Journal of Symbolic Computation 90 (2019) 3–41

Contents lists available at ScienceDirect

Journal of Symbolic Computation

www.elsevier.com/locate/jsc

Mechanical synthesis of sorting algorithms for binary trees by logic and combinatorial techniques

Isabela Drămnesc [a], Tudor Jebelean [b], Sorin Stratulat [c]

[a] West University of Timişoara, Romania
[b] RISC, Johannes Kepler University, Linz, Austria
[c] Université de Lorraine, CNRS, LORIA, F-57000 Metz, France

ARTICLE INFO

Article history:
Received 5 March 2017
Accepted 25 June 2017

Keywords:
Algorithm synthesis
Automated reasoning
Natural-style proving

ABSTRACT

We develop logic and combinatorial methods for automating the generation of sorting algorithms for binary trees, starting from input-output specifications and producing conditional rewrite rules. The main approach consists in proving (constructively) the existence of an appropriate output from every input. The proof may fail if some necessary sub-algorithms are lacking. Then, their specifications are suggested and their synthesis is performed by the same principles.

Our main goal is to avoid the possibly prohibitive cost of pure resolution proofs by using a natural-style proving in which domain-specific strategies and inference steps lead to a significant increase of efficiency. In addition to classical techniques for natural-style proving, we introduce novel ones (priority of certain types of assumptions, transformation of elementary goals into conditions, special criteria for decomposition of the goal and of the assumptions), as well as methods based on the properties of domain-specific relations and functions. In particular, we use combinatorial techniques in order to generate possible witnesses, which in certain cases lead to the discovery of new induction principles. From the proof, the algorithm is extracted by transforming inductive proof steps into recursions, and case-based proof steps into conditionals.

E-mail addresses: isabela.dramnesc@e-uvt.ro (I. Drămnesc), Tudor.Jebelean@jku.at (T. Jebelean), sorin.stratulat@univ-lorraine.fr (S. Stratulat).

URLs: http://web.info.uvt.ro/~idramnesc (I. Drămnesc), http://www.risc.jku.at/home/tjebelea (T. Jebelean), https://members.loria.fr/SStratulat/ (S. Stratulat).

https://doi.org/10.1016/j.jsc.2018.04.002

The approach is demonstrated in parallel using the *Theorema* system, by developing the theory, implementing the prover, and performing the proofs of the necessary properties and synthesis conjectures. It is also validated in the *Coq* system, which allows to compare the facilities of the two systems from the point of view of our application.

1. Introduction

By **algorithm synthesis** we understand finding an algorithm which satisfies a given specification. We are working in the context of proof-based synthesis of functional algorithms, starting from their formal specifications. A formal specification is expressed as two predicates: the input condition $I[X]$ and the output condition $O[X, T]$.[1] The desired function F must satisfy the correctness condition $\underset{X}{\forall}(I[X] \Rightarrow O[X, F[X]])$, which corresponds to the *synthesis conjecture:* $\underset{XT}{\forall\exists}(I[X] \Rightarrow O[X, T])$. An algorithm which implements F can be extracted from the (constructive) proof of this conjecture. The algorithm is expressed as a list of *clauses* (conditional rewrite rules) of the form: $C[\overline{Z}] \Longrightarrow F[P[\overline{Z}]] = \mathcal{T}[\overline{Z}]$, where $\overline{Z}$ is a vector of variables, $P[\overline{Z}]$ is a pattern over these variables (with the property that it matches unambiguously certain elements of the domain), and $\mathcal{T}[\overline{Z}]$ is a term over the matching variables. The research presented here is focused on developing effective and efficient techniques for mechanizing the synthesis-proofs of sorting algorithms over the domain of binary trees, the synthesis-proofs of the auxiliary functions occurring in these algorithms, and the proofs of the additional properties which are necessary in the synthesis-proofs.

Our approach is experimental: we try various scenarios and techniques and refine them in order to obtain efficient proofs and various algorithms. The way the constructive proof is built is essential since the definition of the algorithm depends on it. For example, case splits may generate conditional branches and induction steps may produce recursive definitions. Hence, the use of different case reasoning techniques and induction principles may output different algorithms. The soundness of the proof rules and of the extraction procedure guarantee the correctness of the algorithm.

The focus of our work is on proof automation. In our experiments all the proofs are produced completely automatically, while the theory exploration (identification of all necessary auxiliary statements), the selection of the assumptions and of the induction principles used by the prover in each proof, and the construction of the conjectures from the failing proofs are performed manually.

The implementations of the new prover and extractor, as well as of the case studies presented in this paper are carried out in the frame of the *Theorema* system (Buchberger et al., 2016) which is itself implemented in Mathematica (Wolfram, 2003). *Theorema* offers significant support for automating the algorithm synthesis; in particular, the new proof strategies and inference rules have been quickly prototyped, tested and integrated in the system thanks to its extension features. Also, the proofs are easier to understand since they are presented in a human-oriented style. Moreover the synthesized algorithms can be directly executed in the system. The implementation files and the full proofs are presented in Dramnesc et al. (2015b). We additionally validate the results in the frame of the *Coq* system (Bertot and Casteran, 2004), by mechanically checking that the synthesized sorting algorithms satisfy the correctness conditions.

In parallel with the attempts to prove the conjectures corresponding to the synthesis problems, we explore the theory of binary trees, that is we introduce the necessary axioms and definitions, and we develop the necessary properties and prove them automatically. The full necessary theory is explored in *Theorema*, while in the *Coq* system, we only introduce the notions and properties necessary for the process of certifying the synthesized sorting algorithms.

[1] The square brackets are used for function and predicate applications instead of round brackets.