg would have multiple links hanging from it, each pointing to a person. An employee peg could later be used as a single pre-existing object representing the set of employees of a particular department. Its availability could be used to simplify some expressions which work with multiple sets of employees.

Many data models use pointer-like objects along with others [7,8,9]. The minimization of complexity depends on using needles exclusively.

CONSTRAINTS ON DATA OBJECT STRUCTURE

Given a rich body of deterministic applications, we consider the problem of choosing sequentially executable language(s) in which to express the applications and of choosing expression of the applications in a way that minimizes the total complexity of the expression plus the deterministic definition of the language(s). This minimization problem provides a context which sharpens the definition of the engineering issues. Different kinds of excess complexity arising from different kinds of sub-optimum design seem sufficiently independent that relatively simple considerations are adequate to show how the minimization imposes constraints on the structure of the data objects used by the language(s).

The analysis presented below focuses on data object structure and is independent of many aspects of language design including the following:

- Syntax including:
 - operator overload
 - macro expansions
 - pattern matching abbreviations.
- Concurrency disciplines for multiprocessing or database.
- Stylistic disciplines which limit the use of informal constructs in large systems.
- Potential extensions for defining reversible functions, as in Prolog.
- Optimization conventions which could compromise diagnostics or numerical accuracy.
- Functional programming which restricts use of functions with side effects.

According to the hypotheses, no design ingenuity in these areas using deterministic language will affect the outcome favoring needle data objects.

Varying the structure of data objects to maximize simplicity can be pursued in a way that appears to lead to an objective optimum rather than a tradeoff. This contrasts with efforts to maximize syntactic brevity that lead to obscure conventions whose value is very subjective. Data typing allows simplifying abbreviations which could affect the analysis, but none of the common abbreviations of that kind appear to have any effect. The analysis ignores consideration of execution time efficiency which could favor machine-oriented language with very different data primitives when pressed hard enough.

By convention, the subject matter of the languages are data states which are composed of data objects which are indivisible. A data object can connect only to data objects taken as a whole, not to any internal structure within them. The most general data object considered is one with connections to other data objects and some internal state. Where simplicity is a serious engineering goal, the possibility of usefully starting from something more general seems hard to imagine, at least so far. Adequately imagining the possible alternatives is critical to the success of the analysis. Each (non-summarizing) step in the sequence of hypotheses explores a set of alternatives and narrows them before the next step.

Efforts to refute the following hypotheses (if they do not succeed) can help validate them. The sequence of hypotheses imposes increasingly severe constraints on the structure of data objects. The constraints progressively decrease the prospect of useful specialization of data objects. As a result, agreement with the early hypotheses increases the value of resolving remaining uncertainties.

Each of the hypotheses has the same premise:

For every sufficiently large and diverse set of procedurally defined applications, whenever the total complexity of deterministic language definition plus the expression of the applications in the language(s) is minimum, then:

H1. The only structural features of a data object are connections to data objects.

When a data object representing a conceptual object is newly created, it must be directly connected to objects already in the data state because failure to do so would make it inaccessible later and its creation would be ineffective. Some accessible objects can be provided which exist prior to execution. If the internal state of a data object includes an encoded reference to an object anywhere in a data state, the representation can be simplified by replacing the encoded reference by a direct connection to the object. For that simplification to work, it is necessary to include in the data state distinct data objects representing all of the distinct conceptual objects to which references are needed. Doing so may require incorporating large sets of numbers and character strings in the initial data state, but that can be done with little complication.

Replacing references in the above way will have the effect of eliminating all state which is internal to data objects so that their only structural features are a set of connections to data objects. A conceptual object may have very complex internal structure which cannot be represented fully by any practical primitive data object. Such internal structure can be represented by an hierarchy of primitive data objects.

H2. The data objects are asymmetrical in their connections.

For the given applications, completely deterministic execution is needed. Determinism allows for the elimination of ambiguity. Any symmetry (such as in undirected arcs) in the connections of a data object would cause a non-deterministic result when a function was executed to fetch a connected object. Any inclusion of symmetrical connections would be an irrelevant complication to the language.

Of course many applications of interest naturally include non-deterministic elements [10]. The requirements for deterministic execution are expected to dominate even in those applications, but that issue should be explored separately.

H3. The language definition will treat all data objects as having the same set of connection types, though not all types of connection occur for all objects.

