T2K2: A Type II KASER

Stuart H. Rubin, *Senior Member, IEEE*, Shu-Ching Chen, *Senior Member, IEEE*, and James B. Law

Abstract—**The transformational methodology described in this paper induces new knowledge, which may be open under any deductive process. The method of transposition is used to maintain a maximum size for the application as well as meta-rule bases. The "move to head" method is used by both the application and meta-rule bases for hypotheses formation. Whenever an application rule is fired, it is transposed on the transposition list and also moved to the head on the other list. If any meta-rule on a solution path individually leads to a contradiction on the application rule base, then the offending meta-rule is expunged. Then, when the system is idle enter dream mode, whereby rule i $\Rightarrow$ rule j is generated by the 3-2-1 skewed twister as a candidate most-specific meta-rule. Candidate most-specific meta-rules are "cored" to create one generalization per candidate. These candidate meta-rules are tested for application to each rule in the application domain rule base. In order to be saved in the meta base, they may not map any existing rule in the application domain rule base to one having the same antecedent as another in this base, but a different consequent (as found by hashing). In addition, all candidate meta-rules must map at least one rule in the application base to another distinct one there, or be symmetrically induced from meta-rules that so map.**

Keywords: Computing with Words, Machine Learning, Natural Language, Randomization, Translation

Manuscript received May 02, 2006. This work was supported in part by the Office of Naval Research (ONR) ILIR program.

S. H. Rubin is with SSC San Diego, CA 92152-5001 USA (phone: 619-553-3554; fax: 619-553-1130; e-mail: stuart.rubin@navy.mil).

S. C. Chen is with the School of Computer Science, Florida International University, Miami, FL 33156 USA (e-mail: chens@cs.fiu.edu).

J. B. Law is with SSC San Diego, CA 92152-5001 USA (e-mail: lawjb@spawar.navy.mil).

I. INTRODUCTION

Randomization is defined to mean any operation, which increases the density of information [1], [2], [3]. An expert[2] system is a metaphor for an information-theoretic black hole and performs the following functions.

a. Uses the semantic normalizer to randomize natural language for matching purposes. Application rule consequents should be run through the semantic normalizer prior to saving.
b. The semantic normalizer may be trained on procedurally-invoked truth maintenance operations (e.g., "posting" and "retraction" – especially in a closed domain using *augmented* natural languages) by observing user alterations to the context in response to the firing of some procedural consequents.
c. *Cognition* may result from allowing virtual rules to post and retract knowledge in a closed loop system of system pairs.
d. Provides a dump of valid rules, meta-rules, and virtual rule application sequence(s). Note that possibility metrics cannot be meaningfully ascribed because the system, in keeping with the dictates of the Incompleteness Theorem [4], can (rapidly) discover axioms that cannot be proven (or assigned a 100 percent possibility metric).
e. Offers a metaphorical explanation subsystem based on following a transformational chain.
f. Rules can invoke functional antecedents and procedural consequents.
g. Offers images and possibly video files and sound bytes.
h. Offers a professional speech synthesizer.
i. Associatively recalls word and sentence corrections [5], [6].
j. Can be integrated with a deductive calculus (e.g., the Japanese Fifth Generation) along with neural to sub-symbolic sensory pre-processors.
k. The algorithm is amenable to cluster computing where the application rule base is global and the

meta-rule base and related data structures are local. The parallel version requires $O(\mathrm{r})$ processors for efficient computation.

II. On the Acquisition and Use of Meta-Rule Bases

The functional LHS $\{f_0, f_1, f_2, \ldots\}$ of a rule will define a conjunction of Boolean functions, while the procedural RHS $(p_0, p_1, p_2, \ldots)$ will define a sequence (e.g., a rule base to play chess).

There will be two rule bases – one for user-supplied validated application domain rules and one for meta rules, which is machine generated for mapping application rules and meta rules onto new candidates.

