1989-02

Inference-security analysis using resolution theorem-provi

Rowe, Neil C.

Monterey, California. Naval Postgraduate School

Inference-security analysis using resolution theorem-proving

Neil C. Rowe

Department of Computer Science

Code 52Rp Naval Postgraduate School Monterey, CA 93943

Abstract

Indirect logical inferences can provide a significant security threat to information processing systems, but they have not been much studied. Classification of data can reduce the threat, but classification decisions are typically left to the intuitive judgment of experts. Progress has been made on analyzing indirect statistical inferences that may compromise security of a database system ([3], chapter 6). We describe and implement a nonnumeric analog of these methods for proving security. Our approach involves analyzing facts and inference rules assumed to be known to a compromiser, deriving all their possible consequences using resolution theorem-proving, a technique which we argue is far more appropriate to this problem than rule-based expert systems or information flow analysis. An important contribution of our work is augmentation of resolution to handle associated time intervals and probabilities of statements being true. Our augmentation is simple to use by domain experts untrained in computers, and we believe it will provide the first truly practical tool for analysis of indirect logical inferences in information systems. We demonstrate capabilities with an example from military security.

This paper appeared in the Fifth International Conference on Data Engineering, Los Angeles, CA, February 1989, 410-416.

1. Introduction

Secrets can be discovered in many indirect as well as direct ways. Intuitive understanding of such inference methods is required to make decisions about classification of data, but it has rarely been formalized. We suggest that it can be formalized, and that techniques from artificial intelligence can be used to automate it. Some research along these lines has been done regarding statistical data obtained from statistical databases ([3], chapter 6), and a little work on combinatorial inferences [2], but the more general problem of logical inferences from logical data and logical rules has been virtually ignored. [8] is the only major research on this with which we are familiar, and it is high-level: it emphasizes knowledge representation, not implementation issues as we address here.

Consider an example. Suppose on a Navy ship our sensors say that a nearby ship (but not near enough to be visible) has abruptly changed course. Ships are expensive machines to operate, and course changes require good reasons. One reason would be navigation obstacles in front of the other ship, such as land masses or reefs. If we are in the open ocean, we can rule out that possibility. If the other ship is nonmilitary there are few other reasons for a course change except a weather report of a dangerous storm ahead, something we could check out ourselves. With these possibilities ruled out, the ship must be military. Perhaps it has been ordered to proceed somewhere new. This would require communications activity to and from the ship shortly before the course change, something we may be able to confirm. Or the other ship may be engaged in maneuvers or sensor placement, which is suggested if it made many similar abrupt course changes recently. Or it may have just noticed our ship and may be trying to avoid it, which is suggested if we noticed radar activity shortly before the course change. Additional information about the ships in the area and the current international climate can be used to further confirm and disconfirm these hypothesis.

So observation of a ship's activities may let us to infer the secret purpose of those activities, using common-sense inference methods. This indirect inference is increasingly worrisome a problem, because increasing computerization of all aspects of human decision-making, as with recent interest in expert systems [4], tends to make decision-making more predictable [8]. For instance, "captain's assistant" expert systems have been designed to make recommendations to ship captains, based on general principles, for specific situations like navigation and emergency management [5,6]. Increasing predictability means increasing danger of security problems.

One solution proposed has been to build another kind of expert system, a "security-analysis" rule-based system [12,1,11]. Human experts could be prompted to give their rules for deciding when something should be classified given that certain other things are classified. Then the rules can applied to specific situations to see what can be inferred. One advantage of rule-based systems is that there is now a wide variety of commercial software supporting their construction, for a wide range of computers. However, rule-based systems have a big disadvantage when security analysis is concerned: they are not guaranteed to find all possible inferences from a set of facts and a set of rules. This is because most rule-based systems reach only positive single-fact conclusions, and cannot directly conclude the falsity of facts, disjunctions of facts, or certain forms of quantification on facts. As one example, if A implies B, and B implies C, neither forward nor backward chaining can conclude that A implies C. As another example, if a rule says that A is true if B is true, then neither chaining method can conclude when A is false then B is false too. Furthermore, rules usually require a certain direction of reasoning (from "if" part to "then" part in forward chaining, and vice versa in backward chaining), and inferences in other directions are not possible with most systems. These are serious disadvantages when one wants to prove the security of something, when valid conclusions unreachable by the control algorithm compromise security.

Another approach to analysis of security from logical inferences is to use information flow analysis as in ([3], chapter 5). This can reason in many different directions, and thus satisfies our first objection to rule-based systems. But it fails to meet our second objection, since inferences are just numbers representing the information-theoretic value of statements about the world. The reduction of everything to numbers means that sophisticated logical inferences are not possible, like those involving disjunctions or existential quantification.

We argue that the best technique for computerized security analysis of logical inferencing is resolution theorem-proving. It answers both objections to rule-based systems, since it can be proved to eventually derive every conclusion the follows from a given set of facts, rules, conditions, policies, and truths. Its basic principle is quite simple, and many efficiency improvements have been devised for it in the more than twenty years since its discovery [7]. Its only disadvantage is that because it is thorough, it is slower than the forward-chaining and backward-chaining engines used in rule-based systems.

2. Our approach to using resolution

Our proposed methodology represents a modification of conventional knowledge acquisition methodology for expert systems. This should be performed hand-in-hand with domain experts, which for instance for inferences about ships would be naval intelligence experts.

