# Technical Commentary

Martin C. Rinard
MIT EECS and CSAIL
{rinard}@csail.mit.edu

Understanding how to get a computer to perform a given task is a central question in computer science. For many years the standard answer to this question has been to use a programming language to write a program that the computer will then execute to accomplish the task.

An intriguing alternative, however, is to provide the computer with examples of inputs and corresponding outputs, then have the computer automatically generalize the examples to produce a program that performs the desired task for all inputs. Researchers have worked on this approach for decades, first in the LISP community [4], then later in the inductive logic programming community [1–3], to name two prominent examples. Given the relatively modest size of the programs that the resulting techniques are able to produce, the field has evolved to focus largely on data mining, concept learning, knowledge discovery, and other applications (as opposed to mainstream software development).

The present paper focuses on an important emerging area — end user programming. As information technology has come to permeate our society, broader and broader classes of users have developed the need for more sophisticated data manipulation and processing. While users in the past were satisfied with relatively simple interactive models of computation such as spreadsheets and other business applications, current users are now looking to automate custom data manipulations such as reformatting, reorganizing, simple calculations, or data cleaning. While such users may have a good command of the interactive functionality of their application, they often lack the expertise, time, or inclination to develop software specifically for their task.

The present paper shows how to apply example-driven program synthesis to automate spreadsheet computations. It is therefore of interest to the millions of people who use spreadsheets worldwide. The methodology consists of four basic steps:

- **Domain-Specific Language:** Develop a domain-specific language capable of representing the desired set of computations.
- **Data Structure:** Develop a data structure that can efficiently represent the large set of programs that are consistent with a given input/output example.
- **Learn and Intersect:** Generate data structures for representing the programs consistent with each individual input/output example, then intersect the data structures to obtain a representation of the programs consistent with all examples.
- **Rank:** Rank the resulting set of programs, preferring more general programs over less general programs. Users can then view the results of the ranked programs on different inputs to guide the program selection process.

This approach effectively addresses many of the issues that complicate example-driven approaches. Domain-specific languages help focus the synthesis process by avoiding the generality of standard programming languages (which can produce very large program search spaces that are intractable to manipulate efficiently). Compact program representations make it possible to manipulate large numbers of programs efficiently. An effective ranking algorithm helps users quickly identify a desirable program (out of the potentially unbounded number of programs that are consistent with the provided examples). And the interactive program evaluation mechanisms (automatic identification of inputs on which candidate programs produce different outputs) help users navigate the space of synthesized programs.

The authors have successfully applied this approach to three classes of spreadsheet programs: syntactic string manipulations (such as converting telephone numbers into a standard format), semantic manipulations that operate on data stored in relational tables (such as transforming dates or strings in inventory tables), and table layout computations (which reorganize data stored in tables). All of these systems have been successfully integrated with Microsoft Excel and have been tested on multiple examples from Excel help forums.

In the future, the need for users to obtain ever more sophisticated custom behavior will only increase. Given the significant obstacles that traditional programming approaches pose for the vast majority of users, we can expect to see a proliferation of solutions that help non-programmers accomplish software development tasks that have traditionally been the exclusive domain of software professionals. Given the difficulty of specifying and implementing large software systems, these solutions will (at least initially) focus on the automatic generation of relatively small but still useful solutions to everyday problems. The present paper provides an outstanding example of the kinds of useful solutions that non-programmers can now automatically obtain and demonstrates the kind of sophisticated implementation techniques that will make such automatic program synthesis systems feasible for a variety of problem domains.

## References

[1] Nada Lavrac and Saso Dzeroski. *Inductive Logic Programming — Techniques and Applications*. Ellis Horwood, New York, 1994.

[2] Stephen Muggleton and Luc De Raedt. Inductive logic programming: theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.

[3] Stephen Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter A. Flach, Katsumi Inoue, and Ashwin Srinivasan. ILP turns 20 - biography and future challenges. *Machine Learning*, 86(1):3–23, 2012.

[4] Douglas R. Smith. The synthesis of LISP programs from examples: a survey. In A.W. Biermann, G. Guiho, and Y. Kodratoff, editors, *Automatic Program Construction Techniques*, pages 307–324. Macmillan, 1984.