

On Knowledge Amplification by Structured Expert Randomization (KASER)

Stuart H. Rubin
SSC San Diego

INTRODUCTION TO RANDOMIZATION

The theory of randomization was first published by Chaitin and Kolmogorov [1] in 1975. Their work may be seen as a consequence of Gödel's Incompleteness Theorem [2] in that it shows were it not for essential incompleteness, a universal knowledge base could, in principle, be constructed—one that need employ no search other than referential search. Lin and Vitter [3] proved that learning must be domain-specific to be tractable. The fundamental need for domain-specific knowledge is in keeping with Rubin's proof of the Unsolvability of the Randomization Problem [4]. This paper went on to introduce the concept of knowledge amplification. Production rules are expressed in the form of situation action pairs. Such rules, once discovered to be in error, are corrected through acquisition. Conventionally, a new rule must be acquired for each correction. This is linear learning.

The acknowledged key to breakthroughs in the creation of intelligent software is cracking the knowledge acquisition bottleneck [5]. Learning how to learn is fundamentally dependent on representing the knowledge in the form of a society of experts. Minsky's seminal work here led to the development of intelligent agent architectures [6]. Furthermore, Minsky [7] and Rubin [4] independently provided compelling evidence that the representational formalism itself must be included in the definition of domain-specific learning if it is to be scalable.

A KASER is defined to be a knowledge amplifier that is based on the principle of structured expert randomization. A Type I KASER is one where the user supplies declarative knowledge in the form of a semantic tree using single inheritance.

A Type II KASER can automatically induce this tree through the use of randomization and set operations on property lists, which are acquired by way of database query and user-interaction. An overview of a Type II KASER is provided below. Unlike conventional intelligent systems, KASERs are capable of accelerated learning in symmetric domains.

Figure 1 plots the knowledge acquired by an intelligent system vs. the cost of acquisition. Conventional expert systems will generate the curve below break-even. That is, with conventional expert systems, cost increases with scale and is never better than linear. Compare this with KASERs where cost decreases with scale and is always better than linear unless the domain has no symmetries (i.e., it is random). Note that such

ABSTRACT

We define Knowledge Amplification by Structured Expert Randomization (KASER). A KASER can automatically acquire a virtual rule space exponentially larger than the actual rule space and with an exponentially decreasing nonzero likelihood of error. The KASER cracks the knowledge acquisition bottleneck in intelligent systems by amplifying user-supplied knowledge. This enables the construction of an intelligent system, which is creative, fail-soft, learns over a network, and otherwise has enormous potential for automated decision-making.

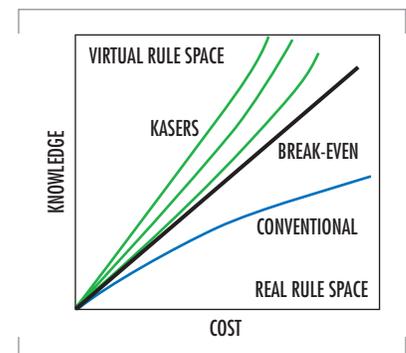


FIGURE 1. The comparative costs of knowledge acquisition.

domains do not exist with scale in practice. Similarly, purely symmetric domains do not exist with scale in practice either. The more symmetric the operational domain, the less the cost of knowledge acquisition and the higher the curve appears in the graph. It is always the case that the virtual rule space \gg the real rule space.

INDUCING PROPERTY LISTS

We will define a production system that can automatically acquire a virtual rule space that is exponentially larger than the actual rule space with an exponentially decreasing non-zero likelihood of error. Moreover, the generalization mechanism will not only be bounded in its error, but other than for a straightforward user-query process, it will operate without any *a priori* knowledge supplied by the user.

To begin, define a production rule (e.g., using ANSI Common LISP) to be an ordered pair—the first member of which is a set of antecedent predicates, and the second member of which is an ordered list of consequent predicates. Predicates can be numbers (e.g., $[1..2] \vee [10..20]$) or words [8].

Previously unknown words or phrases can be recursively defined in terms of known ones. For example, the moves of a Queen in chess (i.e., unknown) can be defined in terms of the move for a Bishop (i.e., known) union those for a Rook (i.e., known). This is a union of property lists. Other basic set operations may likewise be used (e.g., intersection, difference, not, etc.). The use of fuzzy set operators here (e.g., "almost the same as") pertains to computing with words [8].

In a Type I KASER, words and phrases are entered through the use of pull-down menus. In that manner, semantically identical concepts (e.g., Hello and Hi) are not ascribed a distinct syntax, which would otherwise serve to dilute the efficiency of the learning mechanism. In a Type II KASER, distinct syntax may be equated to yield the equivalent normalized semantics. To better visualize this, think of a child who may ask, "What is a bird?" to which the reply is, "It is an animal that flies," to which the question is, "What is an animal?" to which the reply is, "It is a living thing," to which the question is, "What is a living thing?" to which the reply (often) is, "Eat your soup!" (i.e., a Type I delimiter, or stop marker gene).

Two sample rules and their representation follow.

Hydrogen \wedge Oxygen \wedge Spark \rightarrow Steam

R1: ({Hydrogen, Oxygen, Spark} (Steam))

Hydrogen \wedge Oxygen \wedge Match \rightarrow Steam

R2: ({Hydrogen, Oxygen, Match} (Steam))

R1 and *R2* may be generalized, since the consequent predicates are identical (i.e., the right-hand sides [RHSs] are equivalent) and the antecedent terms differ in exactly one predicate. This is termed a level-1 generalization because it is one level removed from ground truth. In a level-*i* generalization, *i* is the maximum level of generalization for any antecedent predicate. The need for a generalization squelch arises because contexts may be presented for which there is no matching rule in the real space. Generalizations can be recursively defined.

The advocated approach captures an arbitrary rule's context—something that cannot be accomplished through the use of property lists alone. If veristic terms such as "Warm" are generalized to such terms as "Heat" for example, then qualitative fuzziness will be captured.

A1: ({Heat} {Spark, Match})(X001 Explosive-Gas-Igniter))

Generalization, *A1*, tells us that antecedent predicate, "Heat" is more general than either a Spark or a Match. We may also write this as *Heat* > {Spark, Match}. Note that the relation ">" is used to denote ancestral generalizations (and vice versa). The general predicate is initially specified as *X00i*, but this is replaced after interactive query with the user, where possible. Otherwise, the next-level expansions will need to be printed for the user to read. Also, "redundant, at-least-as-specific" rules are always expunged.

The common property list follows the set of instances. Here, the list informs us that a spark or a match may be generalized to Heat because both share the property of being an Explosive-Gas-Igniter. Properties are dynamic. They must be capable of being hierarchically represented, augmented, and randomized. In addition, property lists are subject to set operations (e.g., intersection). Properties can be acquired by way of database and/or user query.

User-queries can be preprocessed by a companion veristic mining system. Similarly, system-generated queries can be post-processed by companion systems. Companion systems can also play a role in imparting tractability to the inference engine.

Consequent terms, being sequences, are taken to be immutable. The idea here is to automatically create a hierarchy of consequent definitions to maximize the potential for rule reuse. Begin by selecting a pair of rules having identical left-hand sides (LHSs), where possible. Consider:

R3: ({Hydrogen, Oxygen, Heat} (Steam))

R4: ({Hydrogen, Oxygen, Heat} (Light, Heat))

Next, an attempt is made to generalize the consequent sequences with the following result.