1. Identify the kinds of evidence that can be available as input (e.g. the observed heading of a ship), plus unobservable facts that may also affect decisions in the domain of study (e.g. the content of a coded message sent to a ship).

2. Identify all useful intermediate concepts, clusters of concepts that recur frequently and simplify domain discussion. For instance for ship movements, the condition "obstacles_in_front" which is true if there is a land mass, a reef, or another ship straight ahead in the direction a ship is traveling.

3. Identify all final conclusions. Typically these represent the performance of actions; for instance for ships, the final conclusions are the ship movements.

4. Identify pairs of mutually exclusive evidence and concepts, as well as sets that together fully cover all possibilities ("closure sets"). The former can be expressed in the form "A implies not B", and the latter in the form "A or B or C or ...". For instance, a ship must either turn or maintain course, and those two possibilities are mutually exclusive.

5. If actions are prioritized, formulate the priorities as logical implications. For instance, if the first priority is to follow orders, the second priority to avoid bad weather ahead, then maintaining course when there is bad weather ahead implies orders to do so.

6. Identify all strong causal implications applying to the concepts, including statements of policy or doctrine. These should be expressed as logical implications from the premises to the conclusions. For instance, if a ship receives orders to turn, or there is a navigational obstacle ahead, the ship should turn.

7. Identify all other reasonably strong inference rules that domain experts use to relate the evidence and concepts of step (1). Unlike most expert systems, however, the rules can have negative conclusions, disjunctive ("or"d) conclusions, and can use quantifiers freely, as resolution is a fully general inference method for first-order predicate calculus.

8. Translate the material from steps (1)-(7) into clause form, a set of disjunctions of literals, a process quite straightforward [14]. Store these clauses in an efficiently encoded form.

9. Now formulate your security experiments. Formulate a variety of situation descriptions, including both positive and negative facts, negative being those known to be *not* true in a situation.

10. For each case in turn, perform resolution theorem-proving to find all resolvents of the facts and clauses; these are conclusions that logically follow from them. Check to see if any violate security. (The basics of resolution are presented in many places (e.g. [9], chapter 14), so we will skip them here.) Formulate countermeasures (see section 4) for security-violating inferences.

Consider an example of finding a cause for a sudden turn of a ship. Suppose we also observed communications to the ship, but none from it; suppose a coast is nearby; suppose there is no report of bad weather ahead and there are no other ships in the vicinity; suppose the ship has not engaged in zigzagging; and suppose the ship is not military. Some policy rules for this situation are:

--Avoidance requires a course change.
--An order for a course change requires a course change.
--Obstacles in front require avoidance.
--Yielding the right of way requires avoidance.
--Communications to the ship that report bad weather ahead require avoidance.
--If a military ship engages in radar activity and recognizes another ship
in the vicinity, this requires avoidance.
--Maneuvers are a kind of ordered course change.
--Sensor placing is an ordered course change.
--Obstacles are in front if there is a grounding danger and coast is nearby.
--Obstacles are in front if there is a grounding danger and there are reefs nearby.
--If another ship has the right of way, you are required to yield right of way.
--A military ship engaged in sensor placing must do zigzagging.

We can also specify some closed-world and uniqueness conditions:

--An observed course change implies either avoidance or an ordered course change.
--Avoidance implies either obstacles in front, yielding the right of way,
bad weather ahead, or radar activity.
--An ordered course change implies other maneuvers or sensor placing.
--An ordered course change implies a military ship.
--Radar activity requires a military ship.
--Either a course change action or a maintain course action is always done.
--Changing course and maintaining the course cannot be done simultaneously.
--Avoidance and an ordered course change cannot be implemented simultaneously.
--Yielding the right of way requires another ship in the vicinity.
--Yielding the right of way requires that the other ship has the right of way.
--An ordered course change requires communications to the ship.
--An ordered course change requires communications from the ship.

Resolution in general can handle clauses with variables. However, few security-application clauses require variables; none of the above example statements require them. Time and probability information associated with clauses can be conveniently handled without variables, as we show in section 3; most reasoning with existential quantifiers gives very weak inferences; and universal quantifiers can usually be made implicit. Thus we chose a variable-free first implementation.

Resolution is a process that takes two clauses as inputs and returns a new clause as output. It is required that the input clauses have a pair of opposites--that is, a term which occurs unnegated in one clause and negated in the other clause. Then the result of the resolution (the "resolvent") is the new clause formed by appending the old clauses together and deleting the pair of opposites plus any duplicates.

Resolution is commutative and associative, so for n clauses without variables there is an upper bound of 2 to the nth power of distinct resolutions among them. Thus the worst-case time complexity of resolution is exponential in the input size. However, many heuristics for making resolution more efficient are known [14,7,13]. Efficiency is not a major concern of ours because no other approach to the same security problems can claim completeness of reasoning.

3. Adding probability and time reasoning

Critically important aspects of security applications are the modeling of time and not-completely-certain information. However, such ideas are rarely supported in programs for resolution by computer, and the methods for them in other areas of artificial intelligence are usually quite complicated. So to make security analysis by resolution possible, we had to develop our own simple yet powerful methods for these important phenomena.

