Applications of Formal Methods to Data Wrangling and Education

Sumit Gulwani

Microsoft Corporation, Redmond, WA, USA
sumitg@microsoft.com

Data Wrangling

Data is the new oil. The digital revolution and evolution of social media, cloud computing, and IoT has led to massive amounts of digital data. This data is the new currency of the digital world since it can help drive business processes and decisions including advertising and recommendation systems. However, this data is locked up in semi-structured formats such as spreadsheets, text/log files, JSON/XML, webpages, and pdf documents. *Data wrangling* refers to the tedious process of converting such raw data to a more structured form that allows exploration and analysis for drawing insights. While data scientists spend 80% of their time wrangling data, programmatic solutions to data manipulation are beyond the expertise of 99% of end users who do not know programming. Programming by Examples (PBE) [6] can enable easier and faster data wrangling.

We have developed PBE technologies for many wrangling tasks including string/number/date transformations, extraction of tabular data from log files or webpages, and formatting or table layout transformations. Some of these technologies appear in mass-market industrial products. The FlashFill PBE technology [4] for string transformations ships as a feature in Excel 2013. The FlashExtract PBE technology [8] for extracting structured data out of log files ships as the ConvertFrom-string Powershell cmdlet in Windows 10 and the custom field extraction capability in Azure Operations Management Suite (OMS).

Our scalable algorithmic approach to synthesizing non-trivial scripts in real time involves two key ingredients from formal methods [11]: (a) restricting the search to an appropriate domain-specific language (DSL) and modeling inverse semantics of the DSL operators, which enables top-down propagation of goal-directed search obligations, (b) operations over grammars/languages (such as intersection, filtering) that enable computation of a sub-language of the underlying DSL s.t. each program in the sub-language is consistent with the examples. This sub-language sets up the structure for cross-disciplinary techniques to deal with ambiguity in the examples [10]. In particular, we use machine learning based ranking techniques to predict an intended program within this sub-language, and active-learning based interaction models to converge to an intended program.

Education

Formal methods can assist with repetitive tasks in Education including problem generation and feedback generation, for a variety of subjects including program-

ming, logic, mathematics, and language learning [5]. This can facilitate interactive and personalized education in both standard and online classrooms.

Problem generation: Generating fresh problems with specific solution characteristics (e.g., difficulty level, use of a certain set of concepts) is a tedious task for the teacher. Automating it can enable personalized workflows for students and help prevent plagiarism (each student can get a different problem with same characteristics). Test input generation techniques can help with generation of procedural problems [3], while template-based generalization techniques [12] and saturation-based reasoning [1] can help with generation of proof problems.

Feedback generation: This 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 enable providing immediate feedback to students thereby improving student learning. Counterexample generation can explain why a computational artifact like program, automata, or CFG [9] is incorrect. Repair techniques can help fix the student's incorrect solution [13] or predict whether the student misread the problem description [2]. Trace analysis can help generate strategy-level feedback [7].

References

1. U. Ahmed, S. Gulwani, and A. Karkare. Automatically generating problems and solutions for natural deduction. In *IJCAI*, 2013.
2. R. Alur, L. D'Antoni, S. Gulwani, D. Kini, and M. Viswanathan. Automated grading of DFA constructions. In *IJCAI*, 2013.
3. E. Andersen, S. Gulwani, and Z. Popovic. A trace-based framework for analyzing and synthesizing educational progressions. In *CHI*, 2013.
4. S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *POPL*, 2011.
5. S. Gulwani. Example-based learning in computer-aided stem education. *CACM*, Aug 2014.
6. S. Gulwani. Programming by examples (with applications to data wrangling). In *Verification and Synthesis of Correct and Secure Systems*. IOS Press, 2016.
7. S. Gulwani, I. Radiek, and F. Zuleger. Feedback generation for performance problems in introductory programming assignments. In *FSE*, 2014.
8. V. Le and S. Gulwani. FlashExtract: a framework for data extraction by examples. In *PLDI*, 2014.
9. R. Madhavan, M. Mayer, S. Gulwani, and V. Kuncak. Automating grammar comparison. In *OOPSLA*, 2015.
10. M. Mayer, G. Soares, M. Grechkin, V. Le, M. Marron, A. Polozov, R. Singh, B. Zorn, and S. Gulwani. User interaction models for disambiguation in programming by example. In *UIST*, 2015.
11. O. Polozov and S. Gulwani. Flashmeta: A framework for inductive program synthesis. In *OOPSLA*, 2015.
12. R. Singh, S. Gulwani, and S. Rajamani. Automatically generating algebra problems. In *AAAI*, 2012.
13. R. Singh, S. Gulwani, and A. Solar-Lezama. Automated feedback generation for introductory programming assignments. In *PLDI*, 2013.