Inexpensive Intelligence Using Procedural Propositional Logic

Dr. Richard C. Hicks, Texas A&M International University, USA
Dr. Keith Wright, University of Houston – Downtown, USA

ABSTRACT

Implementations of inference engine systems invoke many costs, including the cost of the inference engine itself, the cost of integrating the inference engine, and the cost of specialized personnel needed to create and maintain the system. These costs make a very high return on investment a criterion for incorporating these systems into the corporate portfolio of applications and technologies. Recently, the No Inference Engine Theory (NIET) [8] has been developed for creating procedural propositional logic rule-based systems. The NIET systems are implemented in traditional procedural languages such as C++ and do not need an inference engine or proprietary languages, thus eliminating the cost of the inference engine, the cost of integrating the system, and the cost for knowledge of a proprietary language. In addition, these procedural systems are an order of magnitude faster [8] than inference systems and maintain linear performance. For problems using propositional logic, the procedural systems described in this paper offer dramatically lower costs, higher performance, and ease of integration. Lowering the external costs and eliminating the need for specialized skills should make NIET systems more profitable and lead to the wider use of propositional logic systems in business.

Keywords: expert systems, propositional logic, verification, procedural inference

INTRODUCTION

The traditional implementation of a rule-based system uses an inference engine, a knowledge base, and a user interface to execute the application. The user interface communicates with the user. The knowledge base contains the rules necessary to solve a specific problem. The inference engine obtains facts from the user or the environment and compares them to the rules in the knowledge base, then uses inference to reach conclusions until the problem is solved [18].

One problem with inference engine implementations is the cost of the inference engine itself. While there are adequate freeware inference engines, many have significant limitations. For example, the freeware version of CLIPS has a limited capacity [8]. The two leading commercial inference engines, Blaze and GenSym, cost $35,000 and one is needed for every computer that executes the application.

A second problem with inference engine implementations is the need for specialized personnel to create and maintain the system [1,4]. The inference engines use proprietary languages that require a significant amount of training. Skill levels among "professional" specialists called Knowledge Engineers (who charge up to $3,000 a day) vary widely, making outsourcing problematic – and expensive. Blaze and Gensym offer development environments for $10,000 a seat.

A third problem with inference engine implementations is integrating the inference engine into the rest of the application. Although inference engine companies continue to improve embedability, there is still much discussion about how to get product X to run in environment Y [4]. This also limits the agility of intelligent applications.

In addition, inference engine applications are computationally inefficient. On problems of a reasonable size, most inference engines struggle to test over 1,000 rules per second [13]. Inference engines also occupy a large amount of memory to store the inference engine itself, the fact set, and the other necessary computational structures [3].

For First Order Logic problems – those with a many pattern/many match requirement – there is currently no superior replacement for the inference engine. However, for Propositional Logic systems without a many pattern/many match requirement, NIET eliminates the inference engine and replace it with procedural code in the company's normal coding environment and languages. Eliminating the inference engine also eliminates the costs associated with it, lowering the cost of implementing intelligence.

In this paper, we will summarize the NIET approach that eliminates the inference engine, lowering costs and making wider usage of intelligent systems practical. For more details on NIET, see [8]. First, we will describe the traditional inference engine and it's specialized functionality, conflict resolution. Next, we will describe how conflict resolution can be performed in the development stage instead of at run-time, transposing the declarative problem into a procedural one. Next, we will compare the performance of the inference engine and it's procedural equivalent. We conclude with a summation of inference engine shortcomings and the NIET solutions to them.

THE TRADITIONAL INFERENCE ENGINE

The inference engine is a software application containing a methodology for solving problems usually stated in Propositional Logic or First Order Logic. It is responsible for determining the sequence of activities that must be performed during the consultation, such as which rule to test, and for reasoning over the rules [18].