3.1. Probabilities

Logical expressions cannot be assumed either completely true or completely false when evidence can be mistaken or rules are heuristic rather than guaranteed. Provided the uncertainties are small, we can extend our methodology to handle them by associating a probability with each clause, computed when the clause is entered in the computer. This probability will represent the clause's certainty, or the expected fraction of the time that it will be true. (Many other methods for modeling uncertainty are being examined in artificial-intelligence research, most notably Dempster-Shafer theory and fuzzy logic, but they can get complicated.) In resolution, we can say that our certainty in the result is our certainty in the conjunct of the certainties of both resolvents. We call this an "and-combination" issue for probabilities in [9] chapter 8, and we can use any of the methods proposed there for it.

It is often reasonable to use the independence assumption (for which the resultant probability is the product of the input probabilities), even when independence cannot be strictly proved to hold. We thus recommend it here. But before applying it, we must take into account duplicate reasoning in obtaining the two clauses resolved, clauses which were used to prove both resolvents. To catch this, we can number all the starting clauses, and store with each new clause the set of the numbers of the starting clauses used to derive it. (A set suffices since resolution is commutative and associative.) Then when doing probability combination, all such duplicates should be counted only once, since two occurrences of the same thing are certainly not probabilistically independent.

We must eliminate clauses whose terms are supersets of other clauses, because these can slow reasoning down and unfairly influence probabilities. Such eliminable clauses can be either newly-derived or old ones made obsolete by new simpler conclusions. And since the approximate values derived by this evidence-combination approach only work well when clauses are almost certain, a probability lower bound should be set so that any derived clauses less certain than it will be discarded. This prevents highly questionable inferences and improves reasoning efficiency.

Multiple evidence for a clause, when it can be proved by more than one chain of reasoning, should make us even more certain of it. This is the classic problem of "or-combination" of probabilities, and its formula can be derived from the "and-combination" formula by the relationship p(A or B) = p(A) + p(B) - p(A and B). With the independence assumption for and-combination, this becomes p(A) + p(B) - p(A)p(B). For or-combination to be valid, one of the two proofs of the same clause must depend on least one initial clause the other does not, so that one proof is not just a permutation of the resolutions done in the other proof. Also with the independence assumption, starting clauses used in making both proofs should be removed from the above calculation and later multiplied into the result (see section 3.3 for an example), since p((A and B) or (A and C)) = p(A and (B or C)) = p(A)p(B or C). In general when or-combination occurs, an arbitarily embedded

Boolean expression without negations is needed to store the derivational history of the resulting clause, to enable proper elimination of such duplicates. For instance, if clauses 1 and 2 were resolved, then or-combined with the resolvent of clauses 1 and 3, and the result or-combined with the resolvent of the resolvent of clauses 4 and 5 with the resolvent of clauses 1 and 6, the result could be expressed as "1 and (2 or 3 or (4 and 5 and 6))".

3.2. Time

Besides by probabilities, observations and conclusions can be qualified by their times. This means further overhead in reasoning, but it further constrains conclusions. We can give two times for every clause, forming a "validity interval": the start and end of the period for which the clause is valid. For observations, these times will be the observed starting and ending times of the observed condition. For instance, a weather report can have a validity interval of a few hours; an observation of a ship's bearing can have a validity interval of about 10 seconds. For conclusions, these times represent the period for which the conclusion will hold.

Note that some events establish conditions that hold indefinitely, and thus it is correct to make their effective validity intervals much longer than the events themselves. For instance, communication to the ship is required to receive an order from headquarters, but the knowledge of that order lasts indefinitely after communications cease, until superseded by another order. Negative events (things observed not to occur), can be given time periods too, periods in which they did not happen. This may pose problems when we want near-real-time inferences, since we cannot be sure about the occurrence of events after the current time, so our conclusions may be incomplete if chains of causation started before the current time are not yet completed.

When we resolve two clauses, we can only be sure of the conclusion clause in the overlap of the two time intervals; so the resolvent validity interval is their intersection. If there is no overlap, the two clauses cannot be resolved.

Associating time with events permits us to reason about occurrences of the same observation at different times. For instance, communication activity of a military ship may occur several times during a complicated maneuver, for orders and reports about different purposes. For efficiency, we can often cluster clauses so that the times in each cluster do not overlap with clause times in any other cluster. Then we need only look for resolutions within each cluster.

Checks for redundant clauses must be done differently when clauses have times, because if the terms of clause A are all terms of clause B, B can only be eliminated when its time interval is included in A's. Otherwise, the overlap of the time intervals can be removed from the interval for B, narrowing it. Another redundancy is when the same clause is proved on two overlapping or adjoining intervals, in which case the two occurrences can be merged to span the union of the time intervals. Then we can make a different sort of probability independence assumption, independence of the probabilities with respect to time, and take the probability for the resulting interval as the the range-weighted average of the two merge-interval probabilities.

