5/0-61
136884
P 7

SOFTWARE DESIGN AS A PROBLEM IN LEARNING THEORY

(A Research Overview)

Leona F. Fass

N93-17509

Introduction: Background and Motivation

Our interest in automating software design has come out of our research in automated reasoning, inductive inference, learnability and algebraic machine theory. We have investigated these areas extensively, in connection with specific problems of language representation, acquisition, processing and design.

In the case of formal context-free (CF) languages we established existence of finite learnable models ("behavioral realizations") and procedures for constructing them effectively. We also determined techniques for automatic construction of the models, inductively inferring them from finite examples of how they should "behave". These results were obtainable due to appropriate representation of domain knowledge, and constraints on the domain that the representation defined.

It was when we sought to generalize our results, and adapt or apply them, that we began investigating the possibility of determining similar procedures for constructing correct software. Discussions with John Cherniavsky, Dick Hamlet and Elaine Weyuker led us to examine testing and verification processes, as they are related to inference, and due to their considerable importance in correct software design. Motivating papers by Cherniavsky [1], Hamlet [3], Weyuker [4] and also, Fetzer [2], led us to examine these processes in some depth.

Here we present our approach to those software design issues raised in [1-4], within our own theoretical context. We describe our results, relative to those of [1-4] and conclude that they do not compare unfavorably.

Our Approach To Software Design

We approach problems of software design as examples or applications of a general learning theory. Our perspective is logical and algebraic: to us, a program or system fulfilling a specification S is "just like" any other realization of a specified behavior. The process of constructing software to perform a particular function or set of tasks, thus is an instance of synthesizing a behavioral realization. The testing of given software for incorrectness, or its verification as correct, are cases of checking a potential model, or realization, against its behavioral domain. If it is determined to exhibit all "good behavior" (positive domain data, as specified by S) and no "bad behavior" (negative data, i.e., the *complementary* domain elements, relative to S) the software is then established as correct.

Within our theoretical framework, successful software design requires analysis of desired behavior for identification of its essential components, and a means of defining--often through constraints--the domain in which the behavior lies. This knowledge must be represented and conveyed to the design system: an algorithm or technique for converting the knowledge into an implementation. Should designed software be *given*, then the knowledge might be conveyed to a testing/verification system to determine correctness of the design. If incorrectness were detected, errors could be removed and flaws repaired. The theoretical system need only reiterate these steps until it conclusively determined the software to be defect-free.

In each of these aspects of software design, our theory assesses as successful a process that is proven to terminate effectively (many would also demand efficiency), determining correct software as its end-product. This implies that all possible behavior must be conveyed finitely; that algorithms and techniques for construction, testing or verification of software operate in finite time and space; and that each process concludes, producing a resultant finite behavioral model.

If the above can be achieved it is a small step from effective determination of correct software to its automated determination or, design. We need only implement the algorithm or technique for the software construction, testing or verification, to create an automated "design system". Then we need only define an appropriately characterizing finite selection of behavioral data that the "system" may use to automatically determine a correct software design. To do so, we might adapt those techniques we devised to find correct language models [5-8], so that instead they produce software that behaves correctly, as specified.

Once a "design system" is implemented, it should be possible for an application specialist to provide it with domain-specific behavior examples. The system should then observe and generalize, to automatically determine software that realizes, or produces, the correct domain behavior in its entirety. At first, this appears to work very well, in theory.

However, our theoretical perspective leads us to examine software design problems somewhat more carefully, relative to those algebraic, constrained problem domains within which we obtained our initial learning theory results. We next describe some of the relationships between our theory and actual practice.

Results, "Results" and Conclusions

While there are, indeed, many similarities between theoretical learning problems and those encountered in practice, what we mainly find is that the constraints that make problems solvable in *theory* do not, in *practice*, generally apply.

We began this research overview by describing our theoretician's perspective, and our interest in adapting or applying our specific learning theory results to the case of

(automated) software design. Within the framework of theory, we noted that software design is "just like" any other modelling process. E.g., if we can infer a grammar generating a language from suitable linguistic examples then, surely, we can infer a program to produce that same language, and be certain that it is correct.

All of the general results in learning theory that come out of our specific CF language learning research were made possible by appropriate knowledge representation, and domain constraints. These enabled us to determine finite realizability of the CF languages and, also, the conclusive effective testability of potential language models. When sufficiency of testing is established, and tests conclusively detect no incorrectness, we establish correctness of a model. We call this "verification by default" [6-9].

