Program Synthesis in the Industrial World: Inductive, Incremental, Interactive

Oleksandr Polozov
Univeristy of Washington
polozov@cs.washington.edu

Sumit Gulwani
Microsoft Research Redmond
sumitg@microsoft.com

1 Introduction

Programming by example (PBE), or *inductive synthesis* is a subfield of program synthesis where specification on a target program is provided in a form of input-output examples or, more generally, constraints on the output [9]. While research in PBE dates back as far as 1980s [8], its mass-market applications only appeared in the last decade. PBE has gained prominence due to its applicability in *end-user programming* – enabling end users to write small scripts for automating repetitive tasks from examples. The first PBE-based product was FlashFill – a system for automatic synthesis of string transformation scripts in spreadsheets [4]. It was incorporated in Microsoft Excel 2013, which made it available to hundreds of millions of end users. FlashFill was followed by FlashExtract, a system for automated data extraction from text files [6], which was deployed in Microsoft PowerShell and Azure Operational Management Suite. The team has since developed similar technologies for numerous data wrangling domains.

Deployment of a mass-market industrial application of program synthesis is a challenging process. First, an efficient implementation requires non-trivial engineering insights, which are often overlooked during development of a research prototype. Second, the development process should be fast and agile, tailoring to versatile requirements of the product/user base. Third, the underlying synthesis algorithm should be understandable to engineers responsible for product maintenance. Initial implementations of FlashFill and FlashExtract violated all these conditions, requiring more than a year of research & development done by PhDs or graduate students. This prompted us to generalize the ideas of FlashFill, FlashExtract, and their successors into FlashMeta – a framework for automatic generation of domain-specific inductive synthesizers [9]. FlashMeta made development of domain-specific PBE applications scalable, allowing us to generate a production-ready synthesis library in a few weeks instead of months.

In 2015, S. Gulwani's PROSE ("PROgram Synthesis using Examples") group at Microsoft [1] moved from Microsoft Research to Microsoft Data Group, establishing the first R&D group in PBE-based data wrangling at the company [3]. At Data Group, we are putting our algorithmic framework at the heart of new *synthesis APIs*, which enable various product teams to develop novel PBE features in their applications. Such an environment turns out fruitful for new problems and insights in program synthesis, which arise only in an *end-user facing* and *industrially deployed* applications. In this presentation, we want to share some of these problems, lessons, and potential solutions with the program synthesis community.

2 Problems

In this section, we outline a selection of research problems that arise during deployment of a PBE-based technology to mass markets. Most problems are cross-disciplinary, involving elements of HCI and AI. Many allow a variety of fundamental solutions, which we will discuss in more detail in the presentation.

Submitted to:
SYNT 2016

Ambiguity resolution Traditionally, the task of program synthesis has lied in finding *any* program in the underlying domain-specific language (DSL) that satisfies the specification. However, input/output examples are highly ambiguous. In a typical data wrangling task an initial specification, provided by an end user, allows up to 10^{20} conforming programs [9]. The real challenge lies not in finding them, but in picking the *right* program – the one that is more likely to be applicable on the rest of the user's data. Ranking is a crucial step in deployment: for instance, FlashFill only appeared in Excel after it was able to solve the most "obvious" transformation tasks from 1 example, before the users lost trust in the system. One way to resolve ambiguity between program candidates is *ranking* – applying a scoring function on programs. Such function can be specified manually by a DSL designer (with the right extension hooks provided by the framework) or learnt from a dataset [10]. Other solutions for ambiguity resolution exist, such as *interactive clarification* (discussed below).

Incremental synthesis When the system is unable to correctly learn the user's intent from one example, the user is forced to provide additional constraints. These constraints prompt an additional round of learning, which should take into account both the accumulated and the newly provided constraints. In the interest of superb end-user experience, iterative learning should be extremely fast – within 500 ms for most real-life scenarios. However, invoking a fresh program synthesis query with a large set of constraints is prone to increase learning time. Thus, we are forced to solve an *incremental synthesis* problem – given a set of solutions from the past user interactions and the new constraints for the same learning task, filter the set down to only those programs that satisfy the new constraints. With the set sizes ranging up to 10^{20} programs, this problem involves challenging research and engineering. Our solution involves a combination of background processing, "fast-track" tactics, and novel data structures for storing and processing DSLs.

Interaction An *interactive* program synthesis session generates suggestions and clarifying questions to the user in addition to a candidate program. In such a session the user plays the role of an *oracle*, and the system solicits their help *proactively* in the form of various queries [5]. In addition to ambiguity resolution, interactive PBE plays a huge HCI role: it establishes trust in the system, and creates unique *predictive* experiences [7]. Treating interactions as a first-class concept in a general-purpose PBE framework yields a number of challenging questions both in problem definition and implementation.

Performance-minded engineering A user-facing application should be responsive at all times, previewing operations in < 500 ms. However, program synthesis is a notoriously hard combinatorial search problem. For instance, the SyGuS-COMP 2015 competition picked a time limit of 3600 seconds of single-core CPU time across all its benchmarks [2]. In data wrangling domains, the required time typically ranges from milliseconds to minutes [6]. With a 500 ms response requirement, improving performance on a long tail of problems becomes a substantial engineering challenge.

Acknowledgments

This story with all its accomplishments would not be possible without the PROSE team: Allen Cypher, Ranvijay Kumar, Vu Le, Daniel Perelman, Mohammad Raza, Danny Simmons, Adam Smith, and Abhishek Udupa.

References

[1] *Microsoft Program Synthesis using Examples SDK*. `https://microsoft.github.io/prose`.

[2] Rajeev Alur, Dana Fisman, Rishabh Singh & Armando Solar-Lezama: *Results and Analysis of SyGuS-Comp '15*.

[3] Sumit Gulwani: *An Approach to Accelerated Innovation*. Available at `http://research.microsoft.com/en-us/um/people/sumitg/pubs/accelerated-innovation.pdf`.

[4] Sumit Gulwani (2011): *Automating string processing in spreadsheets using input-output examples*. In: *POPL*, 46, pp. 317–330.

[5] S. Jha & S. A. Seshia (2015): *A Theory of Formal Synthesis via Inductive Learning*. *ArXiv e-prints*.

[6] Vu Le & Sumit Gulwani (2014): *FlashExtract: A framework for data extraction by examples*. In: *PLDI*, ACM, p. 55.

[7] Mikaël Mayer, Gustavo Soares, Maxim Grechkin, Vu Le, Mark Marron, Oleksandr Polozov, Rishabh Singh, Benjamin Zorn & Sumit Gulwani (2015): *User Interaction Models for Disambiguation in Programming by Example*. In: *UIST*.

[8] Stephen Muggleton (1991): *Inductive logic programming*. *New generation computing* 8(4), pp. 295–318.

[9] Oleksandr Polozov & Sumit Gulwani (2015): *FlashMeta: A Framework for Inductive Program Synthesis*. In: *OOPSLA*, pp. 107–126.

[10] Rishabh Singh & Sumit Gulwani (2015): *Predicting a correct program in programming by example*. CAV.