Both rule bases apply the method of transposition to the fired rules so as to allow space occupied by the least-frequently used (LFU) rules to be reclaimed. In keeping with the principle of temporal locality, rules in the application domain and meta-rule bases maintain a logical pointer, which has a rule "move to the head" whenever it is properly fired. This is needed for efficient hypotheses formation. For example, two distinct rules are:

1. $f_0, f_1, f_2, \ldots \rightarrow p_0, p_1, p_2, \ldots$
2. $f_1, f_3 \rightarrow p_1, p_3$

The meta-rule base comprises a set of transformation rules, which are automatically created from the application rule base during *dreaming* [3]. "Coring" is realized using random search. For example, a most specific and an unrelated most general (i.e., "cored" using our statistical mechanical approach) meta-rule, followed by an induced symmetric meta rule are:

a. $f_0, f_1, f_2, \ldots \rightarrow p_0, p_1, p_2, \ldots \Rightarrow f'_0, f'_1, f'_2, \ldots \rightarrow p'_0, p'_1, p'_2, \ldots$(most specific)
b. $f_1 \rightarrow p_1 \Rightarrow f'_1 \rightarrow p'_1$ (a more general meta rule)
c. $f_0, f'_1, f_2, \ldots \rightarrow p_0, p'_1, p_2, \ldots \Rightarrow f'_0, f'_1, f'_2, \ldots \rightarrow p'_0, p'_1, p'_2, \ldots$(apply (b) to (a))

A "cored" rule need map its parent's LHS to its RHS. This is necessary to guarantee that at least one rule in the application domain base will be mapped to another. Such a complete mapping by the candidate meta-rule insures that this requirement will be satisfied in the most computationally efficient manner. The candidate meta-rule may not map any application domain rule to another domain rule already in the base, having a distinct consequent. Note that most specific meta-rules as well as more general meta-rules are to be cored. Coring the more general meta-rules allows for the hill climbing of still more general meta-rules in a manner that would not otherwise be computationally efficient. Successful most-specific and cored meta-rules are created at relatively great computational cost at chance. Thus, it follows that these resultant (pseudo) random seeds need to be extended by way of more economical symmetric induction [7]. Also, the auto application of meta-rules enables the evolution of fixed point transformations in the meta-rule base, where the rule base is held constant [8]. For example, we may find that $\mathrm{H} = \mathrm{m} \bullet \mathrm{r}^2$, where H is the number of induced virtual rules, m is a domain-specific symmetry constant, and r is the number of rules in the application domain rule base. Note that we allow for r application domain rules, meta-rules, and 2r ADT rules (see below). The reason for this is that given a finite amount of memory, when the rule base is young, we favor creativity via meta-rules. Later, when the system is older, we favor stored application knowledge and retain only the most frequently used meta-rules [9]. This is similar to human behavior – when people are young, they are most creative, but know the least (and vice versa). We will use the following meta-rule notation for convenience. $\mathrm{L1} \rightarrow \mathrm{R1} \Rightarrow \mathrm{L2} \rightarrow \mathrm{R2}$. "Cored" meta rules may be formed as follows.

1. If a function is in L1 and not in L2, retain it in L1 and omit it from L2.
2. If a function is in L2 and not in L1, omit it from L1 and retain it in L2.
3. If a set of functions are in both L1 and L2, retain the same subset in L1 and L2. For example, starting with the previous most-specific meta rule, one has $f_0, f_2 \rightarrow p_1 \Rightarrow f_0, f_2 \rightarrow p'_1$. Note that there are 2^{m} subsets of F, where m is the number of functions common to both L1 and L2.
4. Use the Traditional "Coring" Method (TCM) to generalize $\mathrm{R1} \Rightarrow \mathrm{R2}$. For example, starting with a most-specific meta rule: $p_0, p_1, p_2 \Rightarrow p_0, p_1, p_2, p_3$, we find that there are three "cored" rules; namely, $p_2 \Rightarrow p_2, p_3$; $p_1, p_2 \Rightarrow p_1, p_2, p_3$, and $p_0, p_1, p_2 \Rightarrow p_0, p_1, p_2, p_3$. The TCM requires the use of contiguous subsequences (i.e., p_1, p_2 here – not say p_1, p_3) because here the space of candidate meta-rules grows far larger than can be