The non-trivial component of the sequencing of activities is conflict resolution, which determines the sequence of activities. The sequencing of rules is very important because it affects both accuracy and efficiency. At any time, a large number of rules may potentially fire. Conflict resolution determines the "best" rule to test based on one of many criteria. Conflict resolution may take up to 95% of the time necessary to perform a consultation [3].

As extensive documentation is available, and this conflict resolution strategy is implemented in the majority of inference engines, consider the public domain implementation of CLIPS. The main factor in rule selection is salience, which is a priority value assigned by the developer to each rule. For rules of equal salience, CLIPS provides seven conflict resolution strategies.

In the depth strategy, newly activated rules are placed above all rules of the same salience. In breadth, newly activated rules are placed below all rules of the same salience. Among rules of the same salience, newly activated rules are placed above all activations of rules with equal or higher specificity in the simplicity strategy. The complexity strategy places newly activated rules above all activations of rules with equal or lower specificity among rules of the same salience. The LEX and MEA strategies orders rules of the same salience by the recency of the pattern activations in the rules. The random strategy uses a random number to determine the order among rules with the same salience [5]. The CLIPS manual recommends specificity, which tests the most specific rules first.

If a different salience is assigned to each rule, the conflict resolution strategy is disregarded. If the same salience, or no salience, is assigned to one or more rules, the conflict resolution strategy will determine the rule orderings [5].

Consider the granularity of these conflict resolution strategies. For example, specificity and generality are determined by counting the number of conditions in a rule. In all but the smallest of rule sets, there will be many rules with the same number of conditions. When the conflict resolution strategy cannot order rules, the physical ordering in the rule base is used. This conflict resolution strategy implies that the developer will manually order every rule by encoding a salience, or will manually order the rules in each cluster while considering the run-time conflict resolution strategy. Any errors in the rule orderings coded by the developer may cause inaccuracies or poor performance.

Conflict resolution determines the sequence in which rules fire, and is a major determinant in the speed and accuracy of an expert system [3]. However, the strategies described above, excepting salience, are heuristic in nature. In addition, there is little guidance about the relationships between rule ordering, rule types, and conflict resolution strategies; the three volume CLIPS manual does not address this topic [5].

We feel that this approach to conflict resolution is theoretically weak and incomplete. Later, we will return to this topic, but one significant omission in these conflict resolution strategies is the belief in the accuracy of the rule, often expressed in confidence factors (CNF). In many domains, the most desirable answer is the one in which the developer has the highest confidence.

In order to make the conflict resolution process more theoretically sound, we use verification to classify rules and determine appropriate solution strategies. Toward this goal, we will next determine the appropriate conflict resolution strategy for different classes of rules using verification criteria.

VERIFICATION, CONFLICT RESOLUTION, AND SOLUTION STRATEGIES

Verification answers the question "Did we build the system Right?" – in other words, did we engineer the system correctly. Early work in the field by TEIRESIAS on the MYCIN project [2,15] included a rule proposer. ONCOCIN detected conflict, redundancy, subsumption, and missing rules [17]. Later extensions to verification came feom Nyugen with the CHECK project [10] and Stachowitz with the EVA project [16]. This line of research was followed by the Two-Tier Verification approach that is used in the EZ-Xpert application [6].

Two Tier Verification partitions both the rule base and the known verification criteria to make verification computationally tractable and flexible.

The rule base is partitioned with each partition containing all of the rules that reach a common conclusion. We will refer to these partitions as "rule clusters". Each rule cluster may have it's own solution strategy and verification requirements. The rule base is the summation of the rule clusters.

Verification criteria are partitioned into Local Verification Criteria and Global Verification Criteria. These criteria are shown in Table 1.

Table 1: Two Tier Verification Criteria

Global:	Reachability
	Global domain constraint consistency
Local:	Completeness
	Consistency
	Conciseness
	Domain constraints

The Global Verification Criteria apply to the rule base as a whole and are always enforced. Reachability holds if every condition can obtain a value in some manner, such as from rules or user input. Global domain constraint consistency holds if the data type and legal values for a given variable are consistent between rule clusters, such as that the text values or numerical ranges are consistent.