3.3. Simplification examples

Here are some examples of our redundancy elimination between derived clauses:

1. (a or b from 10 to 20 with probability 0.8) with (a or b or c from 12 to 14 with probability 0.9): eliminate second clause

2. (a or b from 10 to 20 with probability 0.8) with (a or b or c from 12 to 25 with probability 0.9): restrict second clause to be from 21 to 25

3. (a or b from 12 to 14 with probability 0.8) with (a or b or c from 10 to 20 with probability 0.9): partition the second clause into two, running 10 to 11 and 15 to 20 respectively

4. (a or b from 10 to 20 with probability 0.6) with (a or b from 10 to 20 with probability 0.9), assuming all the starting clauses needed to derive the two were different except for one common clause of probability 0.9: coalesce into one clause with probability 0.9 * [(0.6/0.9) + (0.9/0.9) - (0.6/0.9)*(0.9/0.9)] = 0.9.

5. (a or b from 10 to 20 with probability 0.8) with (a or b from 21 to 25 with probability 0.9): coalesce into one clause with range from 10 to 25, and change its probability to the range-weighted average of the probabilities, (10*0.8 + 4*0.9) / (10+4) = 0.82

3.4. A demonstration

As an example, take the same clauses of our previous example and add some probabilities and times (the latter assumed to be only integers):

--Assume that the probabilities were 0.9 that we saw communications to the ship and did not see communications from the ship, 0.95 that the ship was nonmilitary, 0.8 that the ship was not zigzagging, and 0.95 that there was no other ship in the vicinity.
--Assume that the sixth, seventh, and eighth rules have probabilities respectively of 0.7, 0.8, and 0.8.
--Assume times can be represented as integers, and assume we reason only between times -10000 and 10000.
--Assume the course change was observed from time 78 to 79, and the ship maintained course at all other times; there were communications to the ship at time 50; the coast was nearby from time -600 to 1200; there was no bad weather ahead from time -1000 to 1000; there was no zigzagging from time -800 to 800; there was no other ship in the vicinity from time -500 to 500; there was a ship from time 500 to 10000.

Then our C-Prolog program in the Appendix concludes the following facts by resolution, in order. (We did not implement probability thresholds or multiple-evidence or-combination in because they do not help in our ship-inference test case.)

not maintain_course with probability 1 from time 78 to 79 (1 resolution)
not course_change with probability 1 from time -10000 to 77 (1 resolution)
not avoidance with probability 1 from time -10000 to 77 (11 resolutions)
not obstacles_in_front with probability 1 from time -10000 to 77 (3 resolutions)
not course_change with probability 1 from time 80 to 10000 (1 resolution)
not avoidance with probability 1 from time 80 to 10000 (2 resolutions)
not obstacles_in_front with probability 1 from time 80 to 10000 (3 resolutions)
not bad_weather_ahead with probability 0.918 from time -1000 to 10000 (5 resolutions)

```
not grounding_danger with probability 1 from time -600 to 77 (11 resolutions)
not grounding_danger with probability 1 from time 80 to 1200 (5 resolutions)
not ordered(course_change) with probability 0.95 from time -10000 to 10000 (1 resolution)
avoidance with probability 0.95 from time 78 to 79 (3 resolutions)
not maneuvers with probability 0.760 from time -10000 to 10000 (2 resolutions)
not sensor_placing with probability 0.760 from time -10000 to 10000 (4 resolutions)
not radar_activity with probability 0.95 from time -10000 to 10000 (1 resolution)
not yield_right_of_way with probability 0.997 from time -10000 to 10000 (9 resolutions)
obstacles_in_front with probability 0.902 from time 78 to 79 (8 resolutions)
not other_ship_has_right_of_way with probability 1 from time 501 to 10000 (5 resolutions)
```

The "obstacles_in_front" conclusion is the most interesting. It is what we were looking for, the inferred purpose of the course change. In general, the number-of-resolutions-used output identifies the most useful conclusions when many are generated by our program, especially among the unnegated conclusions.

The program also derives the following multi-term clauses which are the only ones remaining at the end of the run after elimination of redundancies:

```
not other_ship OR not other_ship_has_right_of_way with probability 1 from time -10000 to -501
(4 resolutions)
not bad_weather_ahead OR not communications(to_ship) with probability 1 from time -10000 to
-1001 (3 resolutions)
not grounding_danger OR not coast_nearby with probability 1 from time -10000 to -601 (4
resolutions)
not grounding_danger OR not reefs_nearby with probability 1 from time -10000 to -601 (4
resolutions)
not grounding_danger OR not coast_nearby with probability 1 from time 1201 to 10000 (4
resolutions)
not grounding_danger OR not reefs_nearby with probability 1 from time 1201 to 10000 (4
resolutions)
```

This run took 574 CPU seconds with a C-Prolog interpreter on an unloaded Sun workstation. The program took 75K of memory, and the dynamic memory required was an additional 127K, 50K of which was heap storage. During this run, 184 clauses were created, considerably less than the upper bound of 2 to the 21st power = 253952 clauses. (We did not pick the example to get this effect, but feel the example's character is representative of real security problems.) Of those 184, some had identical clause terms but different time intervals, and arose from redundancy elimination, so the number of resolutions performed was less than that. Bear in mind that those 184 represent a *complete* analysis of every possible conclusion that can be derived from the input.

4. Countermeasures

Protection methods used for statistical-database inferences have analogs for the logical inferences we have been considering.

4.1. Security analysis by experiment

A good way to understand this security threat is by experiments attempting to infer specific things from specific sets of statements, because the ease of compromise varies enormously with the input. Furthermore, experiments can examine in detail the secrecy of very particular things; we can try to prove those things from a set of common-sense clauses, and if we cannot after all possible resolutions, they are safe. So controlled experimentation seems the ideal way to monitor security threats of this nature, provided the number of facts to be protected is small enough that testing can be thorough.