tractably searched in parallel – large-scale quantum computers not withstanding. Moreover, the TCM yields more robust meta-rules than is possible through the use of non-contiguous subsequences. This is the case because it is better to have a meta-rule that is less frequently applicable with higher validity than vice versa. Also, TCM meta-rules can continue to be generalized through the intermediary action of symmetric induction. Thus, correctness can be approximately preserved through the use of symmetric transformations, while increasing the space of possible transformations. New meta-rules are inserted at the bottom of both pointer lists because they have yet to compete for placement via transposition. Moreover, a pair of meta-rules may have identical antecedents and distinct consequents (i.e., non-determinism) and still map an application domain rule to a distinct pair of potentially valid application domain rules. For example,

1. $f_0, f_2 \rightarrow p_1 \Rightarrow f_2 \rightarrow p_3$
2. $f_0, f_2 \rightarrow p_1 \Rightarrow f_0 \rightarrow p_4$

Now, the upper bound or worst case for the TCM is $\frac{n(n-1)}{2}+n-1$, which reduces to, $\frac{n(n+1)}{2}-1$, which is $O(n^2)$, where n is the min {|R1|, |R2|}. However, the number of predicates in the search space is likely to be far less in practice so that a good estimate of the average case behavior for the TCM is $O(n)$. Combining these results with those for L1 and L2, we find that the worst case behavior for the overall meta rule is $O(2^m \bullet n^2)$, where m is the min {|L1|, |L2|}. Similarly, a good estimate of the average case behavior for the overall meta rule is $O(2^{m/2} \bullet n)$. Note that we used the exponent m/2 because it bears the same relation to m that n does to n^2. The complexity of the overall number of meta-rules is used to approximate the number of candidate meta-rules that can be formed for each visited application domain rule pair. As the rule base gets larger, an ever-greater percentage of meta-rules will be rejected. Then, the remaining candidates will tend to participate in the formation of ever-better hypotheses; albeit at the cost of more cpu cycles.

It follows that on the average there will be about $O(r^2)$ meta-rules. This implies $O(r^3)$ first-order candidate hypotheses (i.e., acting on r application rules), and so on depending upon the domain symmetry constant [3]. This result strongly implies the need for a random "dream" mode to insure a non-skewed (i.e., where move to the head is not a factor) exploration of the search space. It similarly insures that there will be few collisions (i.e., depending on the move to the head skew), as desired.

III. CANDIDATE META-RULE FORMATION

Next, we describe how to form the candidate meta-rules. A pair of distinct application domain rules are selected at random using the Mersenne Twister algorithm. Rules are selected with due regard to their logical position in the "move to head" pointer list. The higher the position of the rule in this list, the greater the likelihood of selection. A good scheme (i.e., the 3-2-1 skew) for achieving this with an application domain base of r rules is to assign the head rule a probability of being selected of $\frac{2r}{r(r+1)}$. The rule just below the head rule has a probability of being selected of $\frac{2(r-1)}{r(r+1)}$. Finally, the tail rule of the base has a probability of being selected of $\frac{2}{r(r+1)}$. Meta-rules created during dreaming are properly inserted at the tail rather than at the head of the move to head list. This follows because if the meta-rules were to be inserted at the head, then over time the fixed point rules would float to the top and result in wasted cpu cycles as a result of duplicate cored and symmetric inductions. This methodology may be indirectly realized using the following algorithm. It is interesting to note that in accordance with Amarel's 1968 Machine Intelligence paper [10], this algorithm (or something equivalently efficient) could not be found by trying to solve the above equations, but only through a change of representation of the given problem. Notice that the method uses an initial skew to accelerate the results obtained during a short "nap", while "long nights" can best be served by a uniform random search for the "smallest" meta-rule. Moreover, since search time grows as the square of the application rule base size and the number of processors is necessarily linear, it follows that the system will not have time to get into pure uniform search with scale. This serves to underscore the importance of transposition in maintaining a quality meta-rule base. The "smaller" the meta-rules in the meta-rule base are, the larger will be the virtual rule space on the average. A highly efficient algorithm for realizing the 3-2-1 skew is given in Fig. 1. Note that this algorithm has the added advantage of favoring just the head of the list during very short *naps*. Also, it is proper to stagnate at a uniform search of all (meta-) rules because skew search time grows as the order square of the size of the base, while the number of parallel processors can only grow linearly. Thus, a point is reached whereupon it is impossible to follow the skew to its conclusion. Should the skew naturally conclude in the small, then we may quit, or run in uniform search mode so as not to waste CPU cycles. Uniform search mode is to be preferred because it looks beyond the immediate past, which has been practically covered.

```
i = 2;
Repeat
  For j = 1 to i
    Repeat
      Select a pair of rules using a uniform
      random number generator (Twister) with
      numbers in [1, i]
    Until
      Wake-Up or the LHS and RHS of the
      pair are distinct;
    The created pair constitutes a most-specific
    meta rule;
    If i < current number of rules in the base, i
    ← i + 1
Until
  Wake-Up;
```