Local Verification is more complex because not all criteria are applicable to all rules. Domain constraints must be applied to all conditions and actions. Conciseness positively effects performance and maintenance but not accuracy, so it is recommended for all rule clusters.

Completeness and Consistency, however, are mandatory for some rule clusters and not expected to hold for others. This is because there are different types of rules, and they are verified and solved in different manners.

Consistency holds when any combination of condition values reaches the same conclusion for any rules fired by these values [7]. Completeness holds when any combination of condition values will fire at least one rule [12].

In NIET, the four types of rules are classified by their verification criteria; Deterministic, Incomplete, Exception, and Belief-Oriented. Deterministic rules are Complete and Consistent – they always return an unambiguous solution. Incomplete rules are Consistent, but the entire search space is not covered by rules, and some combinations of values will not return any solution. Exception rules, such as those about Penguins and bird flight, are expected to be Incomplete and Inconsistent, and may not return values. Belief-orient rules, where the belief in the rules is less than total, are also expected to be Incomplete and Inconsistent, and may also fail to return a solution [8].

Multi-valued rule clusters return all of the rules in the rule cluster that can be fired, so any solution strategy eventually boils down to an exhaustive approach, meaning that multi-valued clusters do not require conflict resolution. To solve them, first obtain all of the values used in any rule, including subgoals, and then test all the rules.

Single-valued rule clusters, on the other hand, test rules until a single rule fires and then returns, leaving any untested rules without ever testing them, potentially returning a sub-optimal [11] or inaccurate solution. For this reason, conflict resolution and rule ordering are significant for single-valued rule clusters.

The solution strategy for each rule cluster is determined by the rule type assigned to the rule cluster. In each case, the rules are ordered by the appropriate criteria and solved procedurally from top to bottom. This replaces the conflict resolution performed by the inference engine, effectively moving conflict resolution from the run-time environment to the development environment and transposing the problem from a declarative one to a procedural one [8].

NIET uses three criteria to order the rules; cost, complexity, and CNF. For each rule type, we will use the first criteria to order the rules, breaking ties with the second or third criteria if necessary. Cost is the cumulative cost (such as counting the number of necessary conditions or calculating individual conditions computational cost) of getting a value for all of the conditions in the rule. Complexity / Simplicity is, as in CLIPS, simply a count of the number of conditions used in each rule. CNF is the confidence factor assigned by the rule developer in the conclusion of each rule. It should be noted that using three criteria offers a much higher granularity than the one or two criteria used by CLIPS. Cost and CNF are also not considered by CLIPS.

Deterministic rules are Consistent, so ordering does not affect accuracy. Therefore, the goal is efficiency, meaning that ordering is first sequenced by lowest cost, then by rule simplicity, then by highest CNF. These rules are complete, so a conclusion will always be reached.

Incomplete rules are also Consistent, but are not complete. Again, ordering does not affect accuracy, so the first criteria is lowest cost, then rule simplicity, then highest CNF. These rules are incomplete, so either Defaults on Unknowns must be accommodated.

Exception rules by definition are expected to be inconsistent and incomplete. In this case, accuracy is affected by ordering. Accordingly, the first ordering is by rule complexity to pick up the exceptions before the general rule, then by higher CNF, and finally by lowest cost. These rules are incomplete, so either Defaults on Unknowns must be accommodated.

Belief-oriented rules seek to find the rules with highest CNF that fit the current facts, and are expected to be incomplete and inconsistent. As the most desirable rule is the most accurate one by rule developer CNF, order the rules first by highest CNF, then by complexity, then by lowest cost. Again, consider Defaults or Unknowns.

NIET supports two types of solutions for single-valued Belief-oriented rules. It supports the traditional approach of gathering all inputs and testing all of the rules, ordering the results by a CNF calculated from both the user's and rule builders CNFs.