If there were multiple kinds of data object with different kinds of connection in any one language, we could define a composite kind of object which has the union of those kinds of connection. Actual objects would have subsets of those connections. If there were multiple languages, then a single data object with a composite of the kinds of connection for all of them could be defined and the same functions with the same definitions would continue to apply but ignoring other connections. Any merging of functions, languages, and data objects that resulted would provide a net simplification of language with no change in the complexity of expression. A diverse enough body of applications would assure that all types of connection between data objects which can contribute to overall simplicity will be included. If there is significant overlap among the kinds of connection between data objects that are referred to in the more basic functions, then those functions will form a common kernel for all of the language(s).

H4. All of the data objects will be purely connective, asymmetric, and have a common unspecialized structure.

This summarizes H1, H2, and H3. Each of those imposes a constraint in the same way that the irreducibility optima (see below) do. They support the existence of a unique optimum which is investigated in the subsequent hypotheses.

In these 3 steps, exhaustive exploration of alternatives is easy. Objects either do or do not have internal state, symmetric connections, or variations in structure.

For H4 to have wide practical implications depends on the minimization of complexity applying to a broad class of applications and not just to extremely rich sets of applications. The rationale supporting H1 to H3 suggests that the applications need not be very diverse. The rationale supporting those hypotheses is close to common sense, and it seems likely that any deficiencies in the rationale would also be common sense. Another requirement for practical implications is that the variety of connection types not be large. Empirically just a few kinds of connection are adequate, so the effort to get them right can have a large payoff.

H5. The connections allow all the objects which comprise a data state to be located in an hierarchy with ordered siblings. A data object will have connections to immediate neighbors in the hierarchy when they exist: its parent, its successor sibling (and perhaps predecessor), its first child (and perhaps its last).

In addition to H3, reasons favoring uniformity of data primitives come from consideration of an

historical dichotomy between two kinds of formal language. Accessible information always exists in some physical representation and when designing any representation system (even musical scores or ice sculptures) we need both:

ability to accurately represent many structures
and
ease of working with the representations of all the structures using a broad class of operations.

For symbolic systems these requirements may seem to conflict. In computers the difficulty is illustrated by an historical dichotomy between:

structurally expressive languages (C, Ada, PL/1, Codasyl data base, etc which use *many* primitive kinds of data object)
and
functionally expressive languages (APL, Relational data base, etc, which use very *few* kinds of primitive data object)

The functionally expressive languages provide for multiple nested subexpressions operating on large aggregates of data in a single encompassing expression. However, the early languages of that kind were restricted to relatively inflexible data structures which do not match a variety of conceptual structures as well as the structurally expressive languages do. Adding functional expressiveness to existing structurally expressive languages would lead to large complications of already complex languages. Variations on the definition of many functions would be needed to provide for different primitive data objects. Efforts to later enrich the data structures of functionally expressive languages have also been impeded by rising language complexity.

Empirically, the functionally expressive languages have very few data object structures. This suggests that the functions which operate on data aggregates have semantics which are sensitive to data object structure. Examination of those functions (from the list under Language Generality below) shows very little structural sensitivity except that the functions can be simplified if they can assume a default procedure for iterating through the data aggregates. Providing functional expressiveness then appears to depend on a high level of regularity in the way data aggregates are organized for iterative access. When an aggregate results from the execution of any of numerous subexpressions, it must be presentable as an argument to any of numerous encompassing expressions in a way that permits the same method of iterative access to its substructure. This constraint requires that there be just one dominant structure in the language for iterative access within data aggregates. That constraint along with the need to keep the language simple favors all data objects having the same kinds of connection for iteration. This is a strong constraint on the structure of data objects whose validity is largely independent of other assumptions related to these hypotheses.

Experience with the KEEP language [2] has shown that the dichotomy between structural and functional expressiveness can be resolved using simple data primitives with common structure for iteration and the advantages of both kinds of language can be combined in a relatively simple language. The experience has shown that the restriction to a single common data structure for iteration need not interfere with language generality.

If an additional constraint that the data objects form an hierarchy can be justified, then that along with the requirement for uniform iteration structures and the overall requirement for simplicity severely constrains the data object structures.

Any realistic computer applications or technical descriptions which are at least moderately rich will include hierarchical structures. The applications are normally designed with ease of human understanding in mind. That favors hierarchical structures which reduce human search efforts. Nesting of subsystem structures, coalescing factored information, and simple name resolution conventions also favor hierarchy. Usually conceptual objects with complex internal structure are most naturally represented by hierarchies of primitive data objects, but imposing such hierarchies on real world structures can be artificial. "Links" which exist within an hierarchy can be used to represent relationships to objects remote from the region in the hierarchy that they are a part of.