Fig. 1. An efficient algorithm for realizing the 3-2-1 skew. The distribution is skewed in favor of more recent data.

IV. META-RULE VALIDATION

Having created a most-specific candidate meta-rule, the next step is to check it against the rule base to insure that this candidate does not map a rule in the application domain rule base to another rule already in this base, which has a different consequent. This is an $O(r)$ process per meta-rule, which is amenable to parallel processing. If a violation is found, then this most-specific candidate is discarded. At the same time (i.e., using parallel processors), create a candidate "cored" meta rule of the form, $F \rightarrow P \Rightarrow F' \rightarrow P'$ using a 3-2-1 skewed meta distribution of the "move to head" pointer to select the parent meta-rule. A single meta rule "core" is generated at random (in parallel) so as to achieve a more uniform coverage of the search space, which is necessary since the search space can grow to an intractable size (i.e., but even partial explorations here can prove invaluable). The new candidate meta-rule core must be checked for contradiction against the application domain rule base just as were the most-specific meta-rules prior to saving them in the meta-rule base. In addition, cored (not symmetric) candidate meta-rules must map at least one rule in the application base to another distinct one there. Symmetric meta-rules are of relatively high quality in comparison with random cores. In the limit, when they apply they are correct (and more likely to be one to one with scale) and such results cannot practically overwrite the meta-rule base, since this base is gated on minimizing meta-rule size. Thus, symmetric meta-rules are to be accepted without any one to one mapping check. However, symmetric meta-rules need to be checked for contradiction at the time of their creation for the following reason. First, symmetric meta-rules tend to propagate exponentially faster than random, or cored, rules with scale. Thus, erroneous symmetric rules can potentially wipe out the meta-rule base before the next tree search. It follows that since the validity of an arbitrary meta-rule cannot be guaranteed after creation that induced symmetric meta-rules need to be checked for contradiction. In summary, there are at least three fundamental reasons not to do a one to one check on symmetrically-induced meta rules:

a. Symmetric meta-rules inherit the validity of their parents, which itself increases with scale.
b. Partial mappings (transformations) are permitted and desirable so long as they are of high quality (e.g., a predictor-corrector methodology).
c. Occam's razor implies not to introduce new time-consuming code for one to one checking here.

V. ORDERING THE RULE BASES

The application rule base and meta-rule base are maintained using transposition ordering and "move to head" pointers. Unlike new meta-rules, new application rules are inserted at the head using both pointers. Distinct new meta-rules (i.e., hashed for non redundancy check) are inserted at the bottom of the meta-rule "move to head" and transposition lists, while these lists are not full. Ideally, the transposition list will move those meta-rules having the greatest number of successful applications to the top and act so as to preserve them. However, it would add an unwarranted order of magnitude complexity to the rule-verification algorithm (i.e., counting the number of proper one to one maps) to do this. An excellent surrogate metric for the number of successful applications is a meta-rules length, defined by $|f + p| \geq 1$, where a meta-rule can transform just the f_is or just the p_js. That is, we need to preserve the shortest meta-rules because they tend to be the most applicable. Furthermore, minimizing the length of f and/or p serves to maximize reuse, which in turn maximizes the size of the virtual rule space. Moreover, it follows from the corner point, or simplest rule; namely, $f_i \rightarrow p_j$ that the f_is and the p_js should be equally weighted – in accordance with the metric given above. Note that the shorter rules, while more error-prone are subjected to more tests for contradiction by reason of their proportionately increased applicability. Now, if the meta-rule base is filled, search the linked transposition list from the bottom up for a meta-rule, if any, that has the same length as the candidate new meta-rule or greater.