If we do not know what observations may be made by a person wishing to violate security, we can assign a priori probabilities to particular observations. We then can conduct repeated experiments, generating test observations according to these probabilities. The fraction of the time that an important secret is revealed approximates the danger.

4.2. Protection by information suppression

In the statistical database work, two main approaches to protection have been examined: suppressing observations and adding noise to observations.

Suppression of observations is a powerful protection technique when you can do it, but its difficulty varies enormously. For instance, a ship can disguise its communication with headquarters by using a different type of transmitter that is not so easily detectable, but weather patterns that affect routing decisions cannot be hidden. Rather than suppressing the suppressable observations at random, we should preanalyze particular suppressions by resolution experiments. To find the implications of some particular observation (or its absence), we can do the classical technique of set-of-support resolution. If we are especially interested in whether one particular secret can be derived from an observation, we can assume its negation, and try to prove the null clause, again using set-of-support.

Often a quick probability calculation for each clause can suggest where security problems lie. Given a priori probabilities of each clause term being false, the probability can be computed that all but one term in a clause will cancel out using only facts, using the standard evidence-combination techniques of artificial intelligence ([9], chapter 8). For instance, suppose the clause is "A or B or C", A has probability 0.8, B has probability 0.3, and C has probability 0.4, and events A, B, and C are roughly probabilistically independent of one another. Then A will remain after cancellation of B and C with probability 0.42; B will remain after cancellation of A and C with probability 0.12; and C will remain after cancellation of B and C with probability 0.14. For a very thorough security analysis, we could also precompute probabilities of clauses resulting from more complicated resolutions too.

If part or all of the observations are from computer databases, then the computer will have a record of exactly what was requested by a potential compromiser; this "database compromise" situation is considered in more detail in [8], and is easier to handle than the general inference problem. Suspicious behavior (like asking for many complex logical operations on data, analogous to the "trackers" that can compromise statistical databases) can initiate more detailed analysis of what is or might be inferable, giving recommendations for specific suppressions.

If a computer system can make inferences itself, as an expert system, then a compromiser could exploit this assistance to efficiently compromise it. One solution to this is to conceal sensitive inference rules and clauses [1]. But this is not much help when the rules can be inferred themselves from general principles or from one's own human experts.

4.3. Protection by adding noise

Effective steps to prevent key observations may be costly or impossible. Fortunately another class of countermeasures exist, methods which protect information by adding confusion to the analysis of it. After we have used the methods of the previous section to identify particularly dangerous observations, we can enable bogus observations of these kinds. For instance, if communications strongly indicate a course change, then frequent unnecessary communications or communications only at exactly regular intervals will conceal true intentions. To camouflage a major policy decision, other postponable or irrelevant actions can be taken simultaneously. For instance, to conceal a major course change of a ship, it can simultaneously cut its speed considerably, make the turn very slowly, and initiate electronic jamming.

If observation time is an important clue, then a useful confusion technique may be to insert delays of random amounts before taking actions. This complicates the matching problem discussed in section 3.2 regarding the same observation at different times, and makes it harder to tell when a set of observations is complete. For instance if a ship is observed to turn a long time after receiving communications, it is less clear that those communications were the cause.

A third idea is to formulate new policies or rules designed to confuse observers. This is a good approach to very serious security problems. For instance, suppose we have a rule that a very-secret event A logically implies event B. If this is the only rule that implies event B, a closed-world rule would say that B must imply A. To fix this, we can change real-world policies to have event C also imply event B.

The main drawback of noise-addition protection methods is the required policy changes. A large bureaucracy like the military may find it hard to consciously change to less efficient ways of doing things (though unconsciously is easy) in pursuit of the often nebulous goal of security.

5. Conclusion

We have presented perhaps the first concrete tool to examine a little-studied security threat to both computerized and noncomputerized information systems, inference from the results from a set of deterministic logical policies or rules. Resolution theorem-proving, unlike rule-based expert systems, provides a simple way to exhaustively explore the threat, permitting evaluation of countermeasures. We have solved a major obstacle to the effective use of resolution by finding a simple way to handle evidence and rule uncertainty as well as time-based reasoning. Standard information-security countermeasures of suppressing and adding noise to information can be used, but their suitability varies significantly with the particular application, and they can require significant overhead.

Much work remains to be done on this subject. Efficiency improvements to resolution algorithms continue to be developed in the spirit of [13] and [14], and these can be incorporated into our program. Work with predicates having variable arguments needs to be done. Alternative formulations of uncertainty need to be studied. Better countermeasures need study. Perhaps the key remaining conceptual issue is the development of intelligent control strategies for this security analysis, perhaps via artificial-intelligence "meta-rules" also obtained from application-domain experts, to control the selection of clauses for faster inference of more useful conclusions and quicker elimination of irrelevant conclusions.

6. Acknowledgements

This work was supported in part by the Foundation Research Program of the Naval Postgraduate School with funds provided by the Chief of Naval Research, in part by the U. S. Army Combat Developments Experimentation Center under MIPR ATEC 88-86, and in part by Ford Aerospace. Thanks to E. Richard Hilton, Roger Marshall, and Matthew Morgenstern for discussions.