In the case of language learning, we were able to establish an inference/testing/verification paradigm [6-10] that could result in automatic design of language models, obtainable in a number of ways. We showed that if the language has a model inferable from a finite sample of positive domain data ("good behavior") then a potential model could be conclusively, effectively tested and thus might be verified, by default, as correct. What we established was that the domain sample of positive data sufficient for inference defined a similar sample of positive and negative data ("good and bad behavior") that was sufficient for conclusive, effective tests.

As Hamlet noted in [3] and in our discussions, and as we have confirmed, these results are dependent on characterizing all necessary behavioral information in a finite way. (Our domain constraints gave us finite realizability and decidable membership queries: we could determine what was good behavior *vs* what was not [6-10]).

While in any typical software design environment our domain constraints and conditions do not apply, we believe our theoretical results compare, not unfavorably, with those of other theoreticians. Cherniavsky [1] noted testing can do more than detect errors in software, and we showed one can test to show software is correct. Fetzer [2] claimed verification was "impossible" and we showed inferable models could be testable, and verified automatically, by default. Weyuker [4] described inference-based testing to establish an approximate method of determining equivalence of a program and its specification. We concur and believe our logical and algebraic approach, and some domain-specific imposed constraints, will result in approximately automated software design. This will improve upon techniques currently in practice.

REFERENCES

[1] Cherniavsky, J. C., "Computer Systems as Scientific Theories: A Popperian Approach To Testing", *Proc. of the Fifth Pacific Northwest Software Quality Conf.*, Portland (Oct. 1987), pp. 297-308.

[2] Fetzer, J. H., "Program Verification: The Very Idea", *CACM*, Vol. 31 (1988), pp. 1048-1063.

[3] Hamlet, R., "Special Section on Software Testing", *CACM*, Vol. 31 (1988), pp. 662-667.

[4] Weyuker, E. J., "Assessing Test Data Adequacy through Program Inference", *ACM Transactions on Programming Languages and Systems*, Vol. 5 (1983), pp. 641-655.

Relevant Publications and Presentations by the Author

[5] Fass, L. F., "Remarks on Inductive Inference and Testing", presented at the *Association for Symbolic Logic 89-89 Annual Meeting*, University of California, Los Angeles, January 1989. Abstracted in the *J. Symbolic Logic*, Vol. 55, No. 1 (March, 1990), p. 374.

[6] Fass, L. F., "A Common Basis for Inductive Inference and Testing", *Proc. of the Seventh Pacific Northwest Software Quality Conf.*, Portland, (Sept. 1989), pp. 183-200.

[7] Fass, L. F., "Acquiring Knowledge by Positive or Negative Means", presented at the *Association for Symbolic Logic 90-91 Annual Meeting*, Carnegie Mellon University, January 1991. Abstracted in the *J. Symbolic Logic*, Vol. 57, No. 1 (March 1992) pp. 356-357.

[8] Fass, L. F., "Learning Through Inductive Inference or Testing", *Proc. Florida Artificial Intelligence Research Symposium, Conf. on Machine Learning*, Cocoa Beach, (April 1991), pp. 176-180.

[9] Fass, L. F., "Inference, Testing and Verification", presented at *Ninth International Congress on Logic, Methodology and Philosophy of Science* and *Logic Colloquium 91*, Section on Foundations of Logic, Mathematics and Computer Science, Uppsala, Sweden, August 1991. Abstracted in *Congress Volume I*, p. 193.

[10] Fass, L. F., "Perfect Learning (More or Less)", to be presented at the 1992 Meeting of *The Society For exact Philosophy*, University of Southwestern Louisiana, Lafayette, May 1992. Extended version in preparation.

Leona F. Fass received a B.S. in Mathematics and Science Education from Cornell University and an M.S.E. and Ph.D. in Computer and Information Science from the University of Pennsylvania. Prior to obtaining her Ph.D. she held research, administrative and/or teaching positions at Penn and Temple University. Since then she has been on the faculties of the University of California, Georgetown University and the Naval Postgraduate School. Her research primarily has focused on language structure and processing; knowledge acquisition; and the general interactions of logic, language and computation. She has had particular interest in inductive inference processes, and applications/adaptations of inference results to the practical domain. She may be reached at

Mailing address: P.O. Box 2914
Carmel CA 93921