Program Synthesis is "Just" Another Learning Problem

Leona F. Fass

Abstract

We view processes of program synthesis as behavior realization problems, within the scope of computational learning theory. If potential program behavior could be finitely, *completely* specified, we believe we could infer and verify programs to be correct. But *realistically* lacking such complete specifications, we support current research in synthesis which we believe produces finite-state correct-program approximations.

Synthesis as Learning

Our interest in program synthesis and processes that might define and determine correct software has come out of our work in knowledge acquisition, inductive inference and automated machine design. Thus we approach software development processes algebraically and logically, as processes of knowledge modeling and behavioral realization. We consider problems of program-and-system design and development to be problems in computational learning theory.

To us, solving a *learning problem* involves discovery of a knowledge model or behavioral realization from some specification of how it should (or should not) "behave". We would prefer a specification that is complete, characterizing all behavioral possibilities. And we would consider a learning process to be perfectly successful if a model or realization might be found effectively, from finite behavioral information (even if the behavior to-be-modeled comprises an infinite set). The entire body of knowledge or behavior would be learned, through effective acquisition of a characterizing model. If multiple models could be found we might establish criteria to determine that which is "best". (E. g., easiest to construct? Most efficient? Simplest?)

If we could determine correct software fulfilling a specification we would "just" have another instance of learning. Success might depend on representing, conveying and realizing an infinite body of knowledge --- all "correct" and "incorrect" program and system behavior --- by finite means. Finding a correct program is "just" finding a "device" that behaves in a specified way. From program candidates, we might find a "best " (Fastest to construct? Most efficient to run? Having the simplest components?) result.

In our initial research we obtained domain-specific results for a completely-specified behavior, synthesizing a finite realization that we considered to be best. We determined that realization to be *learnable*, and since our goal then was to have the behavior learned, that goal was achieved once the realization was defined and synthesized. It was our hope to generalize and "scale up" our original learning techniques to the area of software development (Fass, 1989). As we describe below, we found we could adapt them, to overcome some of the obstacles we found in "real world" software domains.

Some Successful Synthesis

We had great success determining behavioral models in a specific (linguistic) domain where we could obtain perfect results. With domain constraints we found behavioral models and minimal (our criterion for "best") models by construction and by inductive inference from selected behavioral examples. Adapting the testing theory of (Cherniavsky 1987) we also found minimal behavioral models by iterative error-detection-and-correction, and the (default) verification of potential models, through effective, conclusive tests.

By analyzing the behavioral domain to determine (congruence) classes of elements, we found the corresponding components of the learnable minimal model. The finite behavioral examples from which it could be inferred also defined the data for the adequate behavioral tests. Thus we characterized *all possible* behavior finitely. Constraints we imposed to convey behavior in our original learning research actually *forced* our successful modeling results. For these made the design problem completely-specifiable, the behavior finitely-realizable (so a finite model exists), and membership decidable (so correct behavior is distinguishable from its complement). This enabled us to characterize all possible "correct" and complementary "incorrect" behavior by finite means.

In the next phase of our research we determined that our techniques and satisfying results were not restricted to the specific (linguistic) domain of our original investigations. We found these results to be domain-specific *only* in the sense that they are applicable in *any* behavioral domain that can be completely specified by a finite-state device (Fass 2000).

Realistic Synthesis in "Real-Life" Domains

Domain constraints enabled us to synthesize perfect behavioral models by imposing a finite-stateness on a (possibly infinite)behavioral domain. Constraints also enabled us to represent the *complement* of a specific behavior finitely. Thus it could be decided, for any domain element, whether that element belonged to the correct specified behavior, or to the complementary behavioral class.

But the constraints applied in our theoretical work do not exist in "real life" program synthesis or software development domains, and decisions we would like to make in assessing software correctness often cannot be made. For example, it is it is known that the behavioral equivalence of a program and its specification is generally undecidable. And, it is known that, in general, there is no adequate test set to determine conclusively if given software is correct or incorrect. We believe that the greatest obstacle to "perfect" software synthesis is that there is really no way to completely specify all correct program behavior (in all environments), or all the ways a program or software system can "go wrong".

Still we find our theoretical modeling results have utility in finding finite-state approximations that "come close" to perfect software models, obtained through logic-based techniques or otherwise. We can decompose behavioral domains to find components that are finite-state representable, and thus, inferable from behavioral examples. With corresponding "complementary" examples we can finite-state test potential models for incorrectness, to detect correctable errors. We believe that by synthesizing such approximating results, program-and-software designers and developers are obtaining the best results they can.

Related Research and Conclusions

Our approximating techniques are similar to those used by the practitioners of model-based design , model-checking, and the finite-state verification being employed today. Much of the model-based software design discussed in (Keller 1992) and (Khatib and Pecheur 2001) depends on analyzing the structure of a behavioral domain to specify an approximating finite-state model. Model-checking and finite-state verification, such as that described in (Khatib and Pecheur 2001) *forces* a finite-stateness onto a software specification or the potential behavior of a program. These techniques, devised to determine correctness, may only achieve such a goal *approximately*. But they *can* finitely detect specification errors. And they *can* determine precise counterexamples and behavioral traces, establishing that the potential behavior of a synthesized program may not be perfectly correct. Such discoveries can be critical, in processes of program and system design.

While it may be impossible to synthesize perfectly correct programs and software systems for all environments, we believe we can find feasible, approximate results that are constructed with sound mathematical foundations. Adaptation of our original results in behavioral analysis, finite-state modeling and synthesis, can help to achieve that goal.

Selected Referências

Cherniavsky, J.C., 1987, "Computer Systems as Scientific Theories: a Popperian Approach to Testing", *Proceedings of the Fifth Pacific Northwest Software Quality Conference*, Portland OR, October 1987, 297-308.

Fass, L. F. 1989, "A Common Basis for Inductive Inference and Testing", *Proceedings of the Seventh Pacific Northwest Software Quality Conference*, Portland OR, September 1989, 183-200.

Fass, L. F., 2000, "Establishing Software 'Correctness' by Logical and Algebraic Means", *Proceedings of the Fifth Joint Conference on Information Sciences*, Atlantic City NJ, February 2000, Vol. I , 639-642.

Keller, R.(Ed.) 1992, Notes of the AAAI Workshop on Automating Software Design: Domain Specific Software Design, San Jose CA July 1992. NASA Ames Research Center AI Research Branch TR FIA-92-18.

Khatib, L. and C. Pecheur (Eds.) 2001, *Papers from the AAAI Spring Symposium on Model-Based Validation of Intelligence*, Stanford University, March 2001. AAAI Press SS01-04.

Leona F . Fass received a B.S. in 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 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. Dr. Fass may be reached at

Mailing address: P.O. Box 2914
Carmel CA 93921
Email: lff4@cornell.edu