In many domains, the CNFs are all 100, making the user's CNF to be irrelevant. For these rule clusters, NIET supports testing by using only the rule builders CNFs assigned to the rules. This eliminates the need to get all inputs (so the user's CNF is known) and solving all rules so that the calculations are complete, making the solution a true First Rule Satisfied one [8].

Table 2: Ordering by Rule Type

Deterministic	Cost/Simplicity/CNF
Incomplete	Cost/Simplicity/CNF
Exception	Complexity/CNF/Cost
Belief-Oriented	CNF/Complexity/Cost

After the rules themselves have been created and ordered, all that is needed is the appropriate inputs and outputs and the solver.

For multi-valued rule clusters, the solver obtains all the facts, creating subgoals as necessary, and then test all of the rules, and returns all solutions. For single-valued rule clusters, the solver component starts with the first rule in the goal rule cluster. It determines if the facts necessary to test the rule are available. If so, it tests the rule. If facts are unavailable, they are obtained from their source. The sequence of needed conditions drives the sequence of events within the consultation; if the source is another rule cluster, then execution flows to the new rule cluster until it is solved. Inference is top down.

A system created in this manner is functionally and logically equivalent to an inference engine system, but eliminates the cost associated with inference engines and dramatically increases performance.

NIET PERFORMANCE

To test the speed of the research prototype EZ-Xpert, several rule bases were constructed using variations of the Chess End Game Test Set [13]. This test set consists of 648 rules with seven conditions in each rule. Larger rule sets are created by duplicating the original rules and modifying the 648th rule so that it occurs only at the end of the rule base or by adding a eighth condition to be used as a multiplier. Each rule base was tested by supplying the facts necessary to fire the last rule in the rule base on a 2.4 mhz PC [8].

Table 3: Procedural Rule Testing Speed

Size	Speed	Rule per Second
648	.031	20,913
12,960	.515	25,165
19,440	.812	23,940

All of these tests show that over 20,000 rules containing seven conditions each can be tested in one second. It should also be noted that the number of rules tested per second is linear and does not increase when rule base size increases, in contrast to other approaches. In contrast, a shortened version of the same rule base containing 90 rules took .046 seconds, or less than 2,000 rules per second, when tested in CLIPS 6.21 [8]. Larger rule bases could not be loaded into this version of CLIPS, and size has a dramatic impact on the performance of inference engines. This test gives us a preliminary indication that the procedural application will perform 10 times faster than the inference engine application [8].

It should be noted that this is one test of many that should be performed, as it measures only raw search speed, but it is also obvious that the procedural application performs in a linear manner, where this is not true of the inference engines. Consider the performance change in CLIPS using the Manners Test Set. When problem size is tripled, speed drops from 22 rules fired per second to 0.6 rules fired per second.

Table 4: Performance on Large Rule Sets

Guests	Rules Fired	Time	Rules Fired / Second
32	624	28	22
96	4994	7,828	0.63

The procedural approach has far less to do at run-time because conflict resolution has already been performed during development and implemented by rule ordering. As Forgy observed, up to 95% of the time in a consultation is spent performing conflict resolution [3]. For this reason, NIET is faster, requires less memory, and gives linear response regardless of problem size.

SUMMARY

We feel that there are many problems associated with inference engine technology that negatively impact on the wide usage of rule-based systems [4]. Inference engine cost and support are high dollar items. Proprietary languages are used. Skilled specialized knowledge is not widely available. Performance and integration are problematic. Available tools have limited value for creating and maintaining these systems [9]. All of these problems are directly associated with the use of an inference engine.

For systems using only Propositional Logic, the inference engine can be replaced with logically equivalent reasoning and higher performance in procedural languages [8]. Elimination of the inference engine also eliminates its cost and support, proprietary languages, and performance problems. This is accomplished by performing conflict resolution during development instead of at run-time.