Within an hierarchy there is a need for applications to access the neighbors of an object in the hierarchy. That will include providing access to all the "child" objects immediately beneath it in the hierarchy. Direct connections to all the children are not possible because sets of children are open ended in size. A straightforward way to provide access to the children is to include a connection to a first child object whenever it exists and a connection to a successor object whenever it exists. Those connections also provide the desired uniform structure for iterating through sets. Some of the sets will be ordered in the application and the explicit successor connection provides for the ordering. Whether required by the application or not, deterministic execution requires a definite ordering.

There appear to be very few alternatives ways of providing access to children which do not introduce obviously avoidable complexity. One possibility would be to avoid direct connections to successors and use separate objects to access successors. Empirically, that approach causes excessive complexity (and also obscurity). One kind of complexity is in providing assurance against loss of integrity in the hierarchy. The direct connection to the first child may be unnecessary if non-deterministic language definition were allowed to specify a function producing the first child. However, it does seem necessary in a strictly deterministic definition.

At least some applications require access to fairly arbitrary neighbors in the hierarchy from fairly arbitrary conceptual objects. Access to the parent is sometimes needed and it can be provided by a language function without requiring users to make prior provision for the access. One approach would be to provide a function which does a large search for the parent. Another would be to implicitly set a direct connection to the parent whenever an object is created. The latter is simpler by a small margin.

Whether predecessor and last child connections are provided by primitive connections or by derivation seems to affect only a few sentences in the language definition.

H6. A given data object will have structural features which include at most the following eight kinds connection to data objects: five kinds of connection to neighbors in the hierarchy (parent, successor, predecessor, first and last child), connections to "class" objects which are used to select from among a set of children, connections to "relatee" objects when the given object represents a relationship between some other object and the relatee and (conceivably) connections which represent other single valued relationships.

Support for this hypothesis is mainly empirical so far. The KEEP language is consistent with this constraint and had leading edge simplicity of expression and language generality for many years within a relatively small language. A more thorough search for other possibly helpful kinds of connection is needed.

The existence of preexisting "computer storage objects" or connections to them or connections between them can be excluded on grounds that they add to complexity which can be avoided. They would cause gratuitous complexity when inserting objects into sequences or deleting them as a result of the need to shift other objects relative to the storage objects or provide chaining conventions.

H7. A data object will have connections to immediate neighbors in the hierarchy and at most one other object.

This hypothesis is based on the above rationale, some further plausibility considerations and consideration of empirical designs for KEEP and Shannon. The simplifications provided by this constraint are relatively small and non-obvious. They depend on rich applications where there is substantial invocation of functions which perform operations on sets of sets. The preliminary design for Shannon provides data objects which have connections to immediate neighbors in the hierarchy and at most one other possibly remote object. For some data objects (called pegs) the possibly remote connection is to a "class" (or type). For other data objects (called links) the possibly remote connection is used when the link represents a relationship that associates its grandparent with some possibly remote "relatee" object.

Assuming H6, the question arises as to whether equal or better simplicity could result from using objects which have more than one connection to non-neighbors in the hierarchy. The following four possibilities are considered in imposing the restriction.

a. Multiple class connections

The class connection is used to select one peg from a set of sibling pegs. Its use is analogous to selecting a field in a record. There appears to be no pressure for more than one class connection on a peg. Current languages seem to provide no capabilities implying a need for anything more complex. The class connection is similar to the connections to hierarchy neighbors in that it too is used (though indirectly) to select a neighbor in the hierarchy, one of a set of children. A need for more complex selection can be handled using additional levels in the hierarchy.

b. Multiple relatee connections

Any need for multiple relatee connections is most simply handled by creating more links rather than multiple connections on one link. Links can represent relationships which may be single-valued or multi-valued. For example, a multi-valued relationship could identify a subset of persons as the employees of a department. See Figure 2 above. A peg connecting to the employee class would be a child of the department object. The links to employees would be children of the peg. An execution which adds a person to the set of employees of a department would be done most simply by creating under the peg a link which connects to the person object. When selecting employees of the department, a connection to the employee class would be needed on the peg but not the links.

Any multi-valued relationship among conceptual objects would similarly be represented by a peg with a set of links for children. For language simplicity, single-valued relationships would be represented the same way, but using just one link as child of a peg.

c. Both class and relatee connections

A peg which is the parent of a set of links may be used to represent the set of objects pointed to by the links. In rich applications the availability of the object representing the set can be used to make simplifications. For example, a function which operates on sets of sets could accept such objects as arguments in a way that is independent of what kind of conceptual object the sets are part of. That would increase the reusability of the function in a way that contributes to overall simplicity. The creation of the peg can be done implicitly so there is no complexity penalty in expressing the applications that way. The overall simplification may not be large, but understanding how to achieve it helps to clarify data model design. If pegs and links were to be combined into one complex object with both a class connection and a relatee connection the simplification would be unavailable.

