Applications of Program Synthesis to End-User Programming and Intelligent Tutoring Systems*

Sumit Gulwani
Microsoft Research, Redmond
sumitg@microsoft.com

ABSTRACT

Computing devices have become widely available to billions of end users, yet a handful of experts have the needed expertise to program these devices. Automated program synthesis has the potential to revolutionize this landscape, when targeted for the right set of problems and when allowing the right interaction model. The first part of this talk discusses techniques for programming using examples and natural language. These techniques have been applied to various end-user programming domains including data manipulation and smartphone scripting. The second part of this talk presents surprising applications of program synthesis technology to automating various repetitive tasks in Education including problem, solution, and feedback generation for various subject domains such as math and programming. These results advance the state-of-the-art in intelligent tutoring, and can play a significant role in enabling personalized and interactive education in both standard classrooms and MOOCs.

Categories and Subject Descriptors

D.1.2 [**Software**]: Programming Techniques—*Automatic Programming*; K.3.1 [**Computing Milieux**]: Computers and Education—*Computer Uses in Education*

Keywords

Programming by Examples, Computer-aided Education

1. INTRODUCTION

Program synthesis is the task of automatically synthesizing a program in some underlying domain-specific language (DSL) from a given specification using some search technique [8]. Program synthesis has the potential to revolutionize the computing landscape, when targeted for the right set of problems and using the right interaction model. In this article, we discuss two such classes of applications.

*This article accompanies Sumit's GECCO'14 invited talk.

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.
ACM 978-1-4503-2881-4/14/07.
http://dx.doi.org/10.1145/2598394.2598397.

2. END USER PROGRAMMING

General-purpose computational devices, such as smartphones and computers, are becoming accessible to people at large at an impressive rate. Most end users of these devices are non-programmers and they need to create small, often one-off applications to automate repetitive tasks.

While the traditional view of program synthesis is that of synthesizing programs from complete logical specifications, we have developed end-user friendly techniques for synthesizing programs from examples [10] and natural language using the following methodology [11]: (a) Study help forums and conduct user studies to identify an important problem domain, (b) Design a DSL expressive enough to capture real-world tasks in the domain, but restricted enough to enable efficient synthesis, (c) Develop a synthesis algorithm, and (d) Rank the various programs returned by the synthesizer.

Programming by Example (PBE): We have developed PBE techniques for various data manipulation tasks. Data is available in documents of various types, e.g., text/log files, spreadsheets, and webpages. These documents offer great flexibility in storing and organizing hierarchical data by combining presentation/formatting with the underlying data model. However, this makes it hard to manipulate the underlying data. We have developed PBE techniques for syntactic string transformations [9] (this technology was shipped as the Flash Fill feature in Excel 2013), semantic string transformations [19], number transformations [20], and table transformations [14]. Besides transformation tasks, we have also invested in extraction [15] and formatting [18] tasks. Combining these technologies in a pipeline of extraction, transformation, and formatting can allow end users to perform sophisticated data manipulation.

Most of the underlying algorithms work by systematically reducing the problem of synthesizing a DSL expression (given input-output examples for that expression) to the problem of synthesizing the sub-expressions of that expression (by translating the examples for the expression to the examples for the sub-expressions).

Programming by Natural Language (PBNL): Some tasks such as filtering, summarization, actions with side-effects can often be best communicated using natural language (as opposed to using examples). There are two general approaches that we have used for understanding natural language based intent. One approach leverages natural language processing techniques along with type-based synthesis—this approach has been used to synthesize spreadsheet formulas [13] and smartphone automation scripts [16]. Another approach leverages the power of web search engines

to identify relevant code snippets from the web and then adapt them to the user's context—this approach was used in *Bing Code Search*, a Visual Studio 2013 add-in [1].

Future Directions: A key future direction is to develop general frameworks that can allow synthesizer writers to easily develop domain-specific synthesizers of the kind described in this article, similar to how declarative parsing frameworks allow a compiler writer to easily write a parser. We have already made some progress in this direction: The FlashExtract framework [15] allows easy development of synthesizers for extracting data from documents of various types such as text files, web pages, and spreadsheets. The Test Driven Synthesis framework [17] allows easy development of synthesizers for transforming various forms of data such as strings, tables, and XML. Another interesting future direction is to develop a multi-modal programming interface that allows easy integration of different PBE and PBNL technologies to accomplish sophisticated tasks.

3. INTELLIGENT TUTORING SYSTEMS

The search techniques used in program synthesis can help automate several repetitive and structured tasks in Education including generation of problems, solutions, and feedback [7]. These tasks can be automated for various subject domains including logic [2], automata theory [3], programming [22], arithmetic [5], algebra [21], and geometry [12, 4].