The meta-rules induced from the most-recently fired, or in other words skewed rules are the most important because they best anticipate the current needs. Make the replacement so as not to increase the sum of the lengths of all meta-rules in the base. Reducing the sum of meta-base lengths takes precedence over maintaining the existing transpositional ordering in keeping with the dictates of statistical mechanics. That is, on the average, the larger the meta-rule, the further it will lie from the front of the transposition list. This in turn means that, on the average, the closer a meta-rule is to the front of the transposition list the more likely it is to be preserved. Here, statistical mechanics allows for exceptions on an individual basis, but not when the cases are considered in the aggregate. Notice that "out of order" replacements are self-delimiting and thus, the more they occur, the less likely they will be able to occur in the future. In particular, this approach is necessary where less than $O(r^2)$ space is available for the meta-rule base so as to prevent flushing out previously proven highly applicable meta rules by most-specific, newly cored, or new symmetric ones. The meta-rule base should be of length, $O(r)$ with scale to balance the need for parallel processors with that of the application rule base. Finally, small meta-rules that are not used (i.e., sludge) will fall to the very bottom of the meta-rule base over time. This sludge must be removed because it can accumulate over time and clog the system. This is most efficiently accomplished by expunging the very last and only very last rule (i.e., whatever the length of the meta-rule base happens to be) on the meta-rule base transposition list every time the system enters napping or dream mode. Eventually, short meta-rules will replace the sludge at the very bottom, be fired, transpose with any sludge above, and iterate.

VI. HYPOTHESES FORMATION

The existing semantic normalizer will normalize the context so that it can better cover a rule antecedent. The most-specific, highest transposed rule (i.e., to break ties) in the application domain rule base is fired where applicable. Thus, the user will be offered a rule consequent. For example, if the context were, f_0, f_1, f_2, f_3, then rule (1) would be fired as the most-specific match. Now, suppose that the context were given as, f'_1, f_3. Clearly, no rule in the application domain base will fire on this context. The context is matched against the application domain rule antecedents by hashing subsets of the context in order of non-increasing cardinality. When application rules of the same specificity are randomly selected vs. following move to head pointers, this favors the creation of more numerous (because of increased diversity in the skew) and thus ultimately smaller meta-rules, which tend to increase the size of the virtual rules space. This means that care should be taken so that contextual subsets of equal size are hashed in random order. If and only if no application domain rules can be fired by this method do we resort to virtual rule creation. It is more efficient to generate the virtual rules at runtime than store them because not only is this greatly conservative of space, but it allows for ordered/heuristic search, which is conservative of time. In all cases, the most-specific application rule covered by the context will be first to be fired.

The best way to break cycles is to listen for an interrupt and then supply a new rule at that time, which maps the context to some desired action. Application domain rules may be non monotonic. This means that application rule consequents may add to, modify, and/or delete conjunctive functions in the context in an iterative fashion to enable or disable the firing of other rules. Hypotheses are generated as follows.

