Automatic Inductive Synthesis of Functional Programs*

Nachum Dershowitz and Ely Pinchover

School of Computer Science
Tel Aviv University
Ramat Aviv, Tel Aviv 69978, Israel
{nachumd,elypinch}@tau.ac.il

Abstract. An automated system for the synthesis of functional programs, expressed as a convergent set of rewrite rules, is presented. Deductive and inductive rewriting techniques are combined to synthesize a program from a formal specification. Deductive consequences of an equational specification, and of equations expressing domain knowledge, are produced by a completion-based inference engine. The novel component of the system is its incorporation of inductive techniques to guess program clauses, which are then subjected to verification. This paper describes the heuristics that are employed for generalizing from a set of deductive consequences to form new equations. The proof-by-consistency method is then used to establish validity in the initial algebra of newly suggested program rules. In the same fashion, induction is used to guess lemmata needed in the proof and deduction is used to verify them. An experimental ML implementation, geared to programs dealing with natural numbers and recursive data structures, has been written and is described. The judicious combination of completion, heuristic generalization, and inductive theorem proving allows it to derive provably correct programs fully automatically.

1 Introduction

Rewriting [10, 23] is a very powerful method for dealing computationally with equations. Oriented equations, called rewrite rules, are used to replace equals by equals, always in one direction. The theory of rewriting centers on the concept of normal form, which is an expression that cannot be rewritten any further. Computations consist of rewriting to a normal form; when the normal form is unique, it is taken as the value of the initial expression. When a rewrite system always rewrites equal ground (variable-free) terms to the same normal form, it is said to be ground convergent.

We have implemented an automated program synthesis system, combining deduction and induction, based on old suggestions in [21, 9, 11]. Domain knowledge, specifications and programs are all expressed as equational identities, and a

* Research supported in part by the Israel Science Foundation (grant no. 254/01).

combination of equational inference and inductive generalization is employed to deduce programs. The derived pattern-directed functional program is composed of a set of oriented equations that can be used as a ground-convergent rewrite system to compute normal forms.

In our system, a modern version of the Knuth-Bendix completion procedure [14] serves as a deductive engine for producing new rules and proving theorems. Generalization is used to guess program statements and lemmas. We propose heuristics for generalizing from a sequence of deductive consequences. To prove them, we use the "inductionless induction" method, pioneered in [18]; see [5]. Test sets (see [5]) are used to determine when a complete program has been obtained.

Example 1 (Toy). The specification of a doubling function d is

$$d(x) = x + x \tag{1}$$

The domain knowledge is

$$x + 0 = x \tag{2}$$
$$x + s(y) = s \tag{3}$$

The result is

$$d(0) \to 0 \tag{4}$$
$$d(s(x)) \to s \tag{5}$$

a self-contained program for doubling. It has four important properties:

1. It is terminating (all evaluations lead to a normal form).
2. It is ground confluent (any variable-free term has at most one normal form).
3. It is correct (true in the initial model of the specification and domain equations).
4. It is complete (every term of the form $d(s^n(0))$ is reducible).

Early work on deriving programs from formal specifications includes [24, 17]; the use of equational reasoning for the creation of functional programs appears in [6, 4]. One early example of the use of induction and of generalization to guess program statements is [22]. The use of Knuth-Bendix completion [14] as a deductive engine for synthesis was suggested in [7]. We proposed combining equational synthesis with induction and generalization in [9]. The original Boyer-Moore theorem prover [3] incorporated similar use of inductive techniques. A recent collection on deductive program-synthesis methods is [12]; an older survey of inductive methods for synthesizing logic programs is [16].

2 The Method

The architecture of our synthesis system is as follows:

1. Initially, the database consists of equations expressing domain knowledge and requirements of the function to be synthesized.
2. Completion is used continuously to infer new equations.
3. Recursive path orderings are used to orient equations into rewrite rules and guide the prover.
4. The database is watched by a heuristic generalizer that suggests new equational hypotheses.
5. Structural induction is used to try to verify hypotheses.
6. Lemmas may be suggested by equations generated in the process.
7. Verified equations are continuously added to the database.
8. Test sets are used to determine when a ground convergent system for the function being synthesized has been obtained.

3 Implementation

The synthesis system described herein combines, for the first time, rewrite-based methods for deductive reasoning, inductive reasoning, and testing of sufficient completeness in one comprehensive framework. Inductive techniques are only used as hypotheses that are subjected to formal validation. The results are provably correct programs—derived automatically.