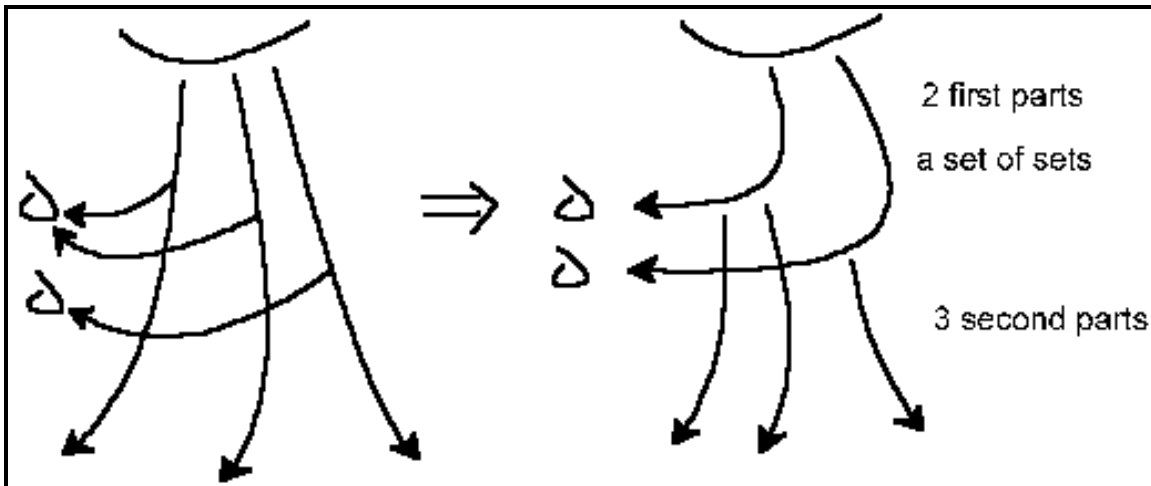


Fig 3. Decomposing complex objects provides sets of sets

The left side of Figure 3 shows three objects each of which has both a class connection and a relatee connection. When they are decomposed as shown on the right side, two of the "first parts" can be collapsed into a single first part (or peg). A set of such first parts with a common class connection can then represent a set of sets.

More generally, if a set of complex objects with a common parent and no necessary ordering have two kinds of connection to remote objects and one of the connections can be used to classify the set into subsets, then there is advantage to decomposing each object into two parts. The first part would include connections for the parent and the classification. The second part would treat the first part as its parent and include the other remote connection. Any pair of first parts which are classified the same way could be collapsed into a single object with multiple second parts (possibly reordered) as its children. A set of such first parts with a common classification could then serve as representatives of a new set of sets which could be used to simplify applications which are sufficiently rich. A function could then accept such a set of sets as a single argument. Without the decomposition, the set of sets would be unavailable directly as an argument necessitating construction of the set of sets, or specializing the function in ways that reduce its reusability. The decomposition reduces the number of remote connections on an object toward one, but it has no effect on the variety of connections to neighbors in the hierarchy. Reducing connections to neighbors cannot help because it cannot succeed. After all possible simplifying transformations have been made, data objects in hierarchies with ordering will still have the same five kinds of association with hierarchical neighbors.

Summarizing, we can use connections to classes, relatees, and neighbors in the hierarchy and experience no need for more than one of each kind of connection on any data object. Further, there is only a need for one connection to a remote object: either a class object or a relatee object.

A class could be selected either by the peg pointing to a class object or by the ordinal position of the

peg in a sequence of pegs without affecting the correctness of the hypothesis.

d. Connections representing other single valued relationships

It is conceivable that a data object could include a set of connections, designated C1, C2, C3, etc, which are used for representing arbitrary single valued relationships. In Figure 1, the age relationship could then be represented alternatively by a C7 connection of the person entity connecting directly to the number entity. There would be no need for the age peg, the age link, or the age class entity. Setting an age relationship could then be expressed by using an expression such as:

SetC7 (George, 27) .

By some measures of complexity, that expression could be regarded as simpler than a more conventional (except for syntax) expression, such as:

Assign (age, George, 27)

based on the representation of Figure 1. A measurement convention favoring the latter expression in writing applications would dominate language complexity issues and eliminate the C1, C2, etc connections. If the measurement convention favored the former expression, then similar expressions would be favored for multi-valued relationships, and could be provided as an abbreviation. The same abbreviation method could also serve for single-valued relationships allowing the semantics of multi-valued and single-valued relationships to be the same. By either measurement convention, the C1, C2, etc connections would lead to excess complexity in the aggregate complexity of application expression and language.