7. References

[1] T. Berson and T. Lunt, "Security Considerations for Knowledge-Based Systems", Third Annual Expert Systems in Government Conference, Washington, D.C., October 1987, 266-271.

[2] R. Demillo and D. Dobkin, "Combinatorial inference", in *Foundations of Secure Computation,* Demillo, Dobkin, Jones, Lipton (eds.), Academic Press, New York, 1982, 27-35.

[3] D. E. R. Denning, *Cryptography and Data Security,* Reading, Mass., Addison-Wesley, 1982.

[4] E.A. Feigenbaum and P. McCorduck, *The Fifth Generation: Artificial Intelligence and Japan's Challenge to the World*, Reading, Mass., Addison-Wesley, 1983.

[5] B. Gogel, "An Expert System to Provide Guidance on the Operation of Installed Damage Control Systems Aboard Naval Ships in Emergency Situations", M.S. thesis, U.S. Naval Postgraduate School, December 1986.

[6] T. Lewallen, "A Collision-Prevention Expert System for a Navy Officer of the Deck", M.S. thesis, U.S. Naval Postgraduate School, March 1987.

[7] D. W. Loveland, "Automated theorem-proving: a quarter century review", in *Contemporary Mathematics, Volume 29,* American Mathematical Society, Providence, Rhode Island, 1983, 1-45.

[8] M. Morgenstern, "Security and Inference in Multi-Level Database and Knowledge-Base Systems", Proceedings of the ACM-SIGMOD Annual Conference, San Francisco, June 1987, 357-373.

[9] N. C. Rowe, *Artificial Intelligence through Prolog,* Englewood Cliffs, N.J., Prentice-Hall, 1988.

[10] M. E. Stickel, "Automated deduction by theory resolution", Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, CA, August 1985, 1181-1186.

[11] S. Su and G. Ozsoyoglu, "Data Dependencies and Inference Control in Multilevel Relational Database Systems", Proceedings of the IEEE Symposium on Security and Privacy, Oakland CA, 1987.

[12] R. P. Trueblood, "Security issues in knowledge systems", Proceedings of the First International Workshop on Expert Database Systems, Kiawah Island, South Carolina, October 1984, 834-841.

[13] L. Wos, *Automated Reasoning: 33 Basic Research Problems,* Englewood Cliffs, N. J., Prentice-Hall, 1988.

[14] L. Wos, R. Overbeek, E. Lusk, and J. Boyle, *Automated Reasoning: Introduction and Applications,* Englewood Cliffs, N. J., Prentice-Hall, 1984.