1. Visit rules in the application domain rule base as well as in the meta-rule base in order of their "move to head" pointers.
2. Apply each meta-rule on a candidate solution path to all matching application domain rules as guided by heuristic search. Expunge all relevant meta-rules that lead to a contradiction on the application rule base. Searching the meta-rules in order of the move to head list applies the current highest-probability of correctness meta-rules first.
3. Cycles (i.e., in the form of repeated contextual states) are detected and prevented using the cycle detection algorithm of the semantic normalizer. Cycles in the application of meta-rules will be implicitly addressed by the tree-search algorithm. Cyclic application rules are detected using hashing and are expunged.
4. If such application results in an antecedent, which is covered by the context, hash this antecedent to see if its' consequent differs from that stored in the application domain rule base, if present. If a contradiction is found, then if the sequence of meta-rules is of length one, then delete this meta rule and continue on. Similarly, rules in the meta rule base need maintain a logical pointer, which has a meta rule "move to the head" whenever it is part of a sequence of meta rules leading to a correct virtual rule. The same meta-rules are "moved to the bottom" whenever they are part of a sequence of meta-rules leading to a contradicted virtual rule. Here, the fired meta-rules are queued (FIFO) in both cases. Notice that while blame or reward cannot be ascribed to any individual meta-rule in a sequence of meta-rules, the aforementioned movement routines will allow for the maintenance of an ordering that approximates the blame or reward as a consequence of this statistical mechanical approach. This application-induced movement of meta-rules serves to make subsequent dreaming (i.e., skewed meta-rule induction) more relevant to current application needs.

In the absence of an interrupt, attempt to find solutions at virtual levels using hill-climbing in conjunction with backtracking. Virtual rules are ascribed a specificity metric, which is defined by maximizing |{contextual predicates} ∩ {antecedent predicates}|. The issues surrounding the use of this metric are a) transformation rules can be directly or indirectly right recursive for a fixed context and antecedent – implying runaway expansion of the RHS using the ADT; b) the consequent predicates should be randomized to maximize the reuse of any application rules – thus contributing to maximizing the size of the virtual rule space; c) the length of the consequent predicate sequence is a tractable surrogate metric (e.g., statistical mechanics) for randomness; d) the longer the consequent predicate sequence, the more difficult it is to maintain; e) the simplest corner-point rule is of the form $f_i \rightarrow p_j$, where the length of the LHS equals that of the RHS, which suggests that no RHS may exceed the ceiling of twice the average RHS lengths among the valid rules (if not twice, then the RHS might be stuck at a length of one – as well as the need for longer intermediate states in Type 0 transformations) [7]. Notice that $i+1 = 2*(1+2+3+\ldots+i)/i$. There is a heuristic symmetry here with the 3-2-1 skew, which suggests that the two methods could have been co-evolved. Furthermore, the use of an average serves to smooth the variance associated with using just the maximum RHS length in lieu, while allowing for gradual increase in the size of the RHSs. If the induced virtual rule is pruned as a result of the length of its RHS exceeding the allowed maximum, then the transforming sequence of meta rule(s) are moved to the bottom of the move to head list to avert recreation and to encourage replacement or deletion to disrupt cyclic meta-rule groups; and f) randomizing (i.e., minimizing) the sequence of applied meta rules would enforce a breadth-first search – contradicting the use of this most-specific-first heuristic.

Whenever the application rule base acquires or loses a valid rule, it is necessary to efficiently re-compute the average lengths of the RHSs for all the valid rules and use this integer for pruning the heuristic search. (Note that the specificity metric given above is to be preferred to the one defined by minimizing |{antecedent predicates}| - |{antecedent predicates} ∩ {contextual predicates}|, where the selected rule has a minimal metric because a) with scale, meta-rules tend to be valid allowing for deeper search; b) a most-specific rule is to be preferred in any case; c) the specificity metric converges on the minimal metric in view of statistical mechanics; and d) the specificity metric allows for noise, or non monotonic search, which serves to anneal the heuristic – allowing for the possibly more rapid discovery of more specific solutions.) Virtual rule nodes are expanded in the search in order of their non-increasing specificity metrics. In all cases, the most-specific contextual match is to be taken. The state space is maintained at length 2r, where r is defined to be the maximal length of the application rule base and r space will necessarily be filled with valid rules and an additional r space is allocated for virtual rules of which there are $O(r^2)$ [11]. Valid rules are matched such that when equally specific matches of the context are encountered, the last one matched going down the move to head list is selected. This is done because it increases the diversity of the meta-rule space, which leads to a larger virtual rule space.