H8. The data objects will form hierarchies where each object is a "pointer" (or needle) which points away from a parent in the hierarchy possibly toward a remote object and which has secondary connections to at most two siblings and two children in the hierarchy.

This summarizes the preceding hypotheses. Support for the hypotheses now takes the form of the above plausibility arguments, experience which is consistent with them, and an apparent absence of contrary evidence despite searches and financial offers.

Additional evidence can be pursued by more extensive searches and solicitation of critical evaluation from the technical community. Additional empirical support would also help.

Effort has been made exploring the possibility of formal proof of some preliminary versions of the hypotheses. While the effort was helpful in reformulating the hypotheses, no proofs resulted. A proof would require a search through a well structured solution space, but here the main uncertainties are about the appropriateness of the solution space and what considerations might be neglected. H7 seems more amenable to formalization than the others.

One area where experience from a large community could help is to check whether there is any simplicity improvement could result from an eighth type of connection between data objects beyond

those identified in H6. There may be no practical way to answer that question by closed analysis. If it takes a long time to find such a connection type, its value is likely to be limited to a rarely used cluster of functions which could be treated as a language extension. Another area where criticism of the hypotheses could focus is to find alternative ways to achieve the advantages of hierarchical interconnection and uniformity of iteration in H5.

EVALUATION OF NEEDLES

The hypotheses have been framed with the intent that they could be either confirmed or denied. However well they may be confirmed, other considerations are needed to fully evaluate the role of needles.

There are other engineering values besides simplicity of expression and simplicity of language. A language designed with maximum simplicity would not provide for adequate diagnostics. Experience with KEEP suggests that good diagnostics can be provided with a language based on needles.

As stated the hypotheses apply only to very rich applications. The supporting rationale offered above suggest that the applications need not be very rich to justify using needles. If there is significant insertion of objects into sequences and tightly nested iteration, then something close to needles minimizes complexity. The references to sets of sets which justify heeding H7 are rare, but there also appear to be language simplicity benefits in doing so.

The restriction to use the "uniform relationship measurement convention" could be lifted and the results explored. Though the symbol setC2 (or some equivalent) may be regarded as one symbol, most people will regard it psychologically as two.. In any case the clarity benefit of referring symbolically to the class of relationship (e.g. age) is likely to outweigh any difference between the two measures of simplicity, at least for a strong majority.

The empirical evidence that is available favors something close to the described constraints. The KEEP language [2] effectively used typed needle-like objects exclusively. Among well developed languages, it provides the greatest simplicity of expression known to the author both for substantial applications and for data base queries. A wide range of widely useful function is included in a relatively small language (about 100 pages of definition). A distributed data base application of several thousand lines of KEEP code was developed which routinely processed a few thousand transactions per day from about 20 locations. The deficiencies of typed needles were only noticed from theoretical considerations (those of H7), despite considerable practical experience. Implementability with reasonable effort and good performance are additional engineering values to be considered. KEEP never had an optimizing compiler but no reasons to expect serious problems in that area were seen. The relationship with non-deterministic language needs clarification. Four different variations on needle-like data object structure have been explored and little variation of the ways applications were expressed resulted.

There are good reasons to clarify further:

- Data object structure impacts the quality of the information represented.
- Information technology seems unique in providing poor information about its most basic structures.
- Sound engineering practice requires justification of specialization.
- Common sense considerations seem to imply that there is a unique optimum structure for data objects in rich deterministic applications.
- This analysis supports the view that information systems and technical communication widely use information structures which are unreasonable in much the same way that square wheels are unreasonable..
- A series of simplifications can become unblocked, as discussed below.
- Anyone developing language needs a sound basis for design choices.

IRREDUCIBILITY OPTIMA

The optimization of data object structure can be clarified when put in the context of the following distinctive set of engineering optimization solutions.

The term "irreducibility optimum" is used here to describe an engineering solution where a structural constraint on a single part eliminates a class of design deficiencies. In many cases the constraint does not create significant adverse side-effects. For example, the optimum design for wheels includes a structural constraint, its round shape, which eliminates a cause of vertical vibration and remains optimum across wide changes in requirements for loading or velocity, etc. Rather than terminating by design choice when competing values are judged to be balanced, the process of approaching an optimum solution stops necessarily when an uncrossable mathematical boundary is reached. There is no way to be rounder than round, more vertical than vertical, flatter than flat, etc. Negative amounts of some physical quantities (vibration, shear forces, distortion, etc) are not possible. About 25 such irreducibility optima have been identified including:

- Round is the right shape for wheels.
- Tubular is right for pressurized pipes.
- Horizontal is right for axles, floors, and tables.
- Flat parallel top and bottom are right for building blocks.
- Vertical is right for loaded pillars, walls, door hinge bearings.
- Monoplane is right for aircraft including kites (after the structure is sound).
- Flat is right for personal mirrors.
- Straight is right for road segments (often compromised).
- Spherical shell is right for pressure vessels (often compromised).
- Helix is right for bolt threads.
- Uniform thickness is right for cables.
- Equal length arms are right for simple balances.

