Journal of Symbolic Computation 69 (2015) 61–92

Contents lists available at ScienceDirect

Journal of Symbolic Computation

www.elsevier.com/locate/jsc

Synthesis of list algorithms by mechanical proving ☆

Isabela Drămnesc [a], Tudor Jebelean [b]

[a] West University of Timişoara, Romania
[b] RISC, Johannes Kepler University, Linz, Austria

ARTICLE INFO

Article history:
Received 7 February 2014
Accepted 25 June 2014
Available online 16 October 2014

Keywords:
List
Algorithm synthesis
Theorema

ABSTRACT

We address the automation of the processes of algorithm synthesis and systematic exploration of the theory of lists. Our focus is on methods, techniques, inference rules and strategies for computer based synthesis of list algorithms based on proving. Starting from the specification of the problem (input and output conditions), a *synthesis statement* is built: "for any list satisfying the input condition, there exists a list satisfying the output condition". The main difficulty is to find a constructive proof of this statement, from which the corresponding algorithm is easily extracted as a set of conditional equalities.

In more detail, we aim at computer automation of the proof of the existence of the sorted version of the input list. By using different proof methods we automatically synthesize five sorting algorithms: `Selection-Sort`, `Insertion-Sort`, `Quick-Sort`, `Merge-Sort`, and a novel algorithm, which we call `Unbalanced-Merge-Sort`, as well as the auxiliary functions used in the sorting algorithms. The theory we use is first order, and mostly contains formulae which are equivalent to Horn clauses. Therefore, except for induction, SLD resolution style inferences are in principle sufficient for performing the proofs. However, for most of the proofs this leads to a very large search space. Therefore we introduce several novel inference rules and specific strategies, which are based on the properties of lists, and which we developed in the course of this case study on sorting.

☆ The work reported in this article was partially supported by the European Commission grant No. FP7-REGPOT-CT-2011-284595 (HOST).

E-mail addresses: idramnesc@info.uvt.ro (I. Drămnesc), Tudor.Jebelean@jku.at (T. Jebelean).
URLs: http://web.info.uvt.ro/~idramnesc (I. Drămnesc), http://www.risc.jku.at/home/tjebelea (T. Jebelean).

http://dx.doi.org/10.1016/j.jsc.2014.09.030
0747-7171/

Moreover, during the process of algorithm synthesis we explore the theory of lists by introducing (automatically prove, and then use) the necessary properties.
When the knowledge base does not contain the auxiliary functions needed for the respective version of the algorithm, then the proof fails and from this failure a new proof goal is created, which is the synthesis statement for the missing auxiliary functions ("cascading").

1. Introduction

1.1. The problem

The algorithm synthesis problem is: starting from the specification of a problem, given as a pair of input and output conditions, how can we automatically discover an algorithm which satisfies the specification?

The concern of algorithm synthesis is to develop methods and tools for mechanizing and automatizing (parts of) the process of finding an algorithm that satisfies a given specification. There are several methods for algorithm synthesis, see Basin et al. (2004) where the authors classify the synthesis methods into: constructive/deductive synthesis, synthesis by transformation, inductive synthesis and schema based synthesis. Our approach, which we describe in this paper, is in the context of constructive synthesis.

Our motivation in choosing this problem is that the user has to describe only what the program should do (by giving the specification of the problem) and not how the program should work (not writing the code of the program). A constructive proof of the synthesis statement is performed and then the corresponding algorithm is extracted from the proof. Although constructive logic already gives comprehensive methods for extracting algorithms from proofs, it is still a challenge to actually find such proofs for concrete problems and to find proof techniques for these constructive proofs.

1.2. Related work

1.2.1. Synthesis methods

In Basin et al. (2004) the authors compare, by synthesizing a common program, three methods for recursive program synthesis, namely constructive/deductive synthesis, schema-based synthesis and inductive synthesis. The paper complements the survey of logic program synthesis from Deville and Lau (1994) and from Flener (2002).

In the constructive approach, also known as "proofs as programs" (see Constable, 1983; Bates and Constable, 1985) a conjecture generated from the problem specification is constructively proved and from the proof an algorithm is extracted. For some case studies in constructive synthesis, see Bundy et al. (1990), Fribourg (1990), Wiggins et al. (1991). In Howard (1980) the author presents the Curry–Howard isomorphism in constructive type theory, which in the context of constructive synthesis states that there exists a relationship between the constructive proof of an existential theorem and the corresponding extracted program.

In Bundy et al. (2006) the authors introduce some techniques for constructing induction rules for deductive synthesis proofs. These techniques are based on a combination of "rippling", see Bundy et al. (2005), and "middle-out reasoning", see Kraan et al. (1993).

As soon as a witness is found in the proof, one can extract the algorithm from proof, see Chiarabini (2008), Audebaud and Chiarabini (2009). Middle-out reasoning was used in order to synthesize algorithms for natural numbers, for lists. However, the authors point out in Kraan et al. (1996) that middle-out reasoning was not successful for the synthesis of sorting algorithms of list partitioning algorithms.