The program is written in ML, and makes heavy use of the rewriting code in [1]. It has been used to synthesize programs dealing with recursive term structures such as natural numbers, list and trees, including the examples in [9].

The program may be perused at [20].

Many variations on the above theme are possible:

- Unfailing completion [13, 2] and ordered paramodulation [19] should be used.
- We use inductionless induction ("proof by consistency"), since it fits the framework so beautifully; other inductive theorem provers could be used.
- Orderings other than the recursive path orderings [8] may be used to guide completion.
- Rippling heuristics [15] could be used to suggest new statements and lemmas.
- Other heuristics for generating "interesting" conclusions and for finding patterns in deduced rules should be experimented with.

We are exploring some of these directions, as well as theoretical issues raised by the use of auxiliary functions when reasoning in the initial model (cf. [25]).

References

1. Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
2. Leo Bachmair and Nachum Dershowitz. Equational inference, canonical proofs, and proof orderings. *J. of the Association for Computing Machinery*, 41(2):236–276, 1994.
3. R. Boyer and J. Moore. *A Computational Logic*. Academic Press, New York, 1979.
4. Robert M. Burstall and John Darlington. A transformation system for developing recursive programs. *J. of the Association for Computing Machinery*, 24(1):44–67, January 1977.
5. H. Comon. Inductionless induction. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 14, pages 913–962. Elsevier Science, 2001.
6. John Darlington. Applications of program transformation to program synthesis. In *Proceedings of the IRIA Symposium on Proving and Improving Programs*, pages 133–144, Arc-et-Senans, France, July 1975.
7. Nachum Dershowitz. Synthesis by completion. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 208–214, Los Angeles, CA, August 1985.
8. Nachum Dershowitz. Termination of rewriting. *J. of Symbolic Computation*, 3(1&2):69–115, February/April 1987. Corrigendum: *4*, 3 (December 1987), 409–410.
9. Nachum Dershowitz and Eli Pinchover. Inductive synthesis of equational programs. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 234–239, Boston, MA, July 1990. AAAI.
10. Nachum Dershowitz and David A. Plaisted. Rewriting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 9, pages 535–610. Elsevier Science, 2001.
11. Nachum Dershowitz and Uday Reddy. Deductive and inductive synthesis of equational programs. *J. of Symbolic Computation*, 15:467–494, 1993.
12. Bernd Fischer and Douglas R. Smith. Logic-based program synthesis: State of the art and future trends: Papers from 2002 aaai spring symposium. Technical report, AAAI Press, 2002.
13. J. Hsiang and M. Rusinowitch. On word problems in equational theories. In T. Ottmann, editor, *Proc. 14th ICALP*, Lecture Notes in Computer Science, pages 54–71, Berlin, 1987. Springer-Verlag.
14. Donald E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.
15. Ina Kraan, David A. Basin, and Alan Bundy. Middle-out reasoning for synthesis and induction. *Journal of Automated Reasoning*, 16(1-2):113–145, 1996.
16. N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.
17. Zohar Manna and Richard J. Waldinger. Toward automatic program synthesis. *Communications of the ACM*, 14(3):151–165, March 1971.
18. David R. Musser. On proving inductive properties of abstract data types. In *Proceedings of the Seventh ACM Symposium on Principles of Programming Languages*, pages 154–162, Las Vegas, NV, 1980.

19. Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. Elsevier Science, 2001.
20. Ely Pinchover. DeadEnd. `http://www.cs.tau.ac.il/~nachumd/deadend.ml`.
21. Uday S. Reddy. Rewriting techniques for program synthesis. In N. Dershowitz, editor, *Proceedings of the Third International Conference on Rewriting Techniques and Applications (Chapel Hill, NC)*, volume 355 of *Lecture Notes in Computer Science*, pages 388–403, Berlin, April 1989. Springer-Verlag.
22. Gerald Jay Sussman. *A Computer Model of Skill Acquisition*. Elsevier Science Inc., 1975.
23. "Terese" (M. Bezem, J. W. Klop and R. de Vrijer, eds.). *Term Rewriting Systems*. Cambridge University Press, 2003.
24. Richard J. Waldinger. *Constructing programs automatically using theorem proving*. PhD thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, May 1969.
25. Claus-Peter Wirth and Bernhard Gramlich. On notions of inductive validity for first-order equational clauses. In Alan Bundy, editor, *Proc. 12th Int. Conf. on Automated Deduction (ACDE'94)*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 162–176. Springer-Verlag, June/July 1994.