More recent cases:

- Binary is the right radix for digital memory elements.
- Helix is right for longitudinal springs.
- Spiral is right for wind-up springs.
- Cylindrical is the right cavity for propelling pistons and bullets.
- Spherical is right for ball bearings.
- Conical (with cylindrical limit) is right for gears and roller bearings.
- Cylindrical is right for solenoids.
- Disk is right for hard, dense, moving, recording surfaces.
- Round is right for manhole covers. (They don't fall into the hole.)
- Straight is right for electric dipole antennas.
- Paraboloidal is right for simple reflectors focusing narrow beams.
- Cube corner is right for reflectors returning light to its source.

Engineering solutions selected in the above way have some shared characteristics. They generally:

- Get wide and enduring acceptance. Many are part of everyday life.
- Are robust. Secondary engineering values rarely reverse and often reinforce them.
Straight line road segments and spherical pressure vessels are exceptions.
- Match practical needs with theoretical ideals.

They contrast sharply with the much more common "tradeoff optima" such as: operating temperatures, rotation rates, recording densities, etc. where improvement of desired properties are judged to be balanced against increasing detrimental properties (excessive cost, risk of failure, etc). Eliminating simple redundancy or extraneous features could be viewed as rudimentary cases. They appear to provide some of the most primitive and the most useful engineering knowledge available.

In each case an idealized model of the engineering problem is used which leads to the irreducibility constraint. Any requirements ignored in the idealization will limit the range of applicability of the solution. The mathematical analysis leading to an irreducibility constraint can be rigorous, but the circumstances determining where it is applicable tend to involve empirical considerations which are less easily modeled.

There is variation among the solutions in the extent to which secondary engineering values lead to compromise of the optimum structure, such as:

- If a wheel is designed to be off-round (when not loaded) there will be vertical vibration which is almost never acceptable. (Under load, wheels usually flatten slightly to avoid damaging roads.)
- At the other extreme are spherical pressure vessels and linear road segments where a mathematically optimum structure is often compromised to meet secondary engineering requirements. The need for

speed and quietness discourage spherical submarines. There are many reasons to go around obstacles in road construction.

- In between are cases like tubular pipes and vertical pillars. It is all right for a tent pole to slope away from the vertical, but pillars supporting heavy buildings deviate very little from vertical. It is all right for low pressure heating ducts to be rectangular but pipes carrying high pressure fluids are consistently tubular. In general, the more demanding the requirements which lead to the constrained structure, the less acceptable compromises are.

Multiple irreducibility constraints may be combined in optimizing one design. An initial analysis of pillars favors centering the load vertically above the base. Further analysis favors straightness of the post in between. A third analysis favors a near uniform cross-section, or monotonicity of any tapering of the cross-section. The analysis for data objects provides a similar multiplicity of constraints.

Optimizations on a single part tend to be clearer and of wider significance than others. Some multi-part optima, like the triangular truss section (which eliminates bending forces on the sides), are widely significant while others, like the Wheatstone bridge for measuring impedances (which eliminates the need to accurately measure currents), are less so.

The constraints associated with H1, H2, H3 seem to fit the above pattern. The inherently discrete design of data objects assures local stability to any optimum. The familiarity and certainties of three dimensional space give an intuitive quality to most of the above irreducibility optima which is not available for data objects.

SIMPLIFICATIONS

The process of eliminating complexity appears to be analogous to removing heat from material where there is a gradual increase in orderliness punctuated with a series of phase changes from gas to liquid to solid etc. When simplifying system descriptions from their early forms, a series of nine possible phase changes or complexity implosions may be identified. Only the first two have been fully exploited. The others depend on understanding data object structure issues.

1. Language formalization

The early development of computability theory and stored program computers enables a transition from use of natural language to a simpler and well defined formal language.

2. Machine independent higher level language

Higher level language eliminated many machine oriented irrelevancies.

3. Functionally expressive language

When the variety of iteration mechanisms reduces to one that dominates, the complexity of system descriptions can be simplified by use of nested expressions on large aggregates of data. This has been exploited in languages like APL, Lisp, and Relational database.

4. Combining structurally and functionally expressive language

When data object structure is simplified, complexities caused by representational mismatch between data objects and conceptual objects can also be reduced. Combining these capabilities is a major departure from current technology.