Conversely, virtual rules are matched such that when equally specific matches of the context are encountered, the first one matched going down the move to head lists is selected. This is done because being first on the move to head lists is associated with increased validity. Note that the use of data-dependent heuristic search at virtual levels not only insures a more uniform coverage of the search space than would be possible exclusively using breadth-first search at all levels (i.e., even when one allows for massively parallel processing); but, the loss in the guarantee of a minimum path length of meta rules is much more than offset by the increased likelihood of heuristically finding a solution path, where one exists.

If a most-specific virtual rule is to be fired (and the user approves), then the virtual rule will have been checked for contradiction before presentation with the result that the transforming sequence of meta rule(s) are moved to the head of the move to head list using a queue (FIFO) structure to preserve the ordering. Similarly, if the most-specific virtual rule is found to be in contradiction with the application rule base (or if the user disallows), then these meta-rule(s) are moved to the bottom of the move to head list using a queue (FIFO) structure. We search to expunge meta-rules (i.e., if any single meta-rule maps an application rule to a contradictory application rule) just prior to their being moved to the head or bottom (where the transformational sequence is more likely to imbue erroneous meta-rules) of the move to head list. Moreover, if any single meta-rule maps a valid rule to a contradictory virtual rule, then it is immediately expunged, but the contradictory virtual rule may continue to undergo transformation as a form of annealing (without poisoning). It follows that we only check virtual rules for contradiction at the first level and prior to firing.

VII. CONCLUSION

The purpose of this paper is to provide the reader with a model for the development of a computational creativity that is based on various facets of the theory of randomization [1], [2], [3]. In particular, the induction of heuristic knowledge, which is open under deduction provides a point of departure from formal logic approaches to the creation of knowledge. It is argued that even the formal logics themselves require heuristic search (e.g., a heuristic back-cutting mechanism for Prolog) for tractable deduction in the large.

ACKNOWLEDGMENT

The author thanks SPAWAR Systems Center, San Diego for their financial support of this research. This work was produced, in part, by a U.S. government employee as part of his official duties and is not subject to copyright. It is approved for public release with an unlimited distribution.

REFERENCES

[1] G.J. Chaitin, "Information-Theoretic Limitations of Formal Systems," *Journal of the ACM*, 21, 403-424, 1974.

[2] G.J. Chaitin, "Randomness and Mathematical Proof," *Sci. Amer.*, vol. 232, no. 5, pp. 47-52, 1975.

[3] S.H. Rubin, S.N.J. Murthy, M.H. Smith, and L. Trajković, "KASER: Knowledge Amplification by Structured Expert Randomization," *IEEE Transactions on Systems, Man, and Cybernetics Part B*, 34 (6), 2317-2329, 2004.

[4] V.A. Uspenskii, *Gödel's Incompleteness Theorem*, Translated from Russian. Moscow: Ves Mir Publishers, 1987.

[5] S.H. Rubin, "Computing with Words," *IEEE Transactions on Systems, Man, and Cybernetics*, 29 (4), 518-524, 1999.

[6] L.A. Zadeh, "From Computing with Numbers to Computing with Words – From Manipulation of Measurements to Manipulation of Perceptions," *IEEE Transactions on Circuits and Systems*, 45 (1), 105-119, 1999.

[7] R.J. Solomonoff, "A Formal Theory of Inductive Inference," *Inf. Control*, 7, 1-22, 224-254, 1964.

[8] S.H. Rubin, "On the Auto-Randomization of Knowledge," *Proceedings of the IEEE International Conference on Information Reuse and Integration*, Las Vegas, NV, 308-313, 2004.

[9] S.H. Rubin, "New Knowledge for Old Using the Crystal Learning Lamp," *Proc. 1993 IEEE Int. Conf. Syst., Man, Cybern.*, pp. 119-124, 1993.

[10] S. Amarel, "On Representations of Problems of Reasoning about Actions," *Machine Intelligence*, 3, 131-171, 1968.

[11] J-H. Lin and J.S. Vitter, "Complexity Results on Learning by Neural Nets," *Mach. Learn.*, vol. 6, no. 3, pp. 211-230, 1991.