A Tutorial on Synthesis of Logic Programs from Specifications

Kung-Kiu Lau
Department of Computer Science, University of Manchester
Oxford Road, Manchester M13 9PL, United Kingdom
kung-kiu@cs.man.ac.uk

Geraint Wiggins
Department of Artificial Intelligence, University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, Scotland
geraint@aisb.ed.ac.uk

Abstract

1 Introduction

Program synthesis is concerned with deriving programs from their specifications. Such specifications (and corresponding derivations) may be *formal* or *informal*. Formal methods have an advantage over informal ones, in that they have the desirable[1] property of being able to formally prove the correctness of the derived programs.

Logic programming provides a uniquely uniform framework for specifications, programs and program synthesis. It is small wonder that in the early days of logic programming, program synthesis was one of the first topics that received attention. Most notable among this early work is that of Clark [2], Hansson [4], Hogger [6], and Tärnlund [5]. Although Hansson and Tärnlund and their colleagues at Uppsala implemented a derivation editor, the early work was not particularly geared towards automated synthesis. For some reason, work on logic program synthesis dwindled considerably in the Eighties. However, at the beginning of the Nineties, interest was rekindled, this time with a firm emphasis on automated or semi-automated synthesis. Now there is an annual international workshop on logic program synthesis and transformation.[2]

The tutorial is organised as follows. We shall first define the basic concepts. Then we shall describe a taxonomy of logic program synthesis methods. For the main approaches in the taxonomy, we shall give examples, with a view to illustrating the approaches as well as comparing them.

[1] Crucial, in the case of safety-critical applications.

[2] The relationship between synthesis and transformation is an intimate and intricate one. In this tutorial, we shall concentrate on synthesis, but we shall also see its relationship with transformation.

2 A Taxonomy

The first attempt at a taxonomy of logic program synthesis methods can be found in Deville and Lau [3]. This classifies methods as follows:

- Formal methods:
  - Constructive synthesis.

    The specification of a program P is a conjecture that for all inputs to P there exists an output that satisfies the input-output relation that P computes. A constructive proof of this conjecture is carried out, and then P is extracted from the proof.
  - Deductive synthesis.

    The specification is a set of if-and-only-if sentences (together with relevant axioms) defining the predicates of P. The clauses for P are deduced directly from the specification, for example by rewriting the sentences in the specification using unfold-fold transformation.

    An important sub-class of deductive synthesis is sometimes called *transformational synthesis*. Here the definitions are program clauses, and are transformed into equivalent clauses by unfold-fold transformations, or other transformations such as partial deduction.
  - Inductive synthesis.

    The specification is incomplete. Inductive synthesis derives P as a generalisation or completion of the specification. This whole area is now known as Inductive Logic Programming.
- Informal methods:

  These include methods such as general divide-and-conquer, stepwise refinement, schema-based program construction, and so on.

Owing to lack of space, we shall concentrate on formal methods, in particular constructive and deductive synthesis. Inductive synthesis, or Inductive Logic Programming, is now a research area in its own right, and is therefore best left outside the scope of this tutorial.

3 A Survey

Deville and Lau [3] also contains a brief survey of logic program synthesis methods. In this tutorial, we shall follow their style, and for each approach, we shall explain it with an example:

- For constructive synthesis, we shall base our description on the work by Bundy and Wiggins [1].

- For deductive synthesis, we follow that of Lau and Ornaghi [8]. For transformational synthesis by fold-unfold, we will describe the work of Sato and Tamaki [11]. For transformational synthesis by partial deduction, we will describe the work of Komorowski [7].

  Transformational synthesis exemplifies the intricate relationship between synthesis and transformation. Here we shall discuss this relationship, and explain the precise distinction and similarities between the two.

- For inductive synthesis, we can only briefly mention the work by Shapiro [10], and refer the readers to a comprehensive survey by Muggleton and De Raedt [9].

For each approach, we shall also survey other existing work and relate it to the chosen central example.

Furthermore, we shall also discuss the inter-relationships between the three approaches. In particular, constructive and deductive synthesis have been shown to be closely related to each other.

4 Conclusion

Logic program synthesis can be seen as a an important first step towards a logic-based approach to formal software development in any paradigm. The state-of-the-art already suggests that such a claim is not entirely groundless, although much work has yet to be done for the claim to be taken seriously. Nevertheless, we are confident that in due course logic programming will be leading the attack on the 'software crisis' that so far remains unresolved. We hope this tutorial will be a small contribution towards this cause.

References

[1] A. Bundy, A. Smaill, and G. Wiggins. The synthesis of logic programs from inductive proofs. In J.W. Lloyd, editor, *Proc. ESPRIT Symposium on Computational Logic*, pages 135–149. Springer-Verlag, 1990.

[2] K.L. Clark. The synthesis and verification of logic programs. Technical Report DOC 81/36, Imperial College, London, September 1981.

[3] Y. Deville and K.K. Lau. Logic program synthesis. *J. Logic Programming*. To appear in Special Issue on "Ten Years of Logic Programming".

[4] Å. Hansson. *A Formal Development of Programs*. PhD thesis, Dept of Information Processing and Computer Science, The Royal Institute of Technology and The University of Stockholm, 1980.

[5] Å. Hansson and S.-Å. Tärnlund. A natural programming calculus. In *Proc. IJCAI-79*, pages 348–355, 1979.

[6] C.J. Hogger. Derivation of logic programs. *J. ACM*, 28(2):372–392, April 1981.

[7] J.H. Komorowski. Synthesis of Programs in the Framework of Partial Deduction. Report Ser. A, No 81, Dept of Computer Science, Abo Akademi, 1989.

[8] K.K. Lau and M. Ornaghi. An incompleteness result for deductive synthesis of logic programs. In D.S. Warren, editor, *Proc.* 10^{th} *International Conference on Logic Programming*, pages 456–477. MIT Press, 1993.

[9] S. Muggleton and L. De Raedt. Inductive logic programming. *J. Logic Programming*. To appear in Special Issue on "Ten Years of Logic Programming".

[10] E.Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, 1983.

[11] T. Sato and H. Tamaki. Transformational Logic Program Synthesis. In *Proc. International Conference on Fifth Generation Computer Systems*, pages 195–201, 1984.