5. Implosion of needed variety in basic language semantics

Functions on needles from any problem domain can coexist easily in one language. Mismatch between data base data and application data can be eliminated.

6. Simpler applications

Applications could expose their data structures and provide only the primary functionality needed. Rarely used secondary capabilities could be improvised by users, from a common underlying language. Both application development and learning by users can be reduced.

7. Easier information transfer between contexts

Complexity of integrating separately developed components is then reduced as a result of:

- uniformity of representations
- adaptability of simpler components
- adaptability of any needed interfacing "glue"
- simplification of adaptive transformations of components. Use of sub-optimum data objects can cause compounding of the associated complications.

8. Technological stability

Near optimum basic semantics would limit further disruptive changes.

9. Pre-integrated technical knowledge

In contrast to traditional mathematical representations, it becomes practical to fully present precise technical information precisely. At present students learning technical subjects must distill precise information from various informal presentation styles: diagrams, formulas, text, examples and integrate it. Readable, but complete and accurate formal language descriptions can reduce that burden and provide assurance that mystification can be cleared up. Simplifying the representation of software and simplifying the textual representation of technical subject matter is largely the same problem.

LANGUAGE GENERALITY

After some searching, the writer has found no evidence of enduring technical obstacles to generality of language data models or functional language semantics for rich applications. Wide acceptance of any common language is a separate non-technical issue.

Increasing the simplicity of expression and increasing generality of language semantics are two aspects of the same problem. The generality of a language is essentially a measure of the range of applications for which it provides adequate simplicity of expression. Increasing the potential simplicity of expression for a language will inherently broaden that range. There is no need to exploit potential simplicity to the point of being cryptic.

A continuing need for specialization and evolution of language vocabulary and syntax may be expected. Those more superficial aspects of language design need have no impact on a more stable underlying data model and function semantics. Type systems seem likely to remain problematic because they attempt to embody judgments about the validity of approaches to classification of real world phenomena. They affect the legality of writing and executing expressions in various contexts but need not otherwise affect the underlying semantics. Precise formal language could become more like natural language where specialized workers use jargon but as seamless extensions to a common unspecialized language.

It is conjectured that:

- All the functions below can be included in one fairly small language (almost all were included in KEEP[2]).
- Semantics of any deterministic specialized language can be straightforwardly expressed as an extension of such a language. However, some notations such as text markup languages would benefit little.
- The readability can be high enough to serve as a medium of technical communication and education for many subjects independent from computer implementations [6]. The function groups listed below include about 40 functions that are currently widely understood and another 20 that are less so. The combined set could be organized in a coherent language and used as part of common technical literacy and as a common foundation for many kinds of formalism.
- Long term interoperability for data access can be based on common data objects and very few additional common conventions for naming and basic functions.

A common core of unspecialized function groups could include:

- creation and deletion of objects and relationships

- fetching objects directly connected to or related to others
- arithmetic operations including comparison
- test for identicalness
- boolean operations
- conditional and case expressions
- unique selection based on key relationships (subscripting)
- subsetting by a selection condition
- subsetting by beginning or ending conditions
- set operations for union, intersection, difference
- testing sets for membership, inclusion, or overlap
- applying a function to each member of a set
- reduction, applying a binary function successively to the members of a set and the preceding result
- transitive closure
- sorting sets
- operations on sets of sets, statistical operations
- first order predicates over sets
- matrix operations
- relational algebra operations

TECHNICAL FACTORS OBSCURING OPTIMUM DATA PRIMITIVES

There are some wrong turns on the road to simplicity which have discouraged efforts to find it. One is the simplicity of Turing machines which lead to extremely complex programming problems and poor execution performance. Another is the allure of syntactic brevity which leads to obscure encoding conventions when taken to extreme.

Aversion to working with “low level representations” has diverted attention from the fine structure of information. The fine structure has been regarded as something to be covered up and escaped from rather than understood. Some interpretations of object-oriented software have viewed objects as “little computers that communicate”, thus obscuring the existence of an “atomic” fine structure for information. Of course, there is no way to work with information and escape from the structure or fine structure of its representation.

Confusion about the nature of abstraction has probably contributed. The traditional view that the abstract and the concrete are opposites tends to suggest that there is something hazy about abstractions. While artistic abstraction often softens detail, operational abstractions usually eliminate some detail while making other detail more hard-edged or exaggerated. Road maps often depict highways as having over ten times their actual width. When modeling an information system in a deterministic machine, the data objects must be sharply defined, explicit, and controllable. In many

ways, this amounts to being "concrete". The emphasis in developing abstraction seems to have focused overly on the removal of detail while neglecting the structure of remaining detail.