Appendix: The program kernel

```
/* Resolution theorem-prover with probabilities and time intervals */

/* Codes in this program: C: clause (as a list), P: probability, */
/* T: time interval, NL: list of clause number-probability pairs, */
/* the ancestors of a clause, B: starting time (integer), */
/* E: ending time (integer), PL: probability list */
/* X,Y: arbitrary items, L: arbitrary list */

go :- setup, not(go2), save(savepostgo2), setup3, combine_all_copies, writeall.
go :- write('*** first go definition failed ***'), nl.
go2 :- resolution(A,B,C), !, go2.
setup :- asserta(clausenumber(1)), setup2, retract(clausenumber(N)).
setup2 :- cl(C,P,T), clausenumber(N), not(clauze(C,P,T,[[N,P]])),
  asserta(clauze(C,P,T,[[N,P]])), retract(clausenumber(N)), Np1 is N+1,
  asserta(clausenumber(Np1)), fail.
setup2 :- setup3.
setup3 :- clauze(C,P,T,NL), fix_subsets(C,P,T,NL), fail.
setup3.
resolution([C1,P1,T1,NL1],[C2,P2,T2,NL2],[Cnew,Pnew,Tnew,Nnew]) :-
  twocs([C1,P1,T1,NL1],[C2,P2,T2,NL2]), matched_items(C1,C2,C1item,C2item),
  interval_overlap(T1,T2,Tn), delete(C1item,C1,C1d), delete(C2item,C2,C2d),
  union(C1d,C2d,Cnew), not(tautology(Cnew)), not(inferred_clause(Cnew,Tn)),
  union(NL1,NL2,Nnew), probcompute(Nnew,Pnew),
  superset_test(Cnew,Pnew,Tn,Nnew,Tnew), fix_subsets(Cnew,Pnew,Tnew,Nnew),
  addclauze(Cnew,Pnew,Tnew,Nnew).
/* Finds the pair of opposites in the terms of the clauses */
matched_items(C1,C2,C1item,not(C1item)) :- member(C1item,C1),
  member(not(C1item),C2).
matched_items(C1,C2,not(C2item),C2item) :- member(not(C2item),C1),
  member(C2item,C2).

/* Checks whether the new clause found is already covered by an old clause */
inferred_clause(C,T) :- (clauze(C2,P2,T2,N2); usedclauze(C2,P2,T2)),
  subset(C2,C), !, inferred_clause2(C,T,C2,T2).
inferred_clause2(C,T,C2,T2) :- subset(C,C2), not(T=T2),
  interval_overlap(T,T2,T3), write('Multiple proofs found of clause '),
  write(C), nl, !.
inferred_clause2(C,T,C2,T2) :- interval_contains(T2,T), !.
inferred_clause2(C,T,C2,T2) :- interval_overlap(T,T2,T3), !,
  inferred_clause3(C,T,T3).
inferred_clause3(C,[B1,E1],[B2,E2]) :- B1<B2, E1=<E2, !,
  B2m1 is B2-1, inferred_clause(C,[B1,B2m1]).
inferred_clause3(C,[B1,E1],[B2,E2]) :- B2=<B1, E2<E1, !,
  E2p1 is E2+1, inferred_clause(C,[E2p1,E1]).
inferred_clause3(C,[B1,E1],[B2,E2]) :- B1<B2, E2<E1, !, B2m1 is B2-1,
  E2p1 is E2+1, inferred_clause(C,[B1,B2m1]), inferred_clause(C,[E2p1,E1]).
/* Probability calculation during resolution uses independence assumption on */
/* set of probabilities of starting clauses used, after deleting duplicates */
probcompute([[N,P]],P).
probcompute([[N,P1]|NPL],P) :- probcompute(NPL,P2), P is P2*P1.
/* Revises those clauses already found when the new clause is more general */
superset_test(C,P,T,N,Tn) :- clauze(C2,P2,T2,N2), subset(C2,C),
  interval_overlap(T,T2,T3), !, superset_test2(C,P,T,N,C2,P2,T2,N2,Tn).
superset test(C,P,T,N,T).
```

```
superset_test2(C,Pn,T,N,C2,P2,T2,N2,T) :- interval_contains(T2,T), !, fail.
superset_test2(C,Pn,[B1,E1],N,C2,P2,[B2,E2],N2,Tn) :- not(subset(C,C2)),
  B1<B2, E1=<E2, B2m1 is B2-1, !, superset_test(C,Pn,[B1,B2m1],N,Tn).
superset_test2(C,Pn,[B1,E1],N,C2,P2,[B2,E2],N2,Tn) :- not(subset(C,C2)),
  B2=<B1, E2<E1, E2p1 is E2+1, !, superset_test(C,Pn,[E2p1,E1],N,Tn).
superset_test2(C,Pn,[B1,E1],N,C2,P2,[B2,E2],N2,Tn) :- not(subset(C,C2)),
  B1<B2, E2<E1, B2m1 is B2-1, E2p1 is E2+1,
  superset_test(C,Pn,[B1,B2m1],N,Tleft),  fix_subsets(C,Pn,Tleft,N),
  addclauze(C,Pn,Tleft,N), !, superset_test(C,Pn,[E2p1,E1],N,Tn).
superset_test2(C,Pn,[B,E],N,C2,P2,T2,N2,[B,E]).
fix_subsets(C,P,T,N) :- not(fix_subsets2(C,P,T,N)).
fix_subsets2(C,P,T,N) :- clauze(C2,P2,T2,N2), subset(C,C2), not(subset(C2,C)),
  interval_overlap(T,T2,T3), nice_retract(C2,P2,T2,N2),
  fix_subsets3(C,P,T,N,C2,P2,T2,N2), fail.
fix_subsets3(C,P,[B1,E1],N,C2,P2,[B2,E2],N2) :- B2<B1, E2=<E1, B1m1 is B1-1,
  addclauze(C2,P2,[B2,B1m1],N2), !.
fix_subsets3(C,P,[B1,E1],N,C2,P2,[B2,E2],N2) :- B1=<B2, E1<E2, E1p1 is E1+1,
  addclauze(C2,P2,[E1p1,E2],N2), !.
fix_subsets3(C,P,[B1,E1],N,C2,P2,[B2,E2],N2) :- B2<B1, E1<E2, B1m1 is B1-1,
  E1p1 is E1+1, addclauze(C2,P2,[B2,B1m1],N2), addclauze(C2,P2,[E1p1,E2],N2), !.

/* Checks for always-true clauses */
tautology(C) :- member(X,C), member(not(X),C).
/* Coalesces occurrences of the same clause on adjacent time intervals */
combine_all_copies :- clauze(C,P,T,N), combine_copies(C,P,T,N), fail.
combine_all_copies.
combine_copies(C,P,T,N) :- clauze(C2,P2,T2,N2), subset(C,C2),