Conflict resolution is performed by ordering the rules by their solution strategy, enabling them to be tested sequentially from top to bottom. Ordering differs between different types of rules; Deterministic rules, for example, are ordered differently than Belief-oriented rules because they are solved for different criteria. Use of three criteria to order rules also achieves a more accurate ordering than the one or two criteria used by the traditional approach.

Agility is also enhanced. The overall cost of an intelligent component using this approach is far lower than that of an equivalent project using an inference engine. Existing personnel can create and maintain the systems without new software skills or software because they select the implementation language.

We feel that usage of the NIET approach will lead to much wider use and acceptance of intelligent applications by lowering the cost and commitment necessary to create accurate, high performance intelligent applications.

REFERENCES

1. Bachant, Judith, and Solloway, Elliot. The Engineering of XCON, Communications of the ACM, 32 (3) (1989).
2. Davis, R. “Applications of Meta-Level Knowledge to the Construction, Maintenance, and Use of Large Knowledge Bases, Ph.D. Dissertation, Stanford Univ. 1976.
3. Forgy, Charles L., “RETE: A fast algorithm for the many pattern / many object pattern match problem”, *Artificial Intelligence*, Vol. 19, No. 1, September 1982.
4. Gill, T. Grandon. “Early Expert Systems: Where are They Now?,” *Management Information Systems Quarterly*, Volume19, Number 1, 1995.

5. Giarratano, Joseph C. *CLIPS 6.21 User's Guide,* International Thompson Publishing, Florence KS, , 2003 pp. 43.
6. Hicks, Richard C. "Two-Tier Verification of Rule-Based Expert Systems," *Journal of Computer Information Systems*, Vol. 37, No. 1, Fall, 1996, pp. 1-4.
7. Hicks, R. C. "Knowledge Base Management Systems -Tools For Creating Verified Intelligent Systems" *Knowledge-Based Systems*, April 2003.
8. Hicks, Richard C. "The No Inference Engine Theory", *Decision Support Systems*, Vol. 43, Issue 2, pp. 435-444, March 2007.
9. Murrell, Steven, and Plant, Robert T. "A survey of tools for the validation and verification of knowledge-based systems: 1985-1995" *Decision Support Systems*, Vol. 21, 1997, pp. 307-323.
10. Nyugen, T.A., Perkins, W.A., Laffey, T.J. and Pedora, D., "Knowledge Base Verification", *Artificial Intelligence*, Vol. 8, No. 2, 1987, pp. 69-75.
11. O'Leary, Daniel E., "Inference engine greediness: subsumption and suboptimality," *Decision Support Systems*, Vol. 21, 1997, pp. 263-269.
12. Preece, A. D. "A new approach to detecting missing knowledge in expert system rule bases", *International Journal of Man-Machine Studies*, Vol. 38, 1993, pp. 661-688.
13. "Benchmarking CLIPS/R2", <http://pst.com/benchcr2.htm>, Production Systems Technology Inc, June 2008.
14. Quinlan, J. R. "Simplifying Decision Trees", *Knowledge Acquisition for Knowledge-Based Systems*, Gaines, B., and Boose, J., editors, Academic Press, 1988.
15. Shortliffe, E. H. *Computer-Based Medical Consultations: MYCIN*, Elsivier Publishing, New York, NY, 1976.
16. Stachowitz, Rolf. "Validation of Knowledge-Based Systems", Tutorial at the Hawaii International Conference on System Sciences, Kona, Hawaii, Jan. 1990.
17. Suwa, M., Scott, A. C., and Shortliffe, E. H., "An Approach to Verifying Completeness and Consistency in a Rule-Based Expert System," *A I Magazine*, Vol. 3, No. 4, 1982, pp. 16-21.
18. Winston, P. *Artificial Intelligence*, Addison-Wesley, Reading, MA 1993.

NOTES