There are many reasons that bits seem like a natural basis for data objects. There has been a traditional association between the structure of computer storage and the structure of data objects. The question of how simple data objects can reasonably be was not a natural one to ask when sequences of bits seemed an unavoidable part of the answer. Quantities of information are measured in bits and information is increasingly delivered packaged in bits. There has been the need to stay close to storage oriented data models for efficiency and for providing a default data model which enables communication among diverse computer system components. The realities of physics also provide a distraction. Electrons are fast, small, well behaved, and irresistible as a means to implement physical memories, but they do not directly connect things. They are excellent for implementing bits but only provide the connections that are really needed indirectly through encoded addressing.

Libraries dominated by sequences of printed words are another distraction. The human throat is an inherently sequential device which produces one word at a time and the result is what gets recorded even though it is very different from the presumably more efficient organization inside the human brain. The natural development of nervous systems has emphasized rich connectivity but that style is beyond any manufacturing process.

ACKNOWLEDGMENT

Helpful discussions with Earl C. Van Horn and comments from William W. Collier and Cecil Chua Eng Huang are gratefully acknowledged.

REFERENCES

- [1] E. S. Lowry, "PROSE Specification", IBM Poughkeepsie Laboratory Technical Report TR 00.2902, Nov 1977.
- [2] E. C. Van Horn, "Expressing product development information in application terms", Proc. IEEE Int. Conf. Computer Design: VLSI in Computers, ICCD'85, Oct. 1985, pp. 82-85.
- [3] E.S. Lowry, "Toward an Optimum Language Data Model", Computer Standards & Interfaces 13(1991) p105-108.
- [4] E.S. Lowry, "Nodes and Arcs, The Ideal Primitive Data Structures", DEC/TR-114 November 1980, Digital Equipment Corporation.

[5] E.S. Lowry, "Accurate description of system structure - A new standard of language quality", Proceedings of the Digital Equipment Computer Users Society, Vol 5, No2, 1978, p833.

[6] E.S. Lowry, "Formal Language as a Medium for Technical Education", Proceedings of ED-MEDIA 96, p407, AACE, June 1996.

[7] John F. Sowa (editor), "Principles of Semantic Networks" Morgan Kaufmann, 1991,

[8] J. Peckham and F. Maryanski, "Semantic Data Models", ACM Computing Surveys, Vol 20, No 3, Sept 1988. p153

[9] D. C. Tschritzis and F. H. Lochovsky, "Data Models", Prentice Hall, 1982.

[10] P. Wegner, "Interaction as a Basis for Empirical Computer Science", ACM Computing Surveys, Vol 27, No 1, March 1995. p45.

APPENDIX: ILLUSTRATIONS OF SIMPLICITY OF EXPRESSION

An empirical approach to evaluating the hypotheses is to compare the simplicity of expression achievable using general procedural languages based on needles and alternative data object structures. The following examples illustrate the level of simplicity of expression which can be obtained in KEEP and which would need to be approximated in an alternative language to raise doubts about the hypotheses.

These examples are translated from the first 10 examples given for Sequel 2 (now SQL) in the IBM Journal of R&D [5]. For the first 10 expressions, Sequel 2 (a specialized data base language) uses 130 tokens. KEEP (a multi-purpose language) uses 99 tokens. The original Sequel 2 code is omitted as irrelevant for this purpose. The functions for "show", "condense", and dynamically enumerated names have not been implemented.

Expression 1.

English:

Names of employees in Dept. 50

KEEP:

name of employee of dept(50)

Expression 2.

Eng:

All the different department numbers in the Employee table.

KEEP:

dept_no of employee condense

Expression 3.

Eng:

Names of employees in Depts. 25, 47 and 53.

KEEP:

name of employee of every dept where 25 or 47 or 53

Expression 4.

Eng:

Names of employees who work for departments in Evanston.

KEEP:

name of employee of dept of Evanston

Expression 5.

Eng:

List the employee number, name and salary of employees in Dept. 50, in order of employee number.

KEEP:

for employee of dept(50) minfirst empno show(empno, name, salary)

Expression 6.

Eng:

Average salary of clerks.

KEEP:

average (salary of clerk)

Expression 7.

Eng:

Number of different jobs that are held by employees in Dept. 50

KEEP:

count job of employee of dept(50) condense

Expression 8.

Eng:

List all the departments and the average salary of each.

KEEP:

for dept show(it, average(salary of its employee))

Expression 9.

Eng:

Those departments in which the average employee salary is less than 10,000.

KEEP:

dept where average(salary of its employee) < 10000

Expression 10.

Eng:

The departments that employ more than ten clerks.

KEEP:

dept where count(its clerk) > 10