Problem Generation: Generating fresh problems that have specific solution characteristics (e.g., difficulty level, use of a certain set of concepts) is a tedious task for the teacher. Automating it can help prevent plagiarism (each student can be provided with a different problem with the same characteristics) and enable personalized workflows for students. [21] describes a PBE like technique for generating algebraic proof problems that are similar to a given problem. There is also recent work on generating fresh problems in geometry [4], natural deduction [2], and arithmetic [5].

Solution Generation: Solution generation is the task of automatically generating solutions given a problem description in some subject domain. [12] shows how to phrase the problem of generating a solution to a geometry construction problem as a program synthesis problem.

Feedback Generation: Feedback generation involves identifying whether the student's solution is incorrect and, if so, the nature of the error and potential fix. Automating it can save teachers time, and enable consistency in grading. It can also be used to provide immediate feedback to students thereby improving student learning. [22] shows how to phrase generation of a minimal fix to the student's program (in an introductory programming course) as a program synthesis problem. [3] suggests generating not only a minimal fix to the student's automata, but also a minimal change to the problem description where the new problem description corresponds to the student's (incorrect) solution.

Future Directions: The above-mentioned search techniques can be augmented with complementary techniques that leverage large amounts of student populations and data (facilitated by recent interest in online education). E.g., we can leverage student data to collect different correct solutions to a problem and use them to generate feedback [6] or to discover effective learning pathways to guide problem selection. We also ought to devise ways to quantify the benefits of computer-aided education on student learning.

4. REFERENCES

[1] Bing Code Search—Visual Studio 2013 add-in. `http://blogs.technet.com/b/inside_microsoft_research/archive/2014/02/17/bing-code-search-makes-developers-more-productive.aspx`.

[2] U. Ahmed, S. Gulwani, and A. Karkare. Automatically generating problems and solutions for natural deduction. In *IJCAI*, 2013.

[3] R. Alur, L. D'Antoni, S. Gulwani, D. Kini, and M. Viswanathan. Automated grading of DFA constructions. In *IJCAI*, 2013.

[4] C. Alvin, S. Gulwani, R. Majumdar, and S. Mukhopadhyay. Synthesis of geometry proof problems. In *AAAI*, 2014.

[5] E. Andersen, S. Gulwani, and Z. Popovic. A trace-based framework for analyzing and synthesizing educational progressions. In *CHI*, 2013.

[6] E. Fast, C. Lee, A. Aiken, M. S. Bernstein, D. Koller, and E. Smith. Crowd-scale interactive formal reasoning & analytics. In *UIST*, 2013.

[7] S. Gulwani. Example-based learning in computer-aided stem education. *To appear in CACM*.

[8] S. Gulwani. Dimensions in program synthesis. In *PPDP*, 2010.

[9] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *POPL*, 2011. `http://research.microsoft.com/users/sumitg/flashfill.html`.

[10] S. Gulwani. Synthesis from examples: Interaction models and algorithms. In *SYNASC*, 2012.

[11] S. Gulwani, W. Harris, and R. Singh. Spreadsheet data manipulation using examples. *CACM*, 2012.

[12] S. Gulwani, V. A. Korthikanti, and A. Tiwari. Synthesizing geometry constructions. In *PLDI*, 2011.

[13] S. Gulwani and M. Marron. NLyze: Interactive programming by natural language for spreadsheet data analysis and manipulation. In *SIGMOD*, 2014.

[14] W. R. Harris and S. Gulwani. Spreadsheet table transformations from examples. In *PLDI*, 2011.

[15] V. Le and S. Gulwani. FlashExtract: A framework for data extraction by examples. In *PLDI*, 2014.

[16] V. Le, S. Gulwani, and Z. Su. Smartsynth: Synthesizing smartphone automation scripts from natural language. In *MobiSys*, 2013.

[17] D. Perelman, S. Gulwani, D. Grossman, and P. Provost. Test-driven synthesis. In *PLDI*, 2014.

[18] M. Raza, S. Gulwani, and N. Milic-Frayling. Programming by example using least general generalizations. In *AAAI*, 2014.

[19] R. Singh and S. Gulwani. Learning semantic string transformations from examples. *PVLDB*, 5, 2012.

[20] R. Singh and S. Gulwani. Synthesizing number transformations from input-output examples. In *CAV*, 2012.

[21] R. Singh, S. Gulwani, and S. Rajamani. Automatically generating algebra problems. In *AAAI*, 2012.

[22] R. Singh, S. Gulwani, and A. Solar-Lezama. Automated feedback generation for introductory programming assignments. In *PLDI*, 2013.