  subset(C2,C), not(T=T2), interval_union(T,T2,Tn), !, union(N,N2,Nn),
  orprobcombine(P,P2,T,T2,Pn), nice_retract(C,P,T,N),
  nice_retract(C2,P2,T2,N2), asserta(clauze(C,Pn,Tn,Nn)),
  nice_write(C,Pn,Tn,Nn), combine_copies(C,Pn,Tn,Nn), !.
orprobcombine(P1,P2,[B1,E1],[B2,E2],P) :- T1 is E1-B1,
  T2 is E2-B2, weighted_average(P1,T1,P2,T2,P), !.
weighted_average(X,WX,Y,WY,A) :- A is (((WX*X)+(WY*Y)))/(WX+WY).

/* Time reasoning functions */
interval_overlap([B1,E1],[B2,E2],[B3,E3]) :-
  B2=<E1, B1=<E2, max2(B1,B2,B3), min2(E1,E2,E3).
interval_union([B1,E1],[B2,E2],[B1,E2]) :- B2 is E1+1, !.
interval_union([B1,E1],[B2,E2],[B2,E1]) :- B1 is E2+1, !.
interval_union([B1,E1],[B2,E2],[B3,E3]) :-
  E1>=B2, E2>=B1, min2(B1,B2,B3), max2(E1,E2,E3).
interval_contains([B1,E1],[B2,E2]) :- B1=<B2, E2=<E1.
/* Other utility functions */
addclauze([C],P,T,N) :- !, asserta(clauze([C],P,T,N)), nice_write([C],P,T,N).
addclauze(C,P,T,N) :- assertz(clauze(C,P,T,N)), nice_write(C,P,T,N).
nice_retract(C,P,T,N) :- write('Retract: '), clause_write(C,P,T,N),
  assertz(usedclauze(C,P,T)), retract(clauze(C,P,T,N)).
writeall :- nl, write('@@@@@The following clauses are in the database:'),
  nl, fail.
writeall :- clauze(C,P,T,N), nice_write(C,P,T,N), fail.
writeall.
nice_write(C,P,T,N) :- write('ASSERT: '), !, clause_write(C,P,T,N).
clause_write([X],P,[B,E],N) :- !, write(X), nl,
  write('  with probability '), write(P), write(' from time '),
  write(B), write(' to '), write(E), write(' using clauses '),
  write(N), nl.
```

```
clause_write([X|L],P,T,N) :- write(X), write(' OR '), clause_write(L,P,T,N).
member(X,[X|L]).
member(X,[Y|L]) :- member(X,L).
union([],L,L).
union([X|L],L2,L3) :- member(X,L2), !, union(L,L2,L3).
union([X|L],L2,[X|L3]) :- union(L,L2,L3).
delete(X,[],[]).
delete(X,[X|L],M) :- !, delete(X,L,M).
delete(X,[Y|L],[Y|M]) :- delete(X,L,M).
subset([],L).
subset([X|L1],L2) :- member(X,L2), subset(L1,L2).
twocs([C1,P1,T1,[[N1,P3]|NL1]],[C2,P2,T2,[[N2,P4]|NL2]]) :-
  clauze(C1,P1,T1,[[N1,P3]|NL1]), clauze(C2,P2,T2,[[N2,P4]|NL2]),
  N1=<N2.
append([],L,L).
append([X|L],L2,[X|L3]) :- append(L,L2,L3).
min2(X,Y,Y) :- Y<X, !.
min2(X,Y,X).
max2(X,Y,Y) :- Y>X, !.
max2(X,Y,X).
```

```
/* Example facts */
cl([course_change],1,[78,79]).
cl([maintain_course],1,[-10000,77]).
cl([maintain_course],1,[80,10000]).
cl([communications(to_ship)],0.9,[50,10000]).
cl([coast_nearby],1,[-600,1200]).
cl([not(communications(from_ship))],0.9,[-10000,10000]).
cl([not(bad_weather_ahead)],1,[-1000,1000]).
cl([not(military)],0.95,[-10000,10000]).
cl([not(zigzagging)],0.8,[-800,800]).
cl([not(other_ship)],0.95,[-500,500]).
cl([other_ship],1,[501,10000]).
/* Top level */
cl([course_change,not(avoidance)],1,[-10000,10000]).
cl([course_change,not(ordered(course_change))],1,[-10000,10000]).
cl([avoidance,not(obstacles_in_front)],1,[-10000,10000]).
cl([avoidance,not(yield_right_of_way)],1,[-10000,10000]).
cl([avoidance,not(bad_weather_ahead),not(communications(to_ship))],1,[-10000,10000]).
cl([avoidance,not(military),not(radar_activity),not(other_ship)],0.7,[-10000,10000]).
cl([ordered(course_change),not(manuevers)],0.8,[-10000,10000]).
cl([ordered(course_change),not(sensor_placing)],0.8,[-10000,10000]).
/* Intermediate clauses */
cl([obstacles_in_front,not(grounding_danger),not(coast_nearby)],1,[-10000,10000]).
cl([obstacles_in_front,not(grounding_danger),not(reefs_nearby)],1,[-10000,10000]).
cl([yield_right_of_way,not(other_ship),not(other_ship_has_right_of_way)],1,[-10000,10000]).
cl([communications(from_ship),not(ordered(course_change))],1,[-10000,10000]).
cl([zigzagging,not(military),not(sensor_placing)],1,[-10000,10000]).
cl([military,not(ordered(course_change))],1,[-10000,10000]).
cl([military,not(radar_activity)],1,[-10000,10000]).
/* Closed-world and uniqueness clauses */
cl([avoidance,ordered(course_change),not(course_change)],1,[-10000,10000]).
cl([obstacles_in_front,yield_right_of_way,bad_weather_ahead,
   radar_activity,not(avoidance)],1,[-10000,10000]).
cl([manuevers,sensor_placing,not(ordered(course_change))],1,[-10000,10000]).
cl([military,not(ordered(course_change))],1,[-10000,10000]).
cl([maintain_course,course_change],1,[-10000,10000]).
cl([not(maintain_course),not(course_change)],1,[-10000,10000]).
cl([not(avoidance),not(ordered(course_change))],1,[-10000,10000]).
cl([not(yield_right_of_way),other_ship],1,[-10000,10000]).
cl([not(yield_right_of_way),other_ship_has_right_of_way],1,[-10000,10000]).
```

Go to paper index