C1: ((Energy) ((Steam) (Light Heat))(X002 Power-Source))

Here, the properties of Steam intersect those of Light and Heat to yield the property, Power-Source. Thus, a property of Energy, in the current context at least, is that it is a Power Source. Rules *R3* and *R4* are now replaced by their valid generalization, *R5*:

R5: ({Hydrogen, Oxygen, Heat} (Energy))

A key concept is that further learning can serve to correct any latent errors. In addition, notice that as the level of randomization increases on the LHS and RHS, the potential for matching rules, and thus inducing further generalizations, increases by way of feedback. Consequent randomization brings the consequents into a normal form, which then serves to increase the possibility of getting antecedent generalizations, since more RHSs can be equated. Antecedent randomization is similar.

Next, consider *R5*, where *R6* is acquired and appears as follows after substitution using *C1*.

R6: ({Candle, Match} (Energy))

The system always attempts to randomize the knowledge as much as possible. Using *A1* and *C1* leads to the level-1 conjecture, *R7*, which replaces *R6*.

R7: ({Candle, Heat} (Energy))

R7 is not to be generalized with *R6*. This is because {Match, Heat} is the same as {Match, Spark, Match}, which of course reduces to Heat and is already captured by *R7*.

At this point, learning by the system can be demonstrated. Suppose the user asks the system what will happen if a spark is applied to a candle. While this is a plausible method to light a candle, this method will not usually be successful. Thus, the user must report to the system the correct consequent for this action:

R8: ({Candle, Spark} (No-Light))

R8 is a more-specific rule than is *R7* because the former is a level-1 generalization, while the latter is at level-0. Thus, *R8* will be preferentially fired when possible by using a most-specific agenda mechanism. It, too, will be subject to subsequent generalization. Notice that the new consequent will protect against similar error.

The learning process has not completed. We still need to correct the properties list so that Matches and Sparks can be differentiated in the context of lighting a candle. The following property (i.e., LISP) list is obtained.

P1: (Match Explosive-Gas-Igniter Wick-Lighter)

P2: (Spark Explosive-Gas-Igniter)

Now, since Heat is a superclass of Match, its property list is unioned with the new property(s): Wick-Lighter. Suppose, at this point, the user poses the same question, "What will happen if a spark is applied to a candle?" Rule *R7* informs us that it will light; whereas, *R8* informs us that it will not. Again, the inference engine can readily select the appropriate rule to fire because of specialization. However, here there is yet more to learn. Here is what is known: *R7* and *R8* differ on the LHS in exactly one predicate and $prop(Heat) \cap prop(No - Light) = \emptyset$. The reason that the candle lights for a match, but not for a spark can be delimited by computing, $prop(Match) - prop(Spark) = prop(P1) - prop(P2) = (Wick-Lighter)$. Rule *R7* is now replaced by *R7'*:

R7': ({Candle, (X003 Wick-Lighter)} (Energy))

that is, a property list named X003 has been substituted for Heat. Notice that X003 is necessarily a subclass of Heat. Then, anything that has (all) the properties on the property list (i.e., X003) can presumably light a candle (e.g., a torch). Observe that the human in the loop need not know why a list of properties is relevant, since the reasons will be automatically discovered. Notice that a Spark can no longer light a candle and only those items having at least Wick-Lighter in their property classes can light a candle. Observe the nonlinear learning that has been enabled here!

Consider now the rule:

R9: ({Candle, Match} (Energy))

Clearly, this rule is correct as written. Candles do indeed produce steam, light, and heat. The usefulness of induction follows from the fact that the

system has no knowledge that a candle is a hydrocarbon and hydrocarbons produce steam as a byproduct of combustion.

Antecedent predicate generalizations can be rendered more class-specific as necessary to correct overgeneralizations by increasing the number of levels of available generalization. The rule consequents will not be affected. For example:

A2: ({Car} {Ford, Fiat})

yields:

A3: ({Car} {Family-Car, Sports-Car})

A4: ({Family-Car} {Ford})

A5: ({Sports-Car} {Fiat})

Property lists can be automatically organized into a hierarchical configuration through the use of simple set operations. This means that rules can be generalized or specialized through the use of the disjunctive or conjunctive operators, respectively. Such property lists can be associatively retrieved through the use of a grammatical randomization process [9]. Moreover, matching operations then need to incorporate searching subclasses and superclasses as necessary.

Finally, we note that this system can incorporate fuzzy programming [10]. Fuzzy programming will enable the system to explore a space of alternative contexts as delimited by optional consequent filters and ranked by the level of generalization used to obtain a contextual match (see below).

GRAMMATICAL RANDOMIZATION

Consider the following three property lists:

P3: (Ice A B C)

P4: (Water B C D)

P5: (Steam C E)

Here, ice, water, and steam share a common property, C, which, for example, might be that they are all composed of H_2O . Also, only ice has property A (e.g., frozen); only water has property D (e.g., liquid); and only steam has property E (e.g., gaseous). Observe that only ice and water share property B (e.g., heavier-than-air).

The use of a hierarchical object-representation is fundamental to the specification of property lists, antecedent sets, or consequent sequences. For example, when one specifies the object, "aircraft carrier," one implicitly includes all of its capabilities, subsystems, and the like. One cannot and should not have to specify each subsystem individually. We proceed to develop a randomization for the sample property lists; although, it should be clear that the same approach will work equally well for the antecedent and consequent predicates. Perhaps the most relevant distinction is that one needs to distinguish object sequence dependence from independence in the notation. Of course, property lists are sequence-independent.

As the example stands now, to specify the properties of ice or water, one need state the three properties of each (in any order). This may not seem

too difficult, but this is only because the list-size is small. Consider now the randomized version of the property lists:

P3: (Ice A Precipitation)

P4: (Water Precipitation D)

P5: (Steam C E)

P6: (Precipitation B C)

Here, the property of precipitation has been randomized from the property data. Observe, that if the user states property B, then the system will offer the user exactly three choices (e.g., by way of a dynamic pull-down menu): B, Precipitation, or Random. The Random choice allows the user to complete the specification using arbitrary objects. In other words, an associative memory has been defined. Similarly, if the user selects Precipitation, then the system will offer the user exactly four choices (e.g., again by way of a dynamic pull-down menu): Precipitation, Ice, Water, or Random.

Suppose that in keeping with the previously described nomenclature conventions, we had the following property list specifications:

P3': (X004 A Precipitation)

P4': (X005 Precipitation D)

In this case, if the user selects Precipitation, then the system will offer the user the following four choices: Precipitation, A, D, or Random. In other words, it attempts to pattern-match and extrapolate the set.

In practice, randomization is based on known classifications—not arbitrary ones. Thus, in the previous example, the randomization of *P3* and *P4* requires that *P6* be known *a priori*. Again, this still allows for the use of integer identifiers.

Next, it can be seen that the usefulness of randomization is a function of its degree. The relevant question then pertains to how to realize the maximal degree of randomization. First, recall that as rules are generalized, the possibilities for further predicate generalization are increased. This, in turn, implies that the substitution and subsequent refinement of property lists for predicates is increased. Finally, as a result, the virtual space of properly mapped contexts (i.e., conjectures) grows at a rapid rate. Experimental evidence to date indicates that this rate may be exponential for symmetric domains.

Next, we turn our attention to the inference engine, which is common to Type I and II KASERs. Basically, in a Type I KASER, conflict resolution is accomplished through the use of a hierarchical tree of objects evolved by a knowledge engineer, which define generalization and specialization (see below); whereas, in a Type II KASER, conflict resolution is the same as in a Type I KASER, but where the system, instead of the knowledge engineer, evolves hierarchical property lists, which serve to increase the size of the virtual contextual space—without sacrificing convergence in the quality of the response. In effect, declarative knowledge is randomized to yield procedural knowledge.

ACTIVE RANDOMIZATION

Active randomization is a symbiosis of property lists and grammatical randomization. Property lists are really just predicates that are subject to

grammatical randomization. Moreover, randomized predicates allow the user to specify contexts and associated actions by using minimal effort [9]. Next, suppose that we had:

(A B C D) (i.e., the properties of A are B, C, D)

Here, A is the randomization of B, C, D. Similarly, we may have

(B E F)

and the two rules (i.e., antecedent differentiation):

$R10: A S \rightarrow W$

$R11: X S \rightarrow W$

Then, we can create a randomization:

(Q A X)

which, since valid, leads to the following replacement of $R10$ and $R11$:

$R12: Q S \rightarrow W$

This replacement allows for the possibility of new rule pairings and the desired process then iterates. Thus, we have

(Q: A \cup X) {expanding A, X}

These are *active transforms* [9] in the sense that whenever A or X change their membership, the properties of Q may change. Evidently, this is a converging process. However, if subsequently we had

$R13: A S \rightarrow T$

$R14: X S \rightarrow G$

where, T and G have no properties in common (i.e., neither is a subsequence of the other), then it becomes clear that A cannot substitute for X and vice versa. In other words,

$R13: (A-X) S \rightarrow T$

$R14: (X-A) S \rightarrow G$

Thus, we have

(A: A - X) {contracting A}

(X: X - A) {contracting X}

These are active transforms, and again, this is a converging process. Next, suppose that T and G are such that G is a subsequence of T without loss of generality. Then, it follows that A is a subset of X and

(A: A \cap X) {contracting A}

(X: X \cup A) {expanding X}

These are active transforms. This is not, however, necessarily a converging process. That is not to say that it will diverge without bounds. It is just not stable. We do not view this as a problem. It is to be viewed as an oscillatory system that, in some ways, may mimic brain waves. The complexity of interaction will increase as the system is scaled up. The eventual need for high-speed parallel/distributed processing is apparent. The case for consequent differentiation is similar. Here though, one is processing sequences instead of sets.

OBJECT-ORIENTED TRANSLATION MENUS

The Type I KASER requires that declarative knowledge be (dynamically) compiled in the form of object-oriented hierarchical phrase translation

menus. Each class (i.e., antecedent and consequent) of bipartite predicates can be interrelated through their relative positions in an object-oriented semantic tree. A declarative knowledge of interrelatedness provides a basis for commonsense reasoning, as will be detailed in the next section. The subject of this section pertains to the creation, maintenance, and use of the object-oriented trees as follows.

1. The phrase-translation menus serve as an intermediate code (as in a compiler) where English sentences can be compiled into menu commands by using rule-based compiler bootstraps. KASERs can be arranged in a network configuration where each KASER can add (post) to or delete from the context of another. This will greatly expand the intelligence of the network with scale and serves to define Minsky's "Society of Mind" [6]. Furthermore, the very-high-level domain-specific language(s) used to define each predicate can be compiled through a network of expert compilers. Alternatively, neural networks can be used to supply symbolic tokens at the front end.
2. Each antecedent or consequent phrase can be associated with a textual explanation, Microsoft's Text-to-Speech engine (Version 4.0), an audio file, a photo, and/or a video file. Images may be photographs, screen captures, scans, drawings, etc. They may also be annotated with arrows, numbers, etc. Voice navigation may be added at a later date.
3. Antecedents and consequents can be captured by using an object-oriented approach. The idea is to place descriptive phrases in an object-oriented hierarchy such that subclasses inherit all of their properties from a unique superclass and may include additional properties as well. Menus can beget submenus, and new phrases can be acquired at any level.

Consider the partial path, office supply, paper clip and the partial path, conductor, paper clip. Here, any subclass of paper clip will have very different constraints depending on its derivation. For example, anything true of paper clips in the context of their use as conductors must hold for every subclass of paper clips on that path. Unique antecedent integers can be set up to be triggered by external rules. Similarly, unique consequent integers can be set up to fire external procedures. All we need do is facilitate such hooks for future expansion (e.g., the radar-mining application domain).

Each project is saved as a distinct file, which consists of the antecedent and consequent trees, the associated rule base, and possibly the multimedia attachments.

4. A tree structure and not a graph structure is appropriate because the structure needs to be readily capable of dynamic acquisition (i.e., relatively random phrases) and deletion, which cannot be accomplished in the presence of cycles due to side effects. Note that entering a new phrase in a menu implies that it is semantically distinct from the existing phrases, if any, in that menu.
5. A tree structure is mapped to a context-free grammar (CFG), where the mapping process needs to be incremental in view of the large size of the trees. Each node or phrase is assigned a unique number, which serves to uniquely identify the path.
6. Each phrase may be tagged with a help file, which also serves the purposes of the explanation subsystem. This implies that conjuncts are not necessary to the purpose of the antecedent or consequent trees.

7. Each menu should be limited to on the order of one screen of items (e.g., 22). Toward this end, objects should be dynamically subdivided into distinct classes. That is, new submenus can be dynamically created and objects moved to or from them.
8. Three contiguous levels of hierarchy should be displayed on the graphical user interface (GUI) at any time, if available.
9. A marker gene or bookmark concept allows the user to set mark points for navigational purposes.
10. A list of recently visited menus serves to cache navigational paths for reuse.
11. A global find mechanism allows the user to enter a phrase and search the tree from the root or present location and find all matches for the phrase up to a prespecified depth. The path, which includes the phrase, if matched, is returned.
12. Entered phrases (i.e., including pathnames) can be automatically extrapolated where possible. This "intellisense" feature facilitates keyboard entry. It can also assist with the extrapolation of pathnames to facilitate finding or entering a phrase. Pathname components may be truncated to facilitate presentation.
13. A major problem in populating a tree structure is the amount of typing involved. In view of this, copy, paste, edit, and delete functions are available to copy phrases from one or more menus to another through the use of place-holding markers. Phrase submenus are not copied over because distinct paths tend to invalidate submenu contents in proportion to their depth. Again, new integers are generated for all phrases. Note that the returned list of objects still needs to be manually edited for error and/or omissions. This follows from randomization theory. This maps well to natural language translation.
14. Disjuncts in a menu serve as analogs and superclasses serve as generalizations for an explanation subsystem. In addition, help files and pathnames will also serve for explanative purposes.
15. An "intellassist" feature allows the system to predict the next node in a contextual, antecedent, or consequent tree. Each node in a tree locally stores the address (number) of the node to be visited next in sequence. If a node has not been trained, or if the pointed-to address has been deleted without update, then a text box stating "No Suggestion" pops up, and no navigation is effected if requested. Otherwise, potentially three contiguous menus are brought up on the screen, where the farthest right menu contains the addressed node. Navigation is accomplished by clicking on a "Suggest" button. Otherwise, all navigation is manually performed by default. The user can hop from node to node by using just the suggest button without registering an entry. The use of any form of manual navigation enables a "Remember" button immediately after the next term, if any is entered. Clicking on this enabled button will result in setting the address pointed to by the *previously entered* node to that of the *newly entered* node. The old pointer is thus overwritten. Note that this allows for changing the item selected within the same menu. Note, too, that if a node (e.g., Toyota) is deleted, then all pointers to it may be updated to the parent class (e.g., car menu) and so on up the tree (e.g., vehicle type menu).

A pull-down menu will enable one of two options: (1) Always Remember (by default) and (2) Remember when Told. The Remember button is not displayed under option (1), but the effect under this option is to click it whenever it would have otherwise been enabled. The system always starts at the root node of the relevant tree.

16. It does not make sense to retain a historical prefix for use by the intellassist feature. That is, there is no need to look at where you were to determine where you want to go. While potentially more accurate, this increase in accuracy is more than offset by the extra training time required, the extra space required, and the fact that it will take a relatively long time to reliably retrain the nodes in response to a dynamic domain environment.

AN A* ORDERED SEARCH ALGORITHM

Expert compilers apply knowledge bases to the effective translation of user-specified semantics [11]. The problem with expert compilers is that they use conventional expert systems to realize their knowledge bases. A KASER is advocated because it can amplify a knowledge base by using an inductively extensible representational formalism.

Here, we present a relatively high-level view of the KASER algorithm. We claim that it represents a great advance in the design of intelligent systems by reason of its capability for symbolic learning and qualitative fuzziness:

1. Click on antecedent menus to specify a contextual conjunct. Alternatively, a manual "hot button" will bring up the immediately preceding context for reuse or update. Renormalization is only necessary if a generalization was made—not for term deletion (see below). Iteratively normalize the context (i.e., reduce it to the fewest terms) by using the tree grammar. Note that contextual normalization can be realized in linear time in the number of conjuncts and the depth of search. Here are the reduction rules, which are iteratively applied in any order—allowing for concurrent processing:
 - a. $S \rightarrow A \mid B \mid C \dots$ then replace $A, B, C \dots$ with S just in case all of the RHS is present in the context. This step should be iteratively applied before moving on to the next one.
 - b. $S \rightarrow A \dots$ and $A \rightarrow B \dots$ and $B \rightarrow C \dots$ then if S, A, B, C are all present in the context, then remove A, B, C since they are subsumed by S . It is never necessary to repeat the first step after conclusion of the second.
2. Compute the specific stochastic measure. Note that the specific stochastic measure does not refer to validity—only to the creative novelty relative to the existing rules while retaining validity. For example, given the antecedent grammar: $C5 \rightarrow C3 \mid C4$; $C4 \rightarrow C1 \mid C2$:
 - a. $\{C3 \ C1\} \{\{C3\}, \{C2 \ C3\},\}$ covers and matches the first $\{C3\}$ at level 0. Note that the first covered match, if any, that does not have a covered superset is the one to be fired—a result that follows from the method of transposition.
 - b. $\{C3 \ C1\} \{\{C5\}, \{C2 \ C3\},\}$ matches nothing at the level 0 expansion, so we expand the RHS with the result, $\{C3 \ C1\} \{\{^*C5 \ (C3 \ C4)\},\}$

- {*C2 *C3}, where the C2 C3 are both primitives and *Ci can be matched, but not expanded again. (..) is used to denote disjunction. Here, {C5} is matched at level 1. Note that at any level, only one term inside the parentheses (e.g., C3) need be covered to get a match of any one disjunct.
- c. {C3 C6} {{C2}, {C5 C6},} matches nothing at the level 0 expansion, so we expand the RHS with the result, {C3 C6} {{*C2}, {*C5 (C3 C4), *C6}}, which matches at level 1 because we matched (C3 OR C4) AND C6. Note that C6 was never expanded because it was pre-matched by the existing context. This economy is possible as a result of pre-normalizing the context.
 - d. The result of applying the method of transposition to the above step is {{C5 C6}, {C2},}.
 - e. Each matched {...} fires a consequent, which, if not primitive, matches exactly one row header (i.e., a unique integer) and step (2) iterates.
 - f. Maintain a global sum of the number of levels of expansion for each row for each consequent term. The specific stochastic measure is taken as the maximum of the number of levels of expansion used for each consequent term.
3. Exit the matching process with success (i.e., for a row) or failure based on reaching the primitive levels, a timer-interrupt, a forced interrupt, and/or by using the maximum allocated memory.
 4. If a sequence of consequent actions has been attached, then the sequence is pushed onto a stack in reverse order such that each item on the stack is expanded in a depth-first manner. A parenthesized sequence of actions will make clear the hierarchy. For example, ((Hold Writing Instrument (Hold Pencil with Eraser)) (Press Instrument to Medium (Write Neatly on Paper))). Here, the subclasses are nested. Such a representation also serves explanative purposes. Thus, here one has, Hold Writing Instrument, Press Instrument to Medium, at the general level, and Hold Pencil with Eraser, Write Neatly on Paper, at the specific level. A companion intelligent system could transform the conceptual sequences into smooth natural language (e.g., Pick up a pencil with an eraser and write neatly on a sheet of paper.) Set the general stochastic measure (GSM) to zero. Note that the stochastic measures for each predicate are computed and held in a data structure. The data will be used by the inference engine.
 5. If a match is not found, then since we already have an expanded antecedent {...}, we proceed to expand the context in a breadth-first manner (i.e., if enabled by the level of permitted generalization). Compute the general stochastic measure. Initialize the general stochastic measure to GSM. Note that the general stochastic measure is a measure of validity. Set the starting context to the context.
 - a. A specialized match was sought in step (2), and a generalized match is sought here. Expanding the context can lead to redundancies. For example, {*C1 *C2 *C3 *C4 C1 C2}. Here, the solution is to simply not include any term that is already in the (expanded) context. Stochastic accuracy is thus preserved. Any method that does not preserve stochastic accuracy is not to be used.

- b. $\{C5\} \{\{ *C3\}, \{ *C2 *C3\},\}$ failed to be matched in step (2), so a level 1 expansion of the context is taken:
 $\{ *C5 C3 C4\} \{\{ *C3\}, \{ *C2 *C3\},\}$ where C3 is matched at level 1.
- c. $\{C5 C6\} \{\{ *C1\}, \{ *C2 *C3\},\}$ matches nothing at level 0, so a level 1 expansion of the context is taken:
 $\{ *C5 C3 C4 *C6\} \{\{ *C1\}, \{ *C2 *C3\},\}$ matches nothing at level 1, so a level 2 expansion of the context is taken:
 $\{ *C5 *C3 *C4 C1 C2 *C6\} \{\{ *C1\}, \{ *C2 *C3\},\}$ matches C1 OR C2 AND C3 at level 2. The first covered set is the one to be fired (i.e., even though both sets are covered), since it does not have a covered superset. Next, the method of transposition is trivially executed with no resulting change in the logical ordering.
- d. Each matched {...} fires a consequent, which, if not primitive, matches exactly one row header (i.e., a unique integer) and step (2) iterates.
- e. One should maintain a count of the maximum number of levels of expansion for the context below the initial level. The general stochastic measure is defined by GSM plus the maximum number of levels that the context minimally needs to be expanded to get the "first" (i.e., method of transposition) match. This stochastic is represented by the maximum depth for any expansion.
- f. If the context fails to be matched, then generalize each term in the starting context one level up in the tree. Remove any redundancies from the resulting generalization. If the generalized context differs from the starting context, then add one to GSM and go to step (5). Otherwise, go to step (6). For example, the starting context $\{C2 C3\}$ is generalized to yield $\{C4 C5\}$. If this now covers a {...}, then the general stochastic measure is one. Otherwise, it is subsequently expanded to yield $\{ *C4 C1 C2 *C5 C3 C4\}$ at the first level. Notice that the second C4 has a longer derivation, is redundant, and would never have been added here. Note, too, that C4 is also a sibling or analog node. If this now covers a {...}, then the GSM remains one, but the specific stochastic measure is incremented by one to reflect the additional level of specialization.
- For another example, Toyota and Ford are instances of the class car. If Toyota is generalized to obtain car, which is subsequently instantiated to obtain Ford (i.e., an analog), then the general and specific stochastic measures would both be one. The general stochastic measure represents the number of levels of expansion for a term in one direction, and the specific stochastic measure represents the number of levels of expansion from this extrema in the opposite direction needed to get a match. The final general (specific) stochastic is taken as the maximum general (specific) stochastic over all terms.
- g. Conflict resolution cannot be a deterministic process as is the case with conventional expert systems. This is because the number of predicates in any match must be balanced against the degree of specialization and/or generalization needed to obtain a match. Thus, a heuristic approach is required. The agenda mechanism will order the rules by their size, general stochastic, and specific stochastic with recommended weights of 3, 2, and 1 respectively.

6. Exit the matching process with success (i.e., for the entire current context for a row) or failure based on reaching the primitive levels, a timer-interrupt, a forced interrupt, and/or by using the maximum allocated memory. Note that a memory or primitive interrupt will invoke step (5f). This enables a creative search until a solution is found or a timer-interrupt occurs. Note, too, that it is perfectly permissible to have a concept appear more than once for reasons of economy of reference, or to minimize the stochastic measures (i.e., provide confirming feedback). The stochastic measures also reflect the rapidity with which a concept can be retrieved.
7. Knowledge acquisition:
 - a. Note that new rules are added at the head.
 - b. If exit occurs with failure, or the user deems a selected consequent (e.g., in a sequence of consequents) in error (i.e., trace mode on), then the user navigates the consequent menus to select an attached consequent sequence, which is appropriate for the currently normalized context.
 - c. If the user deems that the selected "primitive" consequent at this point needs to be rendered more specific, then a new row is opened in the grammar, and the user navigates the consequent menus to select an attached consequent sequence.
 - d. A consequent sequence can pose a question, which serves to direct the user to enter a more specific context for the next iteration (i.e., conversational learning). Questions should usually only add to the context to prevent the possibility of add/delete cycles.
 - e. Ask the user to eliminate as many specific terms (more general terms will tend to match more future contexts) from the context as possible (i.e., and still properly fire the selected consequent sequence given the assumptions implied by the current row). A context usually consists of a conjunct of terms. This tends to delimit the generality of each term as it contributes to the firing of the consequent. However, once those antecedent terms become fewer in number for use in a subsequent row, then it becomes possible to generalize them while retaining validity. The advantage of generalization is that it greatly increases reusability. Thus, we need to afford the user the capability to substitute a superclass for one or more terms. Note that this implies that perfectly valid rules that were entered can be replayed with specific (not general) stochastics greater than zero. This is proper, since the specific stochastic preserves validity in theory. Thus, the user may opt to generalize one or more contextual terms by backtracking their derivational paths. If and only if this is the case, step (1) is applied to normalize the result. An undo/redo capability is provided. Validated rule firings are only saved in the rule base if the associated generalization stochastic is greater than zero. The underlying assumption is that rule instances are valid. If a pure rule instance proves to be incorrect, then the incorrect rule needs to be updated or purged, and the relevant object class menu(s) may be in need of repair. For example, what is the minimal context to take FIX_CAR to FIX_TIRE? A companion intelligent system could learn to eliminate and otherwise generalize specific terms (e.g., randomization theory).

- f. The system should verify for the user all the other {...} in the current row that would fire or be fired by the possibly over-generalized {...} if matched. (Note that this could lead to a sequence of UNDOs.)

For example, ({C5} A2) ({C5} A1) ({C5 C6} A2) ({C5 C7} A2, A3) informs the user that if the new C5 acquisition is made, then A2 and not A1 is proper to fire. If correct, then the result is {{C5} A2 {C5 C7} A2, A3}. {C5 C6} A2 has been eliminated because it is redundant. Also, {C5 C7} A2, A3 is fired just in case C5 AND C7 are true—in which case, it represents the most specific selection since it is a superset of the first set. If the elimination of one or more specific terms causes one or more {...} to become proper supersets, then warning message(s) may be issued to enable the user to reflect on the proposed change(s). If the elimination and/or generalization of one or more specific terms enables the firing of another rule in the same row in preference to the generalized rule, then the generalization is rejected as being too general. Note that there is no need to normalize the results, as they would remain in normal form. Also, any further normalization would neutralize any necessary speedup.

- g. A selected consequent number may not have appeared on the trace path with respect to the expansion of each consequent element taken individually. Checking here prevents cycle formation.
- h. It should never be necessary to delete the least frequently used (LFU) consequent {...} in view of reuse, domain specificity, processor speed, and available memory relative to processor speed. Nevertheless, should memory space become a premium, then a hierarchy of caches should be used to avoid deletions.
8. A metaphorical explanation subsystem can use the antecedent/consequent trees to provide analogs and generalizations for explanative purposes. The antecedent/consequent paths (e.g., ROOT, FIX_CAR, FIX_TIRE, etc.) serve to explain the recommended action in a way similar to the use of the antecedent and consequent menus. The antecedent/consequent menus will provide disjunction and "user-help" to explain any level of action on the path. Note that the system inherently performs a fuzzy logic known as computing with words [4] (i.e., based on the use of conjuncts, descriptive phrases, and tree structures). The virtual rule base is exponentially larger than the real one and only limited by the number of levels in the trees, as well as by space-time limitations on breadth-first search imposed by the hardware.
9. A consequent element could be a "do-nothing" element if need be (i.e., a Stop Expansion). The provision for a sequence of consequents balances the provision for multiple antecedents. The selected consequent(s) need to be as general class objects as can be to maximize the number of levels and, thus, the potential for reuse at each level. The consequent grammar is polymorphic since many such grammars can act (in parallel via the Internet) on a single context with distinct, although complementary results. Results can be fused as in a multi-level, multicategory associative memory. Multiple context-matched rules may not be expanded in parallel because there can be no way to ascribe *probabilities* to partially order the competing rules and

because any advantage would be lost to an exponential number of context-induced firings. The consequent {...}s cannot be ranked by the number of matching terms (i.e., for firing the most specific first) because the most specific terms are generally incomparable. However, a covered superset is always more specific than any of its proper subsets. Thus, the first covered set that does not have a covered superset in the same row is the one to be fired. If it does have a covered superset, then the superset is fired only if it is the next covered one to be tested in order. It is not appropriate to tag nodes with their level, use a monotonically increasing numbering system, or any equivalent mechanism to prevent the unnecessary breadth-first expansion of a node(s) because the menus are dynamic, and it would be prohibitively costly to renumber, for example, a terabyte of memory. Note that node traversal here is not synonymous with node visitation. Even if parallel processors could render the update operation tractable, the search limit would necessarily be set to the depth of the deepest unmatched node. Here, the likelihood of speedup decreases with scale. The contextual terms should only be *'d if this does not interfere with their expansion—even if normalized. Let the context be given as {C5 C6} and the RHS be {C5 C7}, {C1 ...}. Clearly, if the context had *C5, then the C1 might never be matched.

10. Unlike the case for conventional expert systems, a KASER cannot be used to backtrack consequents (i.e., goal states) to find multiple candidate antecedents (i.e., start states). The problem is that the preimage of a typical goal state cannot be effectively constrained (i.e., other than for the case where the general and specific stochastics are both zero) in as much as the system is qualitatively fuzzy. Our answer is to use fuzzy programming in the forward-chained solution. This best allows the user to enter the constraint knowledge that he/she has into the search. For example, if the antecedent menus are used to specify CAR and FUEL for the context and the consequent is left unconstrained for the moment, then the system will search through all instances, if any, of CAR crossed with all instances of FUEL (i.e., to some limiting depth) to yield a list of fully expanded consequents. Generalization-induced system queries, or consequents that pose questions, if any, will need to be answered to enable this process to proceed. Thus, in view of the large number of contexts that are likely to be generated, all interactive learning mechanisms should be disabled or bypassed whenever fuzzy programming is used. Note that CAR and FUEL are themselves included in the search. Each predicate can also be instantiated as the empty predicate in the case of the antecedent menus, if user-enabled. If the only match occurs for the case of zero conjuncts, then the consequent tree is necessarily empty. A method for fuzzy programming is to simply allow the user to split each conjunct into a set of disjuncts and expand all combinations of these to some fixed depth to obtain a list of contexts. This use of a keyword filter, described below, is optional. For example, the specification $(A \vee A' \vee !A'') \wedge (B \vee B') \wedge (C)$ yields 23 candidate contexts—including the empty predicate (i.e., if one assumes that A'' is primitive and allows for redundancy), which excludes the empty context. The exclamation mark, "!", directs the system to expand the nonterminal that follows it to include (i.e., in addition to itself) all of the next-level instances of its class. For example, !CAR would yield (CAR TOYOTA FORD MAZDA HONDA ... λ). Here, lambda denotes the empty predicate and is included as a user option.

A capability for expanding to two or more levels if possible (e.g., "!!") is deemed to be nonessential but permissible (e.g., for use with relatively few conjuncts). This follows because the combinatorics grow exponentially. One can always take the most successful context(s) produced by a previous trial, expand predicates to another level by using "!"s where desired, and rerun the system. Note that, in this manner, the user can insert knowledge at each stage—allowing for a far more informed, and thus, deeper search than would otherwise be possible. Moreover, the fuzzy specialization engine will stochastically rank the generalized searches to enable an accurate selection among contexts for possible rerun.

The search may be manually terminated by a user interrupt at any time. The search is not to be automatically terminated subsequent to the production of some limit of contexts because to do so would leave a necessarily skewed distribution of contexts—thereby giving the user a false sense of completeness. We would rather have the user enter a manual interrupt and modify the query subsequently. A terminated search means that the user either needs to use a faster computer, or more likely, just narrow down the search space further and resubmit. For example, if we have the antecedent class definitions:

```
(CAR (FORD TOYOTA)) (FUEL (REGULAR_GAS HIGH_TEST
DIESEL)) (AGE (OLD (TIRES ...)) (NEW (TIRES ...)))
```

and the contextual specification:

```
(!CAR) ^ (!FUEL) ^ (NEW),
```

then we would have the following 35 contexts allowing for the empty predicate. Note that the use of the empty predicate is excluded by default, since its use is associated with an increase in the size of the search space and since it may not be used with the consequent menus (see below).

```
CAR
DIESEL
FORD
FUEL
HIGH_TEST
NEW
REGULAR_GAS
TOYOTA
CAR DIESEL
CAR FUEL
CAR HIGH_TEST
CAR NEW
CAR REGULAR_GAS
DIESEL NEW
FORD NEW
FUEL NEW
HIGH_TEST NEW
REGULAR_GAS NEW
TOYOTA FUEL
TOYOTA DIESEL
TOYOTA HIGH_TEST
TOYOTA NEW
TOYOTA REGULAR_GAS
CAR DIESEL NEW
```

```

CAR FUEL NEW
CAR HIGH_TEST NEW
CAR REGULAR_GAS NEW
FORD DIESEL NEW
FORD FUEL NEW
FORD HIGH_TEST NEW
FORD REGULAR_GAS NEW
TOYOTA DIESEL NEW
TOYOTA FUEL NEW
TOYOTA HIGH_TEST NEW
TOYOTA REGULAR_GAS NEW

```

The user may also have used the consequent menus to specify an optional conjunctive list of key phrases, which must be contained in any generated consequent. Those generated consequents, which contain the appropriate keywords or phrases, are presented to the user in rank order—sorted first in order of increasing generalization stochastic and within each level of generalization stochastic in order of increasing specialization stochastic (i.e., best-first). For example, (general, specific) (0, 0) (0, 1) (1, 0) (1, 1) ... Recall that only the specific stochastic preserves validity.

The specified antecedent and consequent classes should be as specific as possible to minimize the search space. Neither the antecedent nor consequent terms specified by the user are ever generalized. For example, if we have the consequent class definitions:

```

(COST_PER_MILE (CHEAP MODERATE EXPENSIVE))
(MPG (LOW MEDIUM HIGH))

```

then we can constrain the space of generated consequents in a manner similar to the way in which we constrained the space of generated antecedents. Thus, for example we can write:

```

(!CAR) ^ (!FUEL) ^ (NEW) -> (!COST_PER_MILE) ^ (!MPG)

```

This is orthogonal programming; that is, reusing previous paradigms unless there is good reason not to reuse them. Each candidate solution has been constrained so that it must contain at least one phrase from the four in the COST_PER_MILE class *and* at least one phrase from the four in the MPG class—including the class name, but excluding the empty predicate of course. IF an asterisk, "*" is placed after the arrow, then the compiler is directed not to filter the produced consequents in any way.

The user can make changes wherever (i.e., to the antecedents, the consequents, or both) and whenever (e.g., interactively) appropriate and rerun the system query. This represents *computing with words* because fuzziness occurs at the qualitative level. It is not really possible for distinct classes to produce syntactically identical phrases because pathnames are captured using unique identifiers. That is, the identifiers are always unique even if the represented syntax is not.

It is not necessary to weight the consequent phrases because instance classes preserve validity (i.e., at least in theory) and because it would be otherwise impossible to ascribe weights to combinations of words or phrases. For example, "greased" and "lightning" might be synonymous with fast, but taken together (i.e., "greased lightning"), an appropriate weight should be considerably greater than the sum of the partial weights. The degree to which the conjunctive weight should be increased does not lend itself to practical determination. Moreover, one is then faced with the indeterminable question (i.e., for ranking) as to which is

the more significant metric: the weight or the two stochastics. Besides, if one follows the dictates of quantum mechanics or veristic computing, it suffices to rank consequent phrases by group as opposed to individually.

Feedback produced, in the form of implausible generalizations, serves to direct the knowledge engineer to modify the involved declarative class structures by regrouping them into new subclasses so as to prevent the formation of the erroneous generalizations. This, too, is how the system learns. The iterative pseudocode for accomplishing the combinatorial expansion follows.

1. Initialize the list of Candidate Contexts to λ .
2. Each conjunct—e.g., $(A \vee A' \vee !A'')$ —in the starting list—e.g., $(A \vee A' \vee !A'') \wedge (B \vee B') \wedge (C)$ will be processed sequentially.
3. Note that $!A''$ means to expand the disjunct to include all members of its immediate subclass, if any. Similarly, $!!A''$ means to expand the disjunct to a depth of two. The provision for multilevel expansion is implementation-dependent and is thus optional. Each expanded conjunct is to be augmented with exactly one λ if and only if the user has enabled the λ -option. This option is disabled by default.
4. Expand the first conjunct while polling for a manual interrupt. Here, the result is
 $(A \vee A' \vee A'' \vee A'' \cdot a \vee A'' \cdot b \vee \lambda)$.
5. Note that the fully expanded list of conjuncts for illustrative purposes appears:
 $(A \vee A' \vee A'' \vee A'' \cdot a \vee A'' \cdot b \vee \lambda) \wedge$
 $(B \vee B' \vee \lambda) \wedge (C \vee \lambda)$
6. Initialize a buffer with the disjuncts in the first conjunct. Here, the first six buffer rows are populated.
7. Copy the contents of the buffer to the top of the list of Candidate Contexts;
8. Current Conjunct = 2;
9. Note that there are three conjuncts in this example.
10. WHILE (Current Conjunct <= Number of Conjuncts) and NOT Interrupt DO
 {
11. Expand the Current Conjunct while polling for a manual interrupt.
12. Let d = the number of disjuncts in the Current Conjunct;
13. Using a second buffer, duplicate the disjuncts already in the first buffer d times. For example, here, the second conjunct has three disjuncts and would thus result in the buffer: $A, A, A, A', A', A', A'', \dots, \lambda, \lambda, \lambda$.
14. FOR each element i in the buffer WHILE NOT Interrupt DO
15. FOR each Disjunct j in the Current Conjunct WHILE NOT Interrupt DO
 {
16. Buffer $[i] =$ Buffer $[i] \parallel$ Current Disjunct $[j]$.
17. (For example, $AB, AB', A \lambda, A'B, A'B', A' \lambda, \dots, \lambda B, \lambda B', \lambda \lambda$.)
- }
18. IF the λ -option has been enabled THEN
 Append the contents of the buffer to the bottom of the list of Candidate Contexts while polling for a manual interrupt.

19. Current Conjunct++
}
20. An interrupt may be safely ignored for the next two steps.
21. IF the λ -option has been enabled THEN
 Final Contexts = Candidate Contexts - λ
22. ELSE
 Final Contexts = contents of the buffer.
23. Duplicate contexts are possible due to the use of λ and possible duplicate entries by the user. Searching to remove duplicate rows is an $O(n^2)$ process. Thus, it should never be mandated, but rather offered as an interruptible user-enabled option.

The iterative pseudocode for constraining the generated consequents follows.

1. Expand each conjunct—e.g., $(A \vee A' \vee !A'')$ —in the starting list—e.g., $(A \vee A' \vee !A'') \wedge (B \vee B') \wedge (C)$. Note that the λ -option is disabled.
2. Here, the result is
 $(A \vee A' \vee A'' \vee A'' \cdot a \vee A'' \cdot b) \wedge (B \vee B') \wedge (C)$.
3. FOR each consequent sequence (i.e., rule) WHILE NOT Interrupt DO
 {
 4. match = FALSE;
 5. FOR each expanded conjunct (i.e., required key concept) WHILE NOT Interrupt DO
 {
 6. FOR each predicate in an expanded conjunct (i.e., PEC) WHILE NOT Interrupt DO
 {
 7. FOR each predicate in a consequent sequence (i.e., PICS) WHILE NOT Interrupt DO
 {
 8. IF PEC = PICS THEN
 {
 match = TRUE;
 BREAK;
 BREAK;
 (Each BREAK transfers control to the next statement outside of the current loop.)
 }
 }
 }
 }
 }
 9. IF NOT match THEN BREAK
 }
10. IF NOT match THEN remove current rule from the candidate list
11. ELSE the rule is saved to the set of candidate rules, which is sorted as previously described.
 }

SUGGESTED NAVAL APPLICATIONS

Figure 2 presents a screen capture of a Type I KASER for diagnosing faults in a jet engine. Observe that the general and specific stochastics are both one. This means, in the case of the general stochastic, that the KASER needed to use a maximum of one level of inductive inference to arrive at the prescribed action. Similarly, the specific stochastic indicates that a maximum of one level of deduction was necessarily employed to arrive at this prescribed action. Contemporary expert systems would not have been able to make a diagnosis and prescribe a course of action, since they need to be explicitly programmed with the necessary details. In other words, the KASER is offering a suggestion here that is open under deductive process. Simply put, it created new and presumably correct knowledge. Here are the two level-0 rules, supplied by the knowledge engineer (i.e., *R15* and *R16*), that were used in conjunction with the declarative object trees to arrive at the new knowledge, *R18*:

- R15*: If Exhaust Flaming and Sound Low-Pitched Then Check Fuel Injector for Carbonization
- R16*: If Exhaust Smokey and Sound High-Pitched Then Check Fuel Pump for Failure
- R17*: If Exhaust Smokey and Sound Low-Pitched Then Check Fuel Pump for Failure

Upon confirmation of *R17*, *R16* and *R17* are unified as follows.

- R18*: If Exhaust Smokey and Sound Not Normal Then Check Fuel Pump for Failure

The KASER finds declarative antecedent knowledge, which informs the system that the three sounds that an engine might make, subject to dynamic modification, are high-pitched, low-pitched, and normal. By generalizing high-pitched sounds one level to SOUNDS (see Figure 3) and then specializing it one level, one arrives at the first-level analogy: low-pitched sounds. This analogy enables the user context to be matched and leads to the creation of new knowledge. Figure 4 depicts the consequent tree and is similar to the antecedent tree shown in Figure 3. The consequent tree is used to generalize rule consequents so as to maximize reusability. Object reuse may simultaneously occur at many levels, even though this example depicts only one level for the sake of clarity. There are many more algorithms, settings, and screens that may be detailed.

Another application is the automatic classification of radar signatures. Basically, the radar data are assigned a feature set in consultation with an expert. Next, a commercial data-mining tool is applied to the resulting very large database to yield a set of rules and associated statistics. These rules are manually fed into the Type I KASER, which interacts with the knowledge engineer to create the antecedent and consequent trees, as well as a fully generalized rule base and miscellaneous sundry. Upon completion of the manual acquisition, the KASER is given a procedure

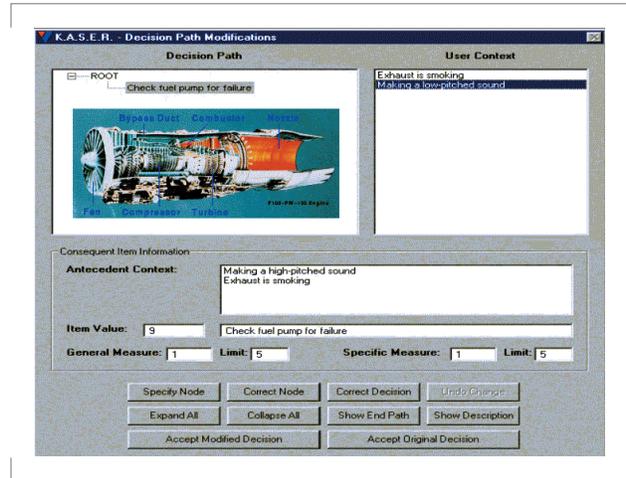


FIGURE 2. Screen capture of an operational Type I kaser.

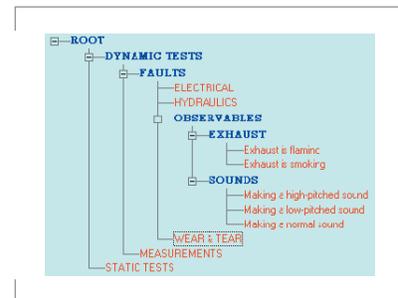


FIGURE 3. Screen capture of an antecedent tree.

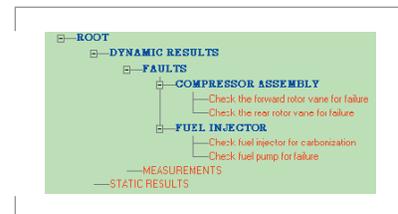


FIGURE 4. Screen capture of a consequent tree.

to link it through open database connectivity (ODBC) to an external electronic intelligence (ELINT) database. This database supplies the radar signatures in approximately real time. The signatures are then automatically classified by the KASER's virtual rule space and the generated stochastics provide an indication of reliability. The KASER, having a virtual rule space >> real rule space can produce erroneous advice if the general stochastic is greater than zero. In this event, the user is requested to supply a corrective consequent(s), which may be "radioed" to the base computer for subsequent update on a daily basis, followed by uploading the more learned KASER. The main benefit here is that the KASER can supply solutions to complex signature-identification problems that would not be cost-effective to supply otherwise (see Figure 1). A Type II KASER should be able to automatically acquire the feature set.

CONCLUSIONS

This project seeks to demonstrate (1) a strong capability for symbolic learning, (2) an accelerating capability to learn, (3) conversational learning (i.e., learning by asking appropriate questions), (4) a metaphorical explanation subsystem, (5) probabilistically ranked alternative courses of action that can be fused to arrive at a consensus that is less sensitive to occasional errors in training, and (6) a capability to enunciate responses. It is argued that the intelligent components of any Command Center of the Future (CCOF) cannot be realized in the absence of a strong capability for symbolic learning.

Randomization theory holds that the human should supply novel knowledge exactly once (i.e., random input), and the machine should extend that knowledge by way of capitalizing on domain symmetries (i.e., expert compilation). In the limit, novel knowledge can only be furnished by chance itself. This means that, in the future, programming will become more creative and less detailed, and thus, the cost per line of code will rapidly decrease. According to Bob Manning [12]: "Processing knowledge is abstract and dynamic. As future knowledge management applications attempt to mimic the human decision-making process, a language is needed that can provide developers with the tools to achieve these goals. LISP enables programmers to provide a level of intelligence to knowledge-management applications, thus enabling ongoing learning and adaptation similar to the actual thought patterns of the human mind."

Moreover, according to Erann Gat at the Jet Propulsion Laboratory, California Institute of Technology, working under a contract with the National Aeronautics and Space Administration [13]: "Prechelt concluded that 'as of JDK 1.2, Java programs are typically much slower than programs written in C or C++. They also consume much more memory.' "

Gat states that "We repeated Prechelt's study by using Franz Inc.'s Allegro Common LISP 4.3 as the implementation language. Our results show that LISP's performance is comparable to or better than C++ in execution speed; it also has significantly lower variability, which translates into reduced project risk. The runtime performance of the LISP programs in the aggregate was substantially better than C and C++ (and vastly better than Java). The mean runtime was 41 seconds versus 165 for C and C++. Furthermore, development time is significantly lower and less variable than either C++ or Java. This last item is particularly significant because it translates directly into reduced risk for software development.

Memory consumption is comparable to Java. LISP thus presents a viable alternative to Java for dynamic applications where performance is important."

In conclusion, the solution to the software bottleneck will be cracking the knowledge-acquisition bottleneck in expert systems (compilers).

ACKNOWLEDGMENTS

I would like to thank Robert Rush, Jr., and James Boerke for their technical programming support in the implementation of the Type I KASER. The Office of Naval Research sponsored this In-house Laboratory Independent Research project.

REFERENCES

1. Chaitin, G. J. 1975. "Randomness and Mathematical Proof," *Scientific American*, vol. 232, no. 5, pp. 47–52.
2. Uspenskii, V. A. 1987. *Gödel's Incompleteness Theorem*, translated from Russian. Ves Mir Publishers, Moscow, Russia.
3. Lin, J.-H. and J. S. Vitter. 1991. "Complexity Results on Learning by Neural Nets," *Machine Learning*, vol. 6, no. 3, pp. 211–230.
4. Rubin, S. H. 1999. "Computing with Words," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 29, no. 4, pp. 518–524.
5. Feigenbaum, E. A. and P. McCorduck. 1983. *The Fifth Generation*, Addison-Wesley Publishing Company, Reading, MA.
6. Minsky, M. 1987. *The Society of Mind*. Simon and Schuster, Inc., New York, NY.
7. Clark, C. T. 2000. "An Interview with Marvin Minsky," *Knowledge Management*, (June) pp. 26–28.
8. Zadeh, L. A. 1999. "From Computing with Numbers to Computing with Words—From Manipulation of Measurements to Manipulation of Perceptions," *IEEE Transactions on Circuits and Systems*, vol. 45, no. 1, pp. 105–119.
9. Rubin, S. H. 1999. "The Role of Computational Intelligence in the New Millennium," Plenary Speech, *Proceedings of the 3rd World Multiconference on Systemics, Cybernetics, and Informatics (SCI '99) and 5th International Conference on Information Systems Analysis and Synthesis (ISAS '99)*, pp. 3–13.
10. Rubin, S. H. 1998. "A Fuzzy Approach Towards Inferential Data Mining," *Computers and Industrial Engineering*, vol. 35, nos. 1-2, pp. 267–270.
11. Hindin, J. 1986. "Intelligent Tools Automate High-Level Language Programming," *Computer Design*, vol. 25, pp. 45–56.
12. Manning, B. 2000. "Smarter Knowledge Management Applications: LISP," *PC AI*, vol. 14, no. 4, pp. 28–31.
13. Gat, E. 2000. "LISP as an Alternative to Java," *Intelligence* (winter), pp. 21–24.



Stuart H. Rubin

Ph.D. in Computer and Information Science, Lehigh University, 1988
Current Research: Intelligent